8-2017

Physical Information Incorporated Clustering Techniques and Their Applications in Transportation Information System

Yang Zhang
University of Tennessee, Knoxville, yzhan157@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Yang Zhang entitled "Physical Information Incorporated Clustering Techniques and Their Applications in Transportation Information System." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Civil Engineering.

Lee Han, Major Professor

We have read this dissertation and recommend its acceptance:

Chris Cherry, Hairong Qi, Hyun Kim

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Physical Information Incorporated Clustering Techniques and Their Applications in Transportation Information System

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Yang Zhang

August 2017

Acknowledgements

There are so many people I must thank for making my graduate school experience so fruitful and memorable.

First I want to thank my advisor Dr. Lee Han, who accepted me three years ago and helped me through that toughest time so far in my life. Life is full of changes and when I look back, becoming Dr. Han's student is an episode I feel so privileged and grateful about. His passion for researching new technology, his free spirits, his resourcefulness, his dedication to help others and his artful presentation talents are infectious. When I came up with an ambitious cross-disciplinary idea, he applauded for it and encouraged me to pursue it. When I produced a small piece of result, he unearthed its essence, sharpened its value and reshaped it with a bigger story. When I struggled presenting my results, he patiently reworded what I wanted to say until I knew how to present technical results concisely and efficiently. When I needed help outside research, he always did more than I expected to clear the barriers. From Dr. Han, I learned not only how to conduct research, communicate ideas and results, but also how to be mature and professional.

I was also inspired by three other wonderful professors I have worked closely with. Dr. Hairong Qi mentored me on both my M.S. and PhD studies. I so much admire and have benefited from her profound expertise in algorithms and machine learning. It was my conversation with Dr. Qi that inspired me to apply clustering algorithms to solve vehicle routing problems. I have witnessed Dr. Chris Cherry's tireless devotion to making the transportation system in our community more friendly and sustainable.

His broad knowledge in traffic planning, design, energy, policy and his vision into the future not only made his classes always intriguing, but also strongly convinced me that transportation is a field worth to stay in. It is also my greatest honor to have been mentored by Dr. Hyun Kim, who is young but one of the most knowledgeable professors I ever knew. His dedication to research and his fearless pursuit of the most difficult problems have made him a great role model to me.

I would also like to thank all my fellow graduate students in the JDT 311 office. I am so grateful to have these wonderful colleagues in my day-to-day life. I have truly enjoyed every moment when we took classes together, discussed research ideas, participated activities, attended TRBs. Thank you Xiaobing Li, Ziwen Ling, Meng Zhang, Yuandong Liu, Zhihua Zhang, Bumjoon Bae, Hyeonsup Lim, Kay Boakye, Ali Boggs, Behram Wali, Pankaj Dahal, Saman Tabrizizadeh, Mohsen Kamrani, Nirbesh Dhakal. And a big thank you to our Big Sister Stephanie Hargrove for her thoughtfulness and leading the entire group.

I am also very thankful to the institutions and departments that have generously funded me during my study. They are the Southeastern Transportation Center, the Tennessee Department of Transportation, the Civil and Environmental Department, Department of Engineering Fundamental at the University of Tennessee. Without their sponsoring, I would never have been able to complete the program.

My roommate Ed Forshee has taken care of me in life, and helped me a lot get used to living in the U.S.. I am so blessed to have met him and become part of his family. Thank you to my friends James Fahey, Jim Gentry, Gerald Kamp, Landon Adam, for all the help you offered. My special thank you to my parents, sister and her family back in China. They lived in rural part of China, sacrificed everything just to support me through my education. Part of my determination to succeed comes from their unyielding faith and love in me, and my achievement is their achievement as well.

Abstract

Massive data from different sources are becoming available in transportation field, and spurring new research on utilizing these data to nurture new intelligent transportation information systems. Clustering algorithms are among the methods that are being applied to the domain but facing challenges. Classical clustering algorithms work fine with "point" based data, which are homogeneous and have no extra constraints. Data in transportation are sometimes involved with specific geometric shapes, have underlying constraints, and can be heterogeneous. There has been no clustering algorithm dedicated to these situations.

In this dissertation, we re-examine the mathematical foundation and underlying philosophy of hierarchical, density based, centroid-based clustering algorithms, and reformulate them to incorporate physical information to solve a big variety of transportation problems. In particular, we first show an example that a density-based data-driven geohash method can gain 40 seconds accuracy in ETA prediction. We then design a network space density-based clustering algorithm, Dijk-DBSCAN, which expands density based clustering from n-dimensional space to a transportation network space. We will show that Dijk-DBSCAN makes accident and other types of hotspot detection more accurate. Further, we present an online step-wise regression based clustering strategy, to collect vehicles' movement trajectory at low storage cost while maintaining high accuracy. We also explore clustering over arbitrary geometric shapes, and develop a hierarchical clustering framework. This algorithm can be applied to allocate resources over large road networks with an energy-efficiency

constraint. In the last section, we present a hierarchical clustering and greedy algorithm that solves general vehicle routing problems, with pickup and delivery, and with time windows. It handles mutually constrained location and time information by clustering orders (e.g. orders of package delivery, passengers' ride requests) and vehicles. The simple implementation and light computational cost make it superior to traditional optimization solvers, and enables real-time and large scale deployment in applications such as city-wide ride-sharing, time constrained package delivery, etc. Our work bridges the gap between classical clustering algorithms and specific data types and problem configurations in transportation domain. All our algorithms are designed to be highly computational efficient, and easily adaptable to other similar problems.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

We are living in an era where mobile technologies are connecting human to human, human to things, things to things in an unprecedentedly direct way. Location-based data produced through mobile devices, have offered transportation domain a whole lot of new opportunities than before. Before smart phones became popular, the main traffic sensors were fixed-location detectors, such as underground loop detectors, overhead Remote Traffic Microwave Sensors(RTMS) Schlaich et al. (2010); Systems (2016); Sun and Ban (2013). These sensors measure the volume, speed, and occupancy at a specific location, and monitor traffic conditions like congestion, accidents. Mobile sensors, e.g. the GPS-enabled devices, smart phones, tracks "when and where" people/vehicles are moving to, or records "when and where" an event/accident happens. Researchers are able to use their domain knowledge and statistical modeling techniques to infer "how and why" things happen. Not only this helps us understand how things happen, but also by leveraging historical and real-time geo-spatial information, we are able to project what might happen in the next moment. Instantaneous decisions can be made towards better mobility and safety purpose. There has been growing research into every aspect of how to utilize mobile sensor data to build a robust transportation information system. On the other hand, outside of academia world, we have seen in recent years, some technology companies, such as Uber, Lyft, Didichuxing, and many many others, are building businesses of delivering people, food, packages, etc, on top of the mobile technologies.

Mobile technology brings in both opportunities and challenges. The substantial difference between fixed-location detectors and mobile sensors means the methodologies utilizing their data are a lot different. On one hand, fixed-location detectors provide direct count information (i.e. vehicle counts, speed, volume, occupancy), while to get these information from mobile data, we need extra processing such as aggregation and statistical inference. If there is not enough mobile data available on a certain road, it is statistically unreliable to use mobile data. In that sense, we always have to consider penetration rate issue when using mobile data Herrera et al. (2010). On the other hand, mobile data contain far more useful information that fixed-location detector data simply cannot offer. Knowing where the vehicles are at the moment, we are able to infer traffic condition and travel time on the road network Hunter et al. (2009); Herring et al. (2010), we are able to forecast the demands Moreira-Matias et al. (2013), which can further support dispatching and dynamic pricing for taxi service, we are able to estimate origin-destination (OD) matrix Calabrese et al. (2011). Knowing where the accidents are happening, we are able to design less dangerous routes to guide people on safe traveling Shah et al. (2011); Kortge and Zhang (2005). Knowing where passengers are requesting their rides, a centralized ride sharing service can better match the drivers and passengers and provide the fastest service Cici et al. (2014). Mobile data are so flexible and enable us to do so many cool things.

The scope of this dissertation is on a class of method that has been broadly applied to process location-related transportation data: clustering algorithms. Clustering algorithms can be generally classfied as connectivity based clustering (e.g. Hierarchical Clustering), centroid based clustering (e.g. K-Means), density based clustering (e.g. Kernel Density Estimation (KDE), Mean Shift, DBSCAN), and other types in most recent literature. A few of these algorithms, such as K-Means and KDE, have been enormously applied to accident and safety analysis Anderson (2009); Aerts et al. (2006); Kim and Yamashita (2007); Prasannakumar et al. (2011), for congestion analysis Downs (2005); Anbaroglu et al. (2014).

While it is exciting to see the transportation community is applying these methods to extract using information and patterns from massive location-related data, it is also not hard to tell that "implanting" these methods to transportation domain is naive and has very limited room. On one hand, some of the publications simply force a transportation problem to fit the classical algorithm, while ignoring the constraints rooted in transportation: for example, when clustering accidents, people use K-Means on a two-dimensional space to find out the accident "hotspot region", while not considering the fact that all accidents happen on the road network. On the other hand, most of the classical clustering algorithms are designed to cluster on "Points", either physical points in a 2-dimensional or 3-dimensional space, or higher dimensional "points" that include more feature variables. There are many "Point"-typed data in transportation, e.g. accident locations. However, there are far more types of data that are not "Points", e.g. another very commonly used data type, trajectory. There is no directly available clustering methods for trajectories and other geometries, e.g. a road network itself, which is a set of mutually connected edges and vertices, is a type of data in transportation as well. Moreover, most clustering algorithms are not designed for data with constraints, e.g. accidents clustering with underlying network constraints. Most clustering algorithms are designed for homogeneous data, but information in transportation can be heterogeneous, for example, in vehicle routing problems with time windows, there is both location and time information, and there is constraints between locations and time.

To this point, we ask ourselves: can traditional clustering algorithms solve more complicated transportation problems beyond just clustering geo-locational points?

The motivation of this dissertation is to explore the answer to this question. Instead of reshaping a specific problem and dataset to fit the classical clustering algorithms, We look into those algorithms in a different way: we look at the mathematical foundation and philosophy of them, and seek for a possibility of reformulating the algorithm to incorporate various transportation related physical information. Amazingly, it has been quite a fruitful journey. We are proud to have found out that using the philosophy of density based clustering, hierarchical clustering, and other type of clustering, many of the problems involving various types of physical information will be effectively and efficiently addressed. Although these reformulated algorithms might look quite different from the original ones, they share the same underlying philosophy and are made to serve as broad a purpose as possible. Here is a quick introduction of the structure of the the rest of the dissertation: in Chapter 2, we present an appetizer-like example of applying data-driven density based geo-hash to quickly and fairly accurately provide ETA service. In Chapter 3, we present Dijkstra-DBSCAN, a network and density based clustering algorithm, useful for accident hotspot detection, and many other potential applications: congestion monitoring, etc. In Chapter 4, we present a clustering technique based on step-wise regression and time sequence segmentation, for compressed but effective vehicle trajectory data acquisition. In Chapter 5, we present a hierarchical clustering algorithm on road network, and apply it to road network coverage and resource distribution. In Chapter 6, we present a general greedy clustering strategy, for vehicle routing and ride-sharing/package delivery problems, and show that the clustering algorithm not only beat best published method on the same problem, but also support vehicle routing problems of any objective.

Chapter 2

An "Appetizer": Density-Based Data-Driven Goehash for Fast ETA Prediction

2.1 Introduction

As an "Appetizer" chapter, we present a simple example of incorporating physical information into a clustering algorithm. The physical information here is the density of traffic activities, e.g. there is more traffic in downtown NYC than in the suburb. We implement geo-hash based on the density information and gain extra accuracy in a Estimated Time of Arrival (ETA) prediction task. We are not trying to solve the well-investigated travel time prediction problem, which can be found in voluminous literature. Researchers have tried various time series prediction and machine learning methods, such as ARIMA, regression, nearest neighbor, random forest, neural network, etc Truong and Somenahalli (2011); Zhang and Rice (2003); Lam and Toan (2008); Li and Rose (2011); Shahsavari and Abbeel (2015). Although these models perform well on the segments chosen in those studies, we realize all those methods are data-hungry, i.e. requires too much features that probably is not practical for real-time and broad-scale deployment. For example, many of the models rely on loop detector data, but loop detectors are deployed only on a very limited number of main streets. In many applications such as vehicle dispatching, dynamic pricing, demand forecasting, we need very fast ETA predictions on the entire road network on a very frequent basis. In these cases, a five-second accuracy improvement might not be as critical as a fast-response, little data-consumption and of course fairly accurate algorithm. We hereby look at a less data consuming approach: predicting ETA using historical trip data. Each trip is recorded by only the origin and destination locations and time stamps.

2.2 Methodology

We break the map into cells. We maintain a three-dimensional Origin-Destination (OD) matrix $OD[i, j, t]$ to store the average speed from cell i to cell j at time interval t. Time interval can be every $15min$, $30min$, $1hour$, or even peak and non-peaj hour based on how rapidly the traffic changes over time in that region. We use our historical trip data to train this OD matrix. When there is a new trip, we simply determine the cell indices i, j and time interval index t and retrieve the estimated average speed for that trip $\hat{v} = OD[i, j, t]$, and calculate the distance s between the origin and destination of the new trip, then the estimated ETA is $\hat{t} = \frac{s}{OD[i,j,t]}$. Figure 2.1 shows the steps.

2.3 Density-Based Data-Driven Geohash

We compare two ways of geo-hashing the map: evenly split the map, or using a Quadtree, which splits the map based on the density of travels. Our data set comes from the NYC taxi data contributed by the NYC Taxi and Limousine Commission the NYC Taxi and Commission (2016). We randomly selected 10 Million trips for training, and 3 Million trip for testing. Figure 2.2 plots the origins and destinations

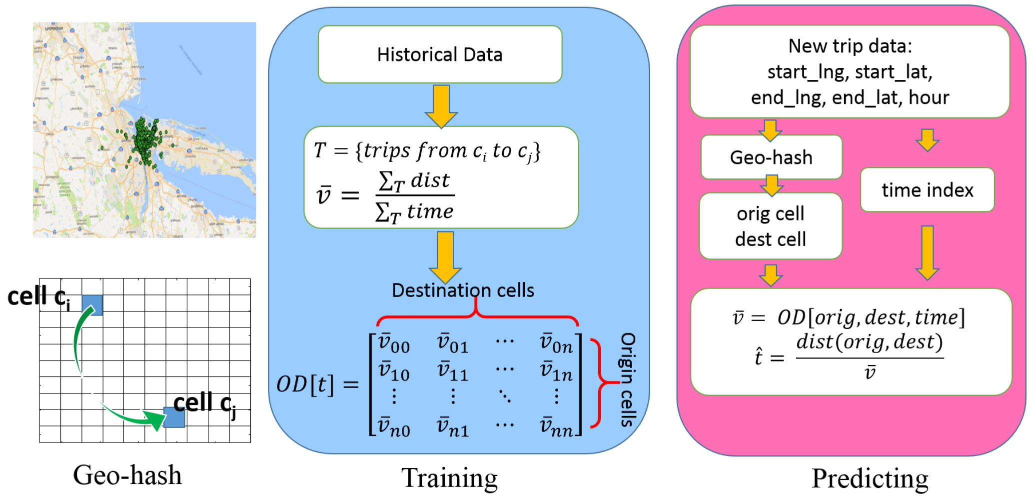


Figure 2.1: Fast ETA prediction using historical trips.

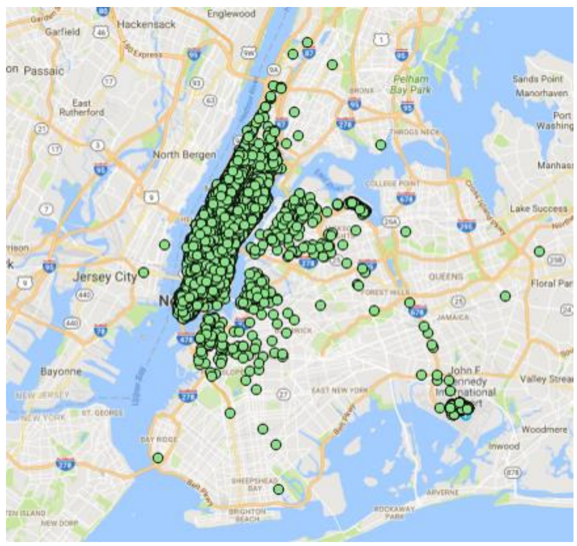
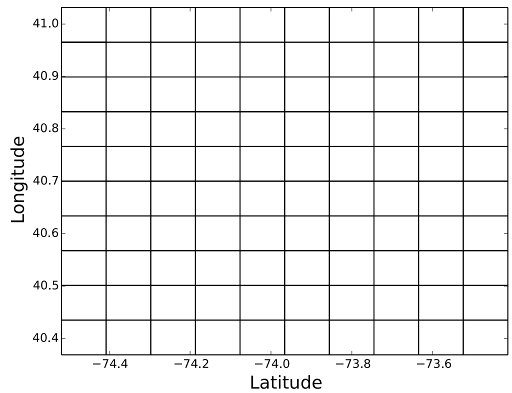


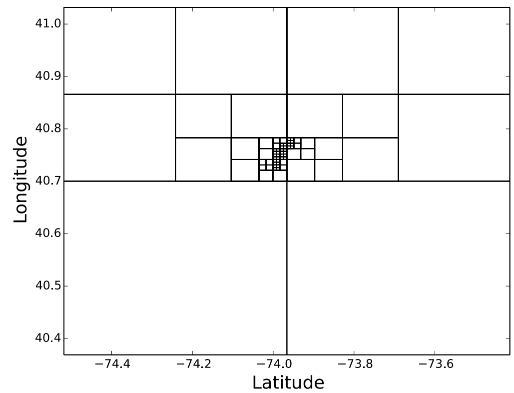
Figure 2.2: Fast ETA prediction using historical trips.

of the training trips. Figure plots the outlines of two geo-hashes. We can see that Quadtree captures the density of the trips, the more trips there are in the space, the smaller the cell will be.

Figure 2.4 show the performance comparison of the two methods. Density-based geohash seems to outperform evenly spaced geohash. This is not hard to understand because when splitting the map based on the density of the trip, a smaller cell will be used for a crowded area, and allows these small cells to keep their own features.



(a) Evenly Spaced



(b) Density-based Geohash

Figure 2.3: Two geohash methods: evenly spaced and Quadtree based.

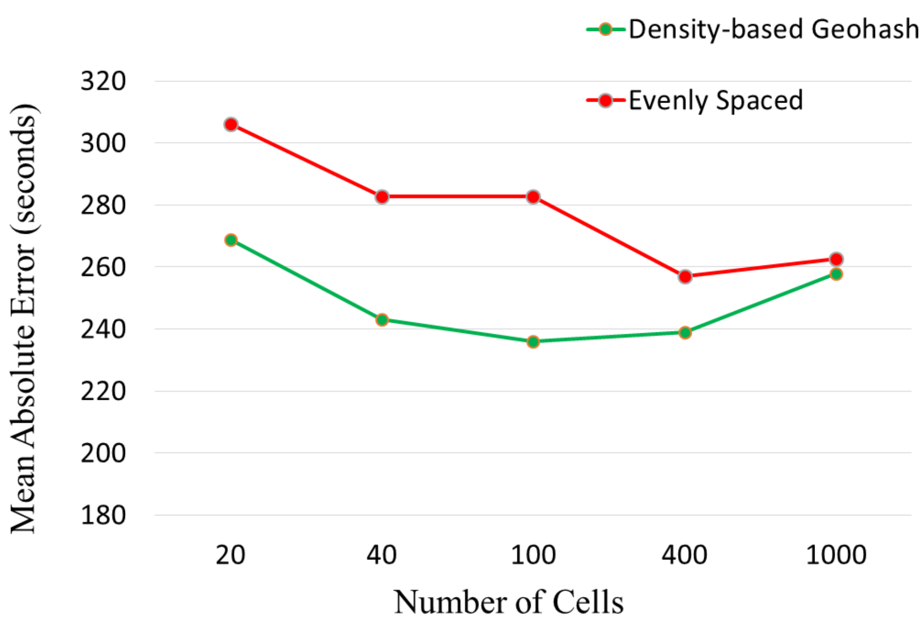


Figure 2.4: Prediction error comparison of density-based and evenly spaced geohashes.

In this case, 100 cells is the best resolution for the map, and density-based geohash is over 40 seconds more accurate than evenly spaced geohash.

Chapter 3

Dijkstra-DBSCAN: A Network Space Density-Based Clustering Algorithm for Accident Hotspot Detection

3.1 Introduction

Spatial hotspot identification benefits transportation management in many ways. Traffic incident hotspots is a powerful tool for lane safety management. It exposes dangerous road segments and becomes a direct indicator for the needs of road maintenance and infrastructure updates Prasannakumar et al. (2011); Truong and Somenahalli (2011); Bills (2009). To make decision on investment locations to achieve the most significant safety improvement with limited budget, the department of transportation has to rely on a reasonable cluster ranking method to evaluate the priority of the candidate sites. In addition to safety applications, growing availability of real time vehicle positional data from various sources, such as GPS, smart phones, RFID devices, blue tooth, is creating new opportunities of using spatial clustering techniques to study the traffic flows in both local and inter-regional scale. Vehicle trajectories clusters uncover "hot routes" and enable urban planners to optimize the allocation of traffic control devices Li et al. (2007), better understand congestions Wen et al. (2014); Anbaroglu et al. (2014); Montazeri-Gh and Fotouhi (2011), build and train models to dynamically predict traffic demands and reliable travel time Zhang et al. (2012); Won et al. (2009); Roh and Hwang (2010); Kim and Mahmassani (2015); Bartin et al. (2007). In a big data era, massive traffic data imposes remarkable challenges on traditional hotspot detection and spatial clustering methods: Conventional region based methods cannot provide the detailed roadway level information we expect for further analysis; an algorithm with high computational complexity will easily fail to handle larger amount of data; capability of parallelization becomes an advantage to utilize distributed computing resources; compatibility with large scale network allows a method to be applicable to broader spatial scale. It is worthwhile to rethink about such respects of classical approaches.

Two major classical hotspot detection methods are kernel density estimation (KDE) and clustering analysis Lawson (2010). KDE method Anderson (2009); Yang et al. (2003) divides the space into cells given a user-defined cell size. Density value of each cell is calculated by summing up the influences from other cells. Kernel function defines how the influence varies with distance, there are different types of kernel function, such as Gaussian kernel, quartic kernel, triangular kernel, minimum variance kernel, etc. Hotspots are identified as the regions with high accumulative density values. A variety of non-kernel clustering algorithms have also been applied to incident hotspot analysis: hierarchical Kidane and Bonds (2015); Phillips and Lee (2006), K-means Anderson (2009); Di Martino et al. (2008); Faria et al. (2014), DBSCAN Shirai et al. (2013); Lopez et al. (2012); Birant and Kut (2007), etc. Both hierarchical and K-means clustering require users to specify the number of clusters, which makes them impractical for a large dataset because a user barely visualize a large dataset Divya et al. (2014), not to mention to provide a reasonable estimation of number of clusters. DBSCAN were reported to have better performance and so adopted more often than others because of its simple parameters Divya et al. (2014).

One important fact that most previous related studies did not address is that these methods were all designed for general spatial analysis, where the space is two or three dimensional. A spatial hotspot refers to, for example, a neighborhood with high crime rates, a town experiencing massive disease infections, an area where biodiversity is endangered, etc. The fundamental difference between traffic related hotspots and general spatial hotspots is that the space of traffic events is neither two dimensional nor three dimensional, but the underlying road network. For one thing, a 2D region cluster is not a direct measure of the dangerous level of road segments. An urban region with dense road connections is not as dangerous as a rural region with the same size and incident counts, but has only one road across, even though they are equivalent in region based approaches. One the other hand, the Euclidean distance measure is not accurate and even misleading to decision makers when applying region based methods to traffic incident clustering. Figure 3.1 gives an example. In Figure 3.1a, three incidents are identified as an incident hotspot by K-means algorithm, because they are close. In Figure 3.1b, two of the incidents actually have no close enough road connection and so the cluster in Figure 3.1a was invalid.

To present our new algorithm Dijkstra-DBSCAN, which precisely clusters traffic incidents and further supports effectively lane management and decision making, this rest part of the paper is organized as follows: Section 3.2 briefly introduces DBSCAN algorithm. Section 3.3 modifies Dijkstra's shortest distance algorithm for our purpose, proves its correctness and finally shows Dijkstra-DBSCAN algorithm. Section 3.4 applies the algorithm to a case study using 5 years' FARS fatality data on the entire US road network. Section 3.5 analyzes the computational complexity and the parallelization of the algorithm. Section 3.6 discuss the results and concludes the paper with possible extensions.

(a) **(b)**

Figure 3.1: Incorrect clustering by K-means, because of the ignorance of underlying road network.

3.2 DBSCAN Algorithm

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) was brought up in 1996 Ester et al. (1996) and has been extensively applied in spatial analysis and hot spot detection. A core point o has at least $minpts$ neighbors in its $\varepsilon - neighbourhood$, i.e. the $\varepsilon - neighbourhood$ of o is the set $N_o = \{x_i | d(o, x_j) < eps\}$, where $|N| \geq minpts$. Here $d(\cdot)$ denotes the distance measure (metric) of two points, $|\cdot|$ denotes the size (cardinality) of a set. A core is a direct neighbor to itself. Two points p and q are density-connected if there exists a path $o_1 o_2 ... o_n$, where $p \in N_{o_1}$, $q \in N_{o_n}$, and $o_i \in N_{o_{i+1}}$, $i = 1, 2, ..., n-1$. A density cluster is defined as a core and all its density connected neighbors.

There are a few important properties from the definition of DBSCAN cluster:

(1) A cluster contains at least points. A cluster contains at least one core, which has at least $minpts$ directly connected neighbors.

(2) Any element in a cluster can reach at least one other point within a ε distance. Suppose an element p does not reach any other point within ε, then $\forall o_i, p \notin N_{o_i}$. Further, for an arbitrary core point o_j, there exists no path connecting p and o_j, by definition of a cluster, p is not an element of the cluster containing core o_j. Contradiction.

Property (1) constraints the minimum counts the points to constitute a valid cluster, and property (2) guarantee closeness of the points. The higher *minpts* and the lower ε are, the more dense the cluster is. These two properties makes DBSCAN an intuitive density based clustering algorithm. The pseudo code of DBSCAN is shown in Algorightm 1. It starts with obtaining the $\varepsilon - neighbourhood$ of each data point. If $\varepsilon - neighbourhood$ set has at least *minpts* elements, this point becomes a core point, indicating that it is qualified for starting up a cluster and expanding it later. The cluster expansion is a recursive depth first search (DFS) process: it starts from an arbitrary core point, traverse its directly connected neighbors and mark them as "visited", if any neighbor point is also a core, the above steps will be repeated on its $\varepsilon - neighbourhood$. The recursion stops after all the expandable cores are fully expanded. The recursion procedure is actually searching for all the path connected neighbors for the first core point. After each cluster expansion, we update the cluster id to start a new cluster till the all elements of the dataset are labeled as visited. Figure 3.2 illustrates the cluster expansion procedure with an example. In this example, P_1 and P_5 are core points because they both have 3 directly reachable neighbors within ε (the radius of the circles). P_1 is selected as a root core point, the algorithm expands the cluster recursively. $N(P_1) = \{P_1, P_2, P_3, P_5\}$, $C_0 = \{P_1, P_2, P_3, P_5\}$. In the directly accessible neighbor set of P_1, only P_5 is core. The unvisited neighbor of P_5 is $N(P_5) = \{P_6\}$. The cluster then is expanded to $C_0 = \{P_1, P_2, P_3, P_5\} \cup \{P_6\}$. The cluster is completely expandable and all points were visited. The algorithm stops and the cluster involves 5 points.

3.3 Modified Dijkstra's Shortest Path Algorithm

As a classic algorithm, Dijkstra's algorithms finds the shortest path from one source s to one or multiple targets t in a graph $G = (V, E)$, where V is the set of vertices and E of edges. Without repeating the basics of this well-known algorithm, we simply recap the essence of the algorithm, based on which we will modify it to fit our purpose: (1)

Algorithm 1: Dijkstra−DBSCAN Algorithm

```
Dijkstra−DBSCAN (D, G, ε, minpts)
    Input  : Data set D, network G, parameters ε and minpts
    Output: Clusters C_id
    id ← 0
    foreach point p ∈ D do
        N(p) ← Modified−Dijkstra (G, p, ε)
        if |N(p)| ≥ minpts then
            mark p as core
        end
    end
    foreach unvisited core point o do
        mark o as visited
        C_id = {o}
        id + +
        Expand−Cluster(o)
    end

Expand−Cluster (core point o)
    foreach unvisited point p′ in N(o) do
        mark p′ as visited
        C_id = C_id ∪ {p′}
        if p′ is core then
            Expand−Cluster(p′)
        end
    end
```

Two disjoint sets of vertices are used: an explored set S, where the shortest distances $d(s, v)$ from vertex v to source s were determined, and an unexplored set $V - S$, where the Dijkstra's distances $d(s, v)$ are undetermined yet. Initially, $S = \{s\}$. (2) In each step, a new element r_i is moved from unexplored set to explored set, where r_i and v_i are determined by: $\min_{\substack{v_i \in S \\ r_i \in V-S}} [d(s, v_i) + e(v_i, r_i)]$, and $d(s, r_i) = d(s, v_i) + e(v_i, r_i)$, where $e(v_i, r_i)$ represents the length of the edge connecting r_i and an element v_i from the unexplored set. In the k^{th} step, the size of the explored set is $|S| = k + 1$ (including s), while the size of the unexplored set is $|V - S| = |V| - k - 1$.

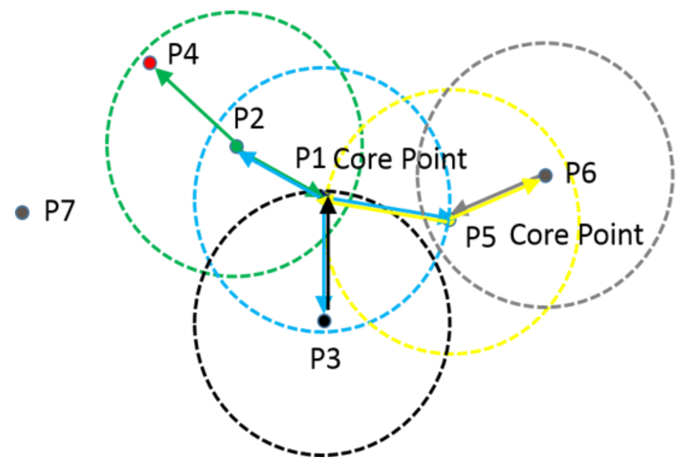


Figure 3.2: Cluster expansion in DBSCAN algorithm ($minpts = 3$).

Theorem 3.1. *For a non-negatively weighted graph, the vertex determined in step $k+1$ has a longer Dijkstra's distance than that of the vertex determined in step k, i.e. $d(s, r_{k+1}) \geq d(s, r_k)$.*

Proof for theorem. 3.1 We use induction method. (1) When $k = 0$, $d(s, r_1) \geq 0$, $d(s, r_0) = d(s, s) = 0$. The inequality holds. (2) When $k = n-1$, suppose the inequality is satisfied: $d(s, r_n) \geq d(s, r_{n-1})$. (3) We need to prove that when $k = n$, $d(s, r_{n+1}) \geq d(s, r_n)$. In (2), r_n is determined by satisfying $\min\limits_{\substack{v_i \in S_{n-1} \\ r_i \in V - S_{n-1}}} [d(s, v_i) + e(v_i, r_i)]$. Since $d(s, r_n)$ is the minimum for all v_i in S_{n-1} and all r_i in $V - S_{n-1}$, we have: $d(s, r_n) \leq d(s, v_i) + e(v_i, r_{n+1})$, $v_i \in S_{n-1}$. To determine next vertex r_{n+1}, we have $d(s, r_{n+1}) = \min\limits_{\substack{v_i \in S_n \\ r_i \in V - S_n}} [d(s, v_i) + e(v_i, r_i)]$, notice that $S_n = S_{n-1} \cup \{r_n\}$. There are two possibilities for v_i: if $v_i = r_n$, $d(s, r_{n+1}) = d(s, r_n) + e(r_n, r_{n+1}) \geq d(s, r_n)$; else, $v_i \in S_{n-1}$, $d(s, r_{n+1}) = d(s, v_i) + e(v_i, r_{n+1}) \geq d(s, r_n)$. So $d(s, r_{k+1}) \geq d(s, r_k)$ holds for $k = n$ as well. By the principle of induction, the inequality is true for all steps. QED. □

Our modification of Dijkstra's algorithm is presented in Algorithm 2. It is based on Theorem 3.1 that the shortest distances ordered by steps is a monotonically increasing sequence. Instead of terminating the routing when the target is found as the original

Algorithm 2: Modified−Dijkstra's Shortest Path Algorithm

```
Modified−Dijkstra (G, s, ε)
    Input  : Network G, source node s, parameters ε
    Output: All neighbor nodes within ε of s, neighbor_list
    d[s] ← 0
    Q ← {s}
    neighbor_list = {}
    while Q not empty do
        u ← Q[0]
        Q ← Q − {u}
        foreach neighbor v of u do
            if v is unvisited and d[u] + e(u, v) < d[v] then
                d[v] ← d[u] + e(u, v)
                Q ← Q ∪ {v}
            end
        end
        sort Q in ascending order based on distance
        if d[Q[0]] > ε then
            return neighbor_list
        end
        else
            neighbor_list ← neighbor_list ∪ Q[0]
        end
        mark Q[0] as visited
    end
```

Dijkstra's does, we control the algorithm by comparing the most lately determined shortest distance with the maximum distance in DBSCAN. When a vertex gets a larger distance than ε, there is no need to evaluate any other nodes because their distance to the source will be even bigger. The modified Dijkstra's algorithm returns the vertices in the explored set, which are exactly the directly reachable neighbors that we need in Dijkstra-DBSCAN.

The complexity analysis is consistent with that for original Dijkstra's algorithm. The difference is that instead of knowing the numbers of edges and vertices, the routing here is on a local neighborhood, the scale of which is determined by ε. The larger ε is, the larger the local network will be and so the more edges and vertices will be visited. We denote the number of edges and vertices as $|E(\varepsilon)|$ and $|V(\varepsilon)|$

respectively. The time complexity for getting the neighborhood of one event point will be $O\left(|E(\varepsilon)| + |V(\varepsilon)|\log(|V(\varepsilon)|)\right)$. Density based clustering usually use a small ε to capture the most significant clusters, so the network is small and getting the neighborhood for one event point takes little time. We should also emphasize one fact that the time complexity of the neighborhood retrieval step is determined by the number of traffic event point and ε, but not the scale of the road network. This fact is the basis for the scalability of the Dijkstra-DBSCAN algorithm.

3.4 Dijkstra-DBSCAN for Incident Hotspots Identification: A Case Study

We test Dijkstra-DBSCAN on the road network of the entire United States. The road network topology includes 27,145,945 edges and 17,464,790 vertices. The traffic event data we used are the 152,089 traffic fatalities of 5 years (2009 to 2013) provided by Fatality Analysis Reporting System (FARS). We partition the US road network and event data into 51 groups by their state label. Figure 3.3 shows the basic statistics of the road network and fatality counts of 51 states. Fatality count of each state is positively related to the scale of the road network. Figure 4.2 presents some sample clusters generated with $\varepsilon = 1000m$ and $minpts = 3$.

Figure 3.4a shows clustering on freeway network. We can see that Dijkstra-DBSCAN gets more accurate clusters than Euclidean based DBSCAN. Even though some points are close to others by Euclidean distance, they are not included in the cluster because of the lack of actual short road connection. It also shows us the routing ability of the algorithm. A cluster generated by Dijkstra-DBSCAN is not interrupted by intersections, roundabouts, ramps, etc. It is so flexible that can cross the corners and connects to other incident locations. Although within short future, most managed lane networks will be limited to freeways, the success of these pilot projects would potentially introduce the system to networks of more complicated road

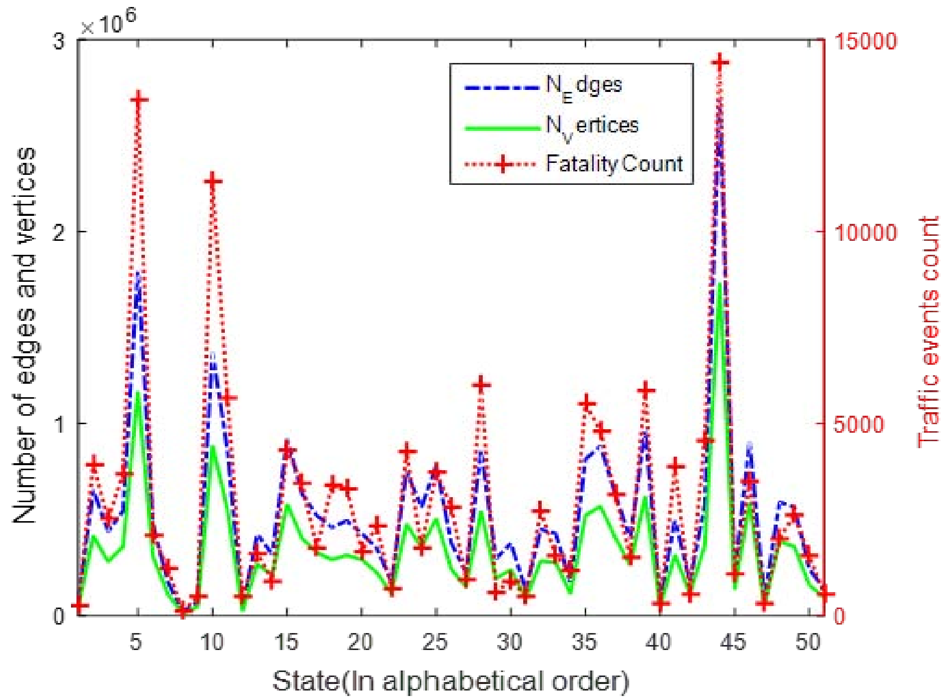
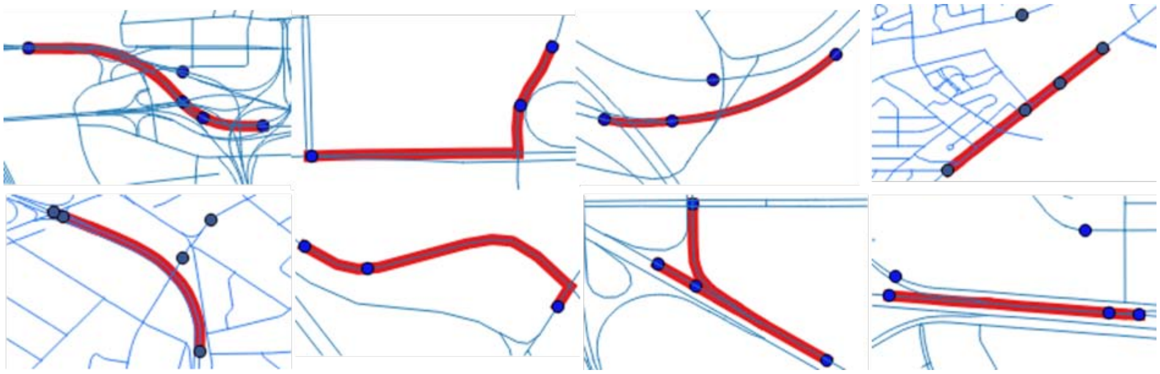


Figure 3.3: Numbers of links, nodes and 5-year fatality counts of 51 states in U.S.

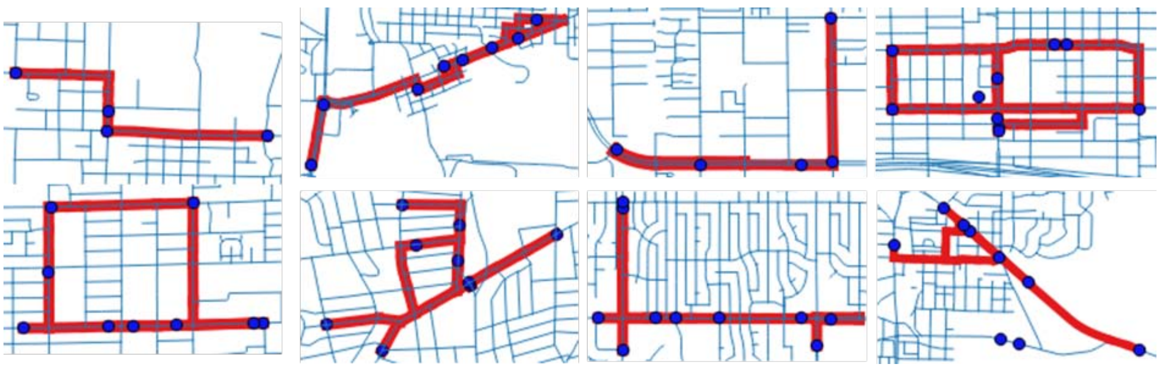
surface configurations. Figure 3.4b shows the ability of Dijkstra-DBSCAN to extract incident clusters of any arbitrary shapes in complicated road network topology. The hidden spatial connections of the incidents over these networks are fully revealed by the algorithm, verifying the effectiveness of using Dijkstra-DBSCAN to discover incident hotspots for simple and complicated managed lane networks.

3.5 Computational Cost Analysis and Parallelization

We have analyzed the time complexity of modified Dijkstra's is $O\left(|E(\varepsilon)| + |V(\varepsilon)|\log(|V(\varepsilon)|)\right)$, where E and V represent the number of edges and vertices of the local graph around an incident point, and ε determines the scale of the local graph. Suppose the dataset contains traffic incidents, then the neighbor retrieval step has a time complexity of $O\left(N \cdot (|E(\varepsilon)| + |V(\varepsilon)|\log(|V(\varepsilon)|))\right)$. The other step in



(a) Freeway networks



(b) More complicated networks

Figure 3.4: Examples of incident clusters extracted by Dijkstra-DBSCAN on road networks.

Dijkstras-DBSCAN is cluster labeling and expanding, which is essentially a Depth First Search (DFS). Every vertex is visited once, so the complexity is $O(N)$, which can be ignored compared with the complexity of neighbor retrieval. So the overall complexity is $O(N \cdot (|E(\varepsilon)| + |V(\varepsilon)| \log(|V(\varepsilon)|)))$.

We consider a relatively large traffic event dataset. Since the local network to retrieve the neighbors of an event point from is a small one, so $|E(\varepsilon)|+|V(\varepsilon)| \log(|V(\varepsilon)|$ will also be a relatively small number compared with N, so the overall complexity of Dijkstra-DBSCAN is near-linear. Compared with the Euclidean distance based DBSCAN, which has a $O(N^2)$ complexity, the network-aware Dijkstra-DBSCAN will achieve a much faster performance.

When we use a larger ε, the local graphs will be bigger and so it will definitely takes longer time to complete the clustering process, but this does not change the order of magnitude of the overall time complexity.

The parallelization of DBSCAN has been a challenging problem, mainly because data access in both steps are sequential in nature Patwary et al. (2012); Brecheisen et al. (2006), and partitioning data will cause heavy communication overhead. Road network and traffic events data, however, have something special in common that makes data partitioning easy: administrative entity tag is available. Any road network database has an attribute showing which state or county a road segment belongs to; for traffic events data, even if the administrative area tag is not directly available, we can still quickly get the administrative entity it lies in from GPS coordinates. Hereby, we can divide the data by their territorial areas.

Now suppose both the road network and event data are split into disjoint territorial groups. Different groups own different volumes of data. As we stated above, the time cost of the algorithm is near linearly related to the number of event points, which then becomes an appropriate measure of the computational loads of each data group. Suppose we have S groups, and their event data volumes are in $V[0, 1, ..., S-1]$, we have P distributed processors. $P \leq S$. Allocating the data to the processors can find its solution from a well-studied list scheduling problem Mokotoff et al. (2001); Brinkmann (2017), the heuristic solution to which is to iteratively assigning a unassigned job to the processor that currently owns the least load until all jobs have been allocated.

We now show that the differences between the loads of different processors are bounded. The loads of two processors p_1, p_2 are $L[p_1]$, $L[p_2]$ respectively, then $|L[p_1] - L[p_2]| \leq \max(V)$. According to Algorithm 3, the two loads must be determined in two different steps. Without loss of generality, we assume $L[p_2]$ is determined a step k, which is later than $L[p_1]$. Then at the beginning of the step k, $L_k[p_2] \leq L[p_1]$ and $L[p_2] = L_k[p_2] + V[k] \leq L[p_1] + V[k]$, so $L[p_2] - L[p_1] \leq V[k] \leq$

Algorithm 3: Job Scheduling for Parallel Dijkstra-DBSCAN

JobScheduling($S, V[S], P$) **Input** : Number of groups S, work load $V[S]$, number of processors P

Output: Job scheduling $J[P]$

for $i \in \{1, 2, ..., P\}$ **do**

  $L[i] \leftarrow 0$

  $J[i] \leftarrow \{\}$

end

for $i \in 1, 2, ..., S$ **do**

  $k \leftarrow argmin\,(L)$

  $J[k] \leftarrow J[k] \cup \{i\}$

  $L[k] \leftarrow L[k] + V[i]$

end

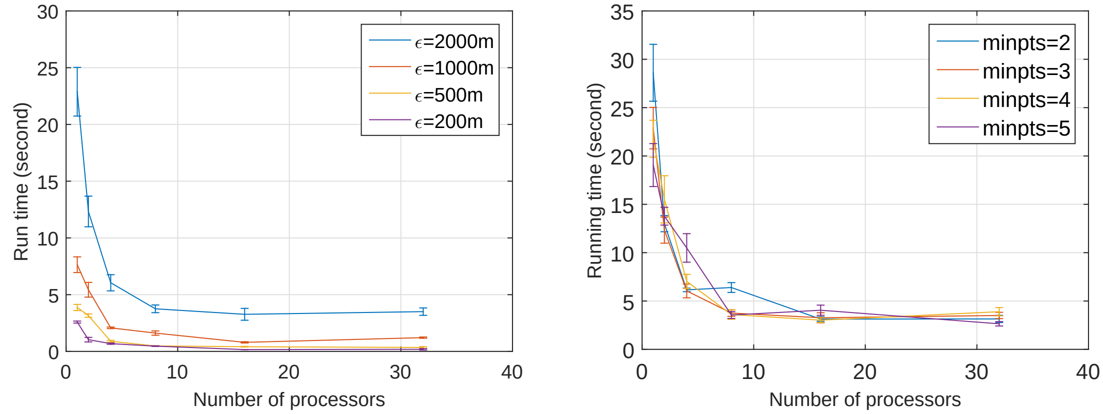
return J

max(V). This tells us that the processors have pretty much balanced loads and can finish the computations within similar length of time.

We benchmark the algorithm on distributed computer clusters. We also changed the parameters to see their influences on the performance. Figure 3.5 show the results. Here is what we observed:

(1) Dijkstra-DBSCAN is fast. Even on a single one-core computer, it can complete clustering task on the entire U.S. network within less than 25 seconds. We also tried Euclidean based DBSCAN, and even for Tennessee state that has 573,444 edges and 366,940 vertices, and the fatality count is 4,580, it takes more than 20 minutes. The reason behind the differences is: Dijkstra-DBSCAN has a near-linear complexity while the original DBSCAN has a $O(n^2)$ complexity. In the neighborhood retrieval step, one incident point has to compare with all other points, so Euclidean DBSCAN is a global search approach; In contrast, Dijkstra-DBSCAN does local search, the underlying road network enables a point to search just within a local neighborhood.

(2) Dijkstra-DBSCAN is scalable. The more processors used, the faster the algorithm runs. When using 16 processors, it taken only 3 to 4 seconds to get the clusters that has a minimum distance of two kilometers. While Dijkstra-DBSCAN is



(a) Different ε values **(b)** Different $minpts$ values

Figure 3.5: Runtime benchmark of parallel Dijkstra-DBSCAN.

parallel friendly, Euclidean based DBSCAN is hard to be parallelized as we already mentioned earlier.

(3) The two parameters have different impacts on the parallel performance of the algorithm. ε is more significant than $minpts$. This is because the time cost of Dijkstra-DBSCAN is mainly determined by the neighborhood retrieval step. When ε is big, a point has to search in a broader range, which then will take more time. $minpts$ determines how many points will be labeled as cores. The cluster expanding step is essentially a DFS. We have analyzed that the time complexity of cluster expanding step equals the number of cores, which can be ignored compared with neighborhood retrieval step. A smaller $minpts$ will identify more core points and indeed increase the time cost, as we can see from Figure 3.5b. But the effect of $minpts$ becomes less and less significant as more processors are used.

3.6 Conclusion and Future Work

We have come up with and implemented Dijkstra-DBSCAN, a density based clustering algorithm specifically for traffic incident hotspot identification, lane safety management and investment location decision support. Compared with region based methods such as KDE, DBSCAN, K-means, hierarchical clustering, the new algorithm has the following advantages: (1) It is based on network space and adopts routing distance measure, and so the clusters are more accurate than region based approaches for traffic management purpose; (2) It has routing flexibility and can extract clusters of irregular shapes (3) It has a time complexity of $O(n)$, which is a large speed up from the $O(n^2)$ complexity of all other algorithms. (4) It is parallelable and can utilize distributed computing resources to handle large scale network.

There are still a lot to investigate on Dijkstra-DBSCAN to fully support incident hotspot analysis and project location selection related decision making. One important issue is that all algorithms including our new algorithm will get a number of clusters, but there has been no literature looking into clustering ranking to find out the most significant few clusters. This problem becomes essential when investment budget is so limited that only a few locations will be selected to be improved. The good thing is that Dijkstra-DBSCAN gives clusters together with additional information of the incidents' distance relations. We will be developing a measure of the overall "density" of a cluster to support practical and reasonable incident cluster ranking, which hopefully will provide valuable reference to investment decision making.

Chapter 4

Online Step-wise Regression: Compressed GPS Trajectory Data Acquisition

4.1 Introduction

Big data is offering new insights to transportation studies and applications. GPS trajectory data is a spatio-temporal data type that records both the latitude-longitude location and the associated timestamp of a moving object. Vehicle travel trajectories are acquired through the real-time communication between the data center and the in-vehicle gadgets, such as smart phones, GPS navigators, etc. Trajectory data are reported to have served for many different mobility purposes. Moreira-Matias, Lus, et al designed a novel scheme to infer time-varying Original-Destination(OD) matrix using high speed GPS trajectory stream (Moreira-Matias et al., 2016). Giannotti, Fosca et al analyzed a dataset containing more than 1,500,000 trajectories. These trajectories were collected during a 5 weeks period and covered a 100km by 100km spatial region centered around Pisa city, Italy Giannotti et al. (2011). The high volume of the data enabled them to reveal the most frequently used routes of the residents in different neighborhoods and the impacts of big population-attracting events on traffic flow, They also used real-time trajectories to predict future traffic density and potential congestion area Giannotti et al. (2011). Vehicle trajectories are also used to approximate average route choice proportion Sohn and Kim (2008). Massive historical trajectory data contain rich information about the traffic conditions. Besides mining the common patterns, it also offers the opportunity to identify the anomalies, such as traffic incidents, special events, emergencies, etc. Sanjay and Yu et al utilized probing taxis' traces to detect segments that are experiencing abnormal flows, and infer the most likely origins and/or destinations that cause it Chawla et al. (2012). Bei and Yu et al captured traffic congestion by comparing drivers routing choices with normal historical patterns Pan et al. (2013). The captured anomalies, after prompt confirmation, will be broadcasted to the drivers in order for them to take alternative routes to avoid those areas. Another amazing application of vehicle trajectories is the emerging route recommendation and travel time prediction services. Based on centralized cloud storage and computing, the system use archives of historical vehicle trajectories to train a deep neural network learning model, and factors such as real-time traffic conditions, weather, day of the week, time of the day, locations, a user's preference, etc. are used as input to compute the fastest route for a user. Yuan et al first brought up this idea and their implementation using the trajectories of more than 3,300 taxis over a 3 months period can accurately predict any route's travel time and make the smartest choices for users in a dynamic urban environment Yuan et al. (2011, 2013). The tremendous benefits of trajectory data is drawing more and more attentions, from not only academic world, but also the open data community and the government agencies that are progressively constructing smarter cities. Thousands of people are uploading their daily travel traces to the OpenStreetMap Public Traces repository Map (2016). Some cities like Bristol are allowing vehicles trace collection to support their intelligent transportation construction Moreira-Matias et al. (2016); 2016 (2016).

GPS trajectory can be recorded at different sampling rates. A lower sampling rate, one point every a few minutes, makes it quite challenging to correctly reconstruct a vehicle's route Newson and Krumm (2009), especially in dense network such as urban area. Yin et al designed map matching algorithms for low sampling rate trajectory data Lou et al. (2009), but the general accuracy is still below 60% even at a prohibitively high computational cost. This study will focus on higher sampling rate data, where the data is sampled every 1 to 30 seconds.

Although collecting trajectory data is distributed, centralized storage of them could be a huge challenge. According to Didi Chuxing, a ride hailing company in China, over 50 Terabytes of trip data are being generated from their services every single day DiTech (2016). Even though massive storage systems are becoming cheaper and cheaper and might not be a big concern anymore, processing such huge amount of data is a substantial barrier for classical learning and predicting algorithms and current computing resources. On the other hand, microscopic and macroscopic level applications of trajectory data have different requirements on data accuracy. Microscopic level applications usually require the data to be able to discriminate lanes and vehicles' relative positions, and so tend to rely on high-accuracy trajectories. Examples include calibrating car-following models for microscopic traffic simulation Ossen and Hoogendoorn (2008); Punzo et al. (2012), identifying drivers' aggressive behaviors, designing collision avoidance strategies in autonomous and connected vehicle environments Shladover and Tan (2006); Ahmed-Zaid et al. (2011), and so forth. For many macroscopic level purposes, such as dynamic OD inference, traffic demand prediction, anomaly identification, route recommendation etc., the requirement is not as demanding. This leaves room for us to sub-sample the data as we collect them.

This chapter is aimed at collecting trajectory data in a real-time manner for macroscopic purposes. Our goal is to reduce data redundancy as much as possible but keep the "points of interest" and maintain a similar accuracy as the original trajectories. Our approach is based on a simple observation that we are driving at straight direction most of the time, unless we make a turn or the road is not straight. We use the piecewise linear regression method to partition a vehicle's trajectory into straight line pieces as it moves along. The paper is organized as follows: In Section 4.2, we overview the incremental liner regression algorithms, where we bring up the incremental QR decomposition using row-wise Givens Rotation and we will prove is theoretically valid. Section 4.3 introduces the online segmentation process, which is based on classical sliding window method. We include a case study using a real world dataset. Section 4.4 is on precision control and performance evaluation. We explore the association between the accuracy, compression power and the parameter value, then introduce the procedure to find the best parameter value for a new type of GPS device. Section 4.5 concludes the chapter by addressing the limitation and future work.

4.2 Incremental Linear Regression

Extensive data sources and real-time data collection and transferring not only provide us with new opportunities to discover in-depth patterns in new dimensions, but also offer new opportunities for instantaneous decision makings. Most of the classical statistical learning models, however, were initially designed for historical data analysis, and might not always be the best or even proper choice for real-time applications. The difference between an incremental and non-incremental algorithm is that when a block of the data changes (including appending new data, editing or deleting some cells), a non-incremental computation recalculates with entire updated data, while an incremental algorithm updates the preserved sub-result with the changed data cells. A non-incremental algorithm stores all the historical data for future updates, while the incremental does not need to save the data but preserves only the intermediate results. Besides less space consumption, incremental algorithm usually works faster than non-incremental when the data quantity is large. Our simultaneous trajectory data collection is based on linear regression, the simplest learning algorithm but the prototype for many other regression models, such as binary logistic regression, multi-nomial logistic regression, nonlinear regression, multiple regression, etc. Below we evaluate the incremental update ability of the four well known linear regression methods one by one. Some of them are able to be modified for online purpose while the others are not. The incremental methods will also apply to other regressions as well.

4.2.1 Setting of Linear Regression

The original simple linear regression model, $Y = \beta X + \varepsilon$, consists of independent variable vector $X_{n\times 1} = [x_1\ x_2\ \cdots\ x_n]^T$, dependent variable $y_{1\times 1}$, coefficient vector $\beta_{1\times n} = [\beta_1\ \beta_2\ \cdots\ \beta_n]$, and constant term $\varepsilon_{1\times 1}$. The learning process is to determine β and ε using inputs X and outputs Y.

We usually reformat the model a little bit, as Equation 4.1:

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}_{m\times 1} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}_{m\times(n+1)} \begin{bmatrix} \varepsilon \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}_{(n+1)\times 1} \tag{4.1}$$

where $y^{(j)}$ is the j^{th} output, and $x_i^{(j)}$ is the j^{th} input of variable x_i. We then denote Equation 4.1 as:

$$B = Aw \tag{4.2}$$

with $w = [\varepsilon\ \beta_1\ \beta_2 \cdots\ \beta_n]^T$, and A the input matrix and B the output matrix.

The four well known methods for solving linear regression are: Normal Equation, Gradient Descent and Stochastic Gradient Descent (SGD), QR Decomposition and SVD. It is not straightforward whether previous data and new coming data can be separated in these methods because of the numerical algorithms involved, plus we also need to consider the numerical stability of the methods, so we will investigate

Table 4.1: Numerical errors for near-singular matrix

Method	ε					
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
Normal Equation	2.58×10^{-12}	1.26×10^{-6}	7.59×10^{-5}	1.52×10^{-2}	2.3	124.63
QR Decomposition	0	0	0	2.84×10^{-14}	1.99×10^{-13}	3.44×10^{-12}

them one by one. "Near-singularity" is a very common case in latitude and longitude data, because two locations can be very close to each other, in which case their latitude/longitude values are almost the same until a few decimal places after the decimal point. Therefore, the numerical stability is crucial.

4.2.2 Normal Equation

Normal equation is the analytical solution to the least square cost function optimization problem. It has a simple and nice form $w = (A^T A)^{-1} A^T B$. However, matrix inversion brings in unacceptable errors when the matrix is near singular. To show this, we create matrices $A = \begin{bmatrix} 1 & 47 \\ 1 & 47+\varepsilon \end{bmatrix}$, $B = \begin{bmatrix} 120 & 120.001 \end{bmatrix}^T$. When $\varepsilon \approx 0$, A is close to singular. We try $\varepsilon = 10^{-1},\ 10^{-2},\ \cdots,\ 10^{-6}$ respectively, calculating w using the normal equation, and the prediction error $\|wA - B\|$. The errors resulting from the normal equation are listed in Table 4.1, with comparison values from QR decomposition, a numerically stable approach. Normal equation has poor numerical stability, and so is not a good option for us.

4.2.3 Gradient Descent (GD) and Stochastic Gradient Descent (SGD)

Gradient Descent (or Batch Gradient Descent) is an iterative method that approaches the optimal w step by step: $w_j = w_j + \alpha \sum_{i=0}^{m} (y^{(i)} - \sum_{k=0}^{n} w_k x_k^{(i)}) x_j^{(i)},\ j = 0, 1, ...n$. GD is not an online algorithm. Its online version is Stochastic Gradient Descent: $w_j = w_j + \alpha(y^{(m)} - \sum_{k=0}^{n} w_k x_k^{(m)}) x_j^{(m)}$. The difference is that SGD uses only the new input to determine the step size, while GD uses the entire dataset. SGD is not as

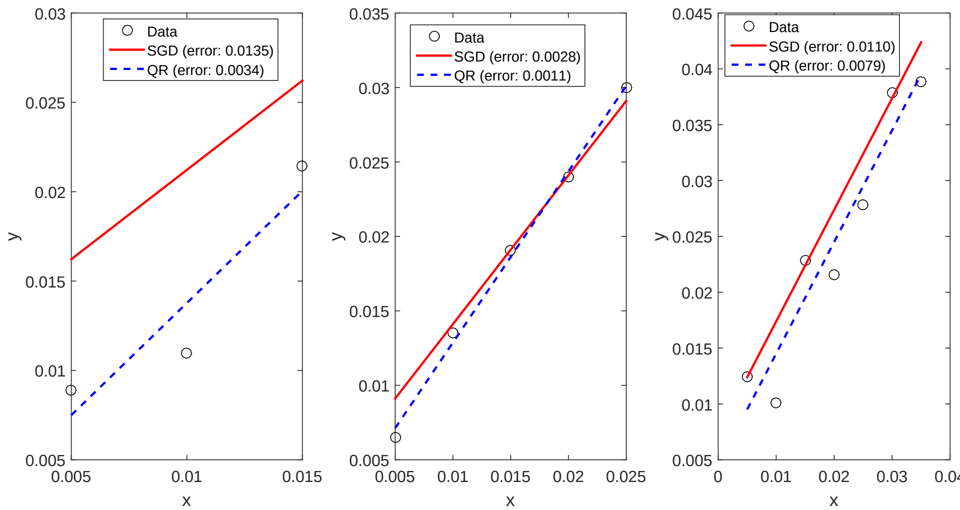
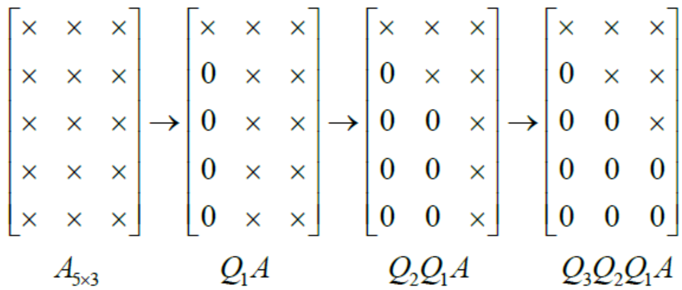


Figure 4.1: Regression on small quantity of data using SGD and QR.

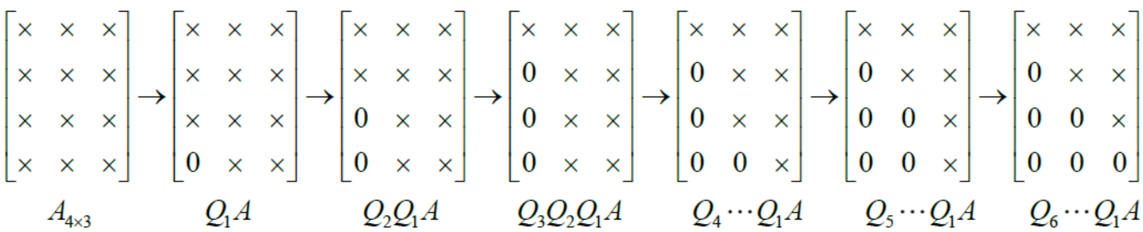
accurate as GD, but is a good approximation when there are sufficient data for it to converge, and so is preferred in large-scale machine learning tasks because of the substantially reduced computational cost. Our application, stepwise linear regression, although handles endless streaming data flow, should not be considered a large scale learning problem. When the trajectory is “flat”, a lot of points will fall into the same segment, so a relatively big learning rate can guarantee SGD converges. However, if the trajectory “fluctuates”, either because the road is not straight, or the driver is making turns, a segment might include only a few points, and the regression can be quite different from an optimal solution, as shown in Figure 4.1. Since we will be controlling the accuracy of sampled data, we want the regression to be optimal, hereby we will not consider SGD.

4.2.4 QR Decomposition

With QR decomposition $A_{m\times(n+1)} = Q_{m\times(n+1)}R_{(n+1)\times(n+1)}$, where Q is an orthogonal matrix and R is an upper triangular matrix, the linear regression function $B = Aw$ becomes $Rw = Q^T B$, and so w can be solved using backward substitution to avoid matrix inversion. Gram-Schmidt orthogonalization, Householder transformation and



(a) Householder transformation



(b) Givens rotation

Figure 4.2: Examples of Householder transformation and Givens rotation.

Givens rotation are three well known techniques to do QR decomposition. Classical Gram-Schmidt(CGS) suffers from round-off errors and server loss of orthogonality Golub and Van Loan (2012), while Modified Gram-Schmidt (MGS) is numerically robust, but there is no row-wise MGS Leon et al. (2013), which is what we need for incremental regression. Householder is more favored in historical data analysis, because of its lower computational complexity than Givens (For a $m \times n$ matrix with $m > n$, householder transformation needs $2mn^2 - \frac{2}{3}n^3$ operations, while Givens needs $3mn^2 - n^3$ Egecioglu and Srinivasan (1995)). The difference between the two is that Householder introduces zeros column by column, while Givens element by element, as shown in Figure 4.2a and Figure 4.2b.

In linear regression application, new-coming data will be placed in a new row. Householder transformation is an offline algorithm due to its column-by-column nature. Givens rotation, however has the potential to be an online algorithm. Before we show how this is feasible, let's look at the validity of the order of Givens rotations in QR factorization algorithm.

4.2.5 Givens Rotation

To zero out a_{ij} $(i > j)$ of matrix $A_{m\times n}$, we left-multiply A by $G^{(i,j)}$, which comes from the identity matrix with four elements modified, see Equation 4.3

$$G^{(i,j)} = \left[\begin{array}{ccccccccc} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cos(\theta) & \cdots & 0 & \cdots & -\sin(\theta) & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & 0 & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \sin(\theta) & \cdots & 0 & \cdots & \cos(\theta) & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \end{array}\right] \begin{array}{|l} \\ \\ j^{th}\ row \\ \\ \\ \\ i^{th}\ row \\ \\ \\ \end{array} \tag{4.3}$$

The four modified elements are $g_{jj} = \cos(\theta),\ g_{ji} = -\sin(\theta),\ g_{ij} = \sin(\theta),\ g_{ii} = \cos(\theta)$, where $\cos(\theta) = \frac{a_{jj}}{\sqrt{a_{jj}^2+a_{ij}^2}}$, $sin(\theta) = \frac{-a_{ij}}{\sqrt{a_{jj}^2+a_{ij}^2}}$. We normally apply the following column-by-column sequence of Givens rotations to A, to reduce it to an upper triangular matrix:

$$\underbrace{G^{(2,1)}, G^{(3,1)}, \cdots\ G^{(m,1)}}_{zero\ out\ 1^{st}\ column}, \underbrace{G^{(3,2)}, G^{(4,2)}, \cdots\ G^{(m,2)}}_{zero\ out\ 2^{nd}\ column}, \cdots \underbrace{G^{(n+1,n)}, G^{(n+2,n)}, \cdots\ G^{(m,n)}}_{zero\ out\ n^{th}\ column}$$

There are other valid sequences too. However, an arbitrary sequence is not always valid. For example, for $A_{3\times 2} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$, applying a valid sequence $G^{(2,1)}, G^{(3,1)}, G^{(3,2)}$, we get $R = \begin{bmatrix} 5.9161 & 7.4374 \\ 0 & -0.8281 \\ 0 & 0 \end{bmatrix}$ and $Q = \begin{bmatrix} 0.1690 & -0.8971 & 0.4082 \\ 0.5071 & -0.2760 & -0.8165 \\ 0.8452 & 0.3450 & 0.4082 \end{bmatrix}$. One can verify that Q is orthogonal and

R is upper triangular. However, if we apply $G^{(3,2)}, G^{(2,1)}, G^{(3,1)}$, we get $R = \begin{bmatrix} 5.9161 & 7.4374 \\ 0 & -0.7509 \\ 0 & -0.3491 \end{bmatrix}$, $Q = \begin{bmatrix} 0.1690 & -0.9856 & -0.0079 \\ 0.5071 & 0.0939 & -0.8568 \\ 0.8452 & 0.1408 & 0.5156 \end{bmatrix}$, where R is not upper triangular and so this Givens rotation sequence does not fulfill QR factorization.

Dianne and Stephen used a "rotation graph" to determine if a given rotation ordering is proper for QR. Their method offers a practical tool that enables effective QR factorization in quantum circuit design and parallel QR decomposition computation Bullock (2005). For our online regression application, it is sufficient to just show a row-by-row rotation sequence will also fulfill QR decomposition. We state this in Lemma 4.0.1 but put the proof in the appendix part in order not to distract the readers to delve into the lengthy and pure algebra proof.

Lemma 4.0.1. *The row-by-row Givens rotation sequence*

$$\underbrace{G^{(2,1)}}_{2^{nd}\ row}, \underbrace{G^{(3,1)}, G^{(3,2)}}_{3^{rd}\ row}, \cdots \underbrace{G^{(n,1)}, G^{(n,2)}, \cdots G^{(n,n-1)}}_{n^{th}\ row}, \underbrace{G^{(n+1,1)}, G^{(n+1,2)}, \cdots\ G^{(n+1,n)}}_{(n+1)^{th}\ row}, \cdots \underbrace{G^{(m,1)}, G^{(m,2)} \cdots\ G^{(m,n)}}_{m^{th}\ row}$$

is valid for QR decomposition.

Using the row-by-row Givens rotation, the incremental solution to the linear regression problem is then presented in Algorithm 4.

Figure 4.3 shows that using incremental Givens rotations can significantly reduce the time complexity of the non-incremental Givens, and the advantage becomes more and more obvious as more data come in. The increasing data quantity does increase the run-time of Givens update itself, mainly because of the more steps in multiplying bigger and bigger Q and G matrices.

4.2.6 Singular Value Decomposition (SVD)

Singular value decomposition of a matrix A has the form $A_{m\times n} = U_{m\times m} S_{m\times n} V^T_{n\times n}$, where U and V are unitary matrices and S is a rectangular diagonal matrix with

Algorithm 4: Incremental row-wise Givens rotation for linear regression

Incremental−Givens

Input : new data $X_{(m+1)}$, y_{m+1}, stored $Q_{m\times m}$ and $R_{m\times(n+1)}$ from last step

Output: w

Step 1: $m = m+1, (m \geq n)$

Append new data to a new row of R and B:

$$R_{(m+1)\times(n+1)} = \left[\begin{array}{c} R_{m\times(n+1)} \\ \hline \begin{bmatrix} 1 & x_1^{(m+1)} & x_2^{(m+1)} & \cdots & x_n^{(m+1)} \end{bmatrix} \end{array}\right]$$

$$B_{(m+1)\times 1} = \begin{bmatrix} B_{m\times 1} \\ y_{m+1} \end{bmatrix}$$

Step 2: Apply $G^{(m+1,1)}, G^{(m+1,2)}, \cdots G^{(m+1,n+1)}$ to R:

$$Q = \left(G^{(m+1,n+1)} \cdots G^{(m+1,2)} G^{(m+1,1)}\right)^T Q$$

$$R = G^{(m+1,n+1)} \cdots G^{(m+1,2)} G^{(m+1,1)} R$$

Step 3: Use backward substitution to solve w:

$$Rw = Q^T B$$

Return w

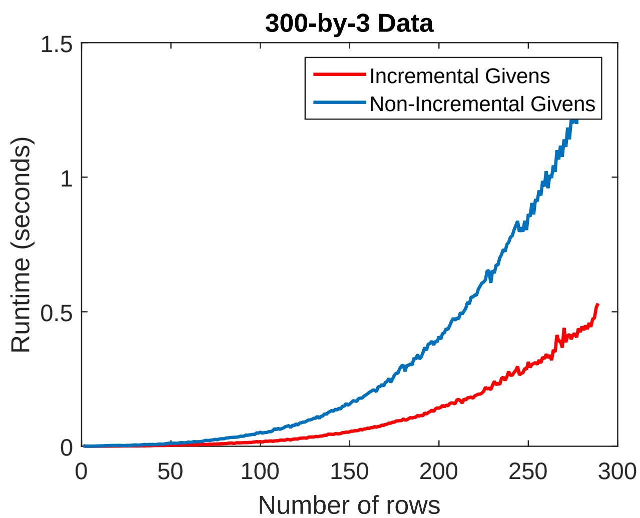


Figure 4.3: Runtime comparison of non-incremental and incremental Givens rotations.

singular values as its diagonal entries. Regression function $Aw = B$ becomes $Sw = VU^TY$. Since S is rectangular diagonal, solving w becomes very easy. Standard SVD numerical algorithm is two-step procedure: First use Householder or Givens rotations to reduce A to a bi-diagonal form, then calculate the singular values and the unitary matrices. The details can be found in Brent et al. (1982). Matthew presented an incremental SVD algorithm Brand (2006). Although they only discussed adding, deleting and modifying a column, we can quickly modify it for row update scenario, and so to fit our online regression application. Readers are referred to the original paper for the details of incremental SVD, we here apply it to online regression.

We write the new data matrix after appending a new row of data as $A_{(m+1)\times n} = A_{m\times n} + a_{(m+1)\times 1} b^T_{1\times n}$, where "+" is the concatenation operator: embedding $A_{m\times n}$ into the first $m \times n$ cells of $a_{(m+1)\times 1} b^T_{1\times n}$. Here $a = \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix}^T_{1\times(m+1)}$ and $b = c^T_{1\times n}$, where c is the new data row. Following Matthew's incremental SVD, we can implement instantaneous linear regression as Algorithm 5.

Since matrix K is built from singularity matrix of the last step, there is only one new row that needs transformations, which can be done in $O(n^2)$ steps. This is significantly faster than the $O(m^2n + n^3)$ complexity of non-incremental SVD Holmes et al. (2007). As Figure 4.4 shows, Incremental SVD takes almost a constant time to process a new row, no matter how many rows are coming in. This property makes it "outperform" non-incremental SVD.

To summarize this section, we have found that row-wise Givens rotation based QR decomposition and incremental SVD are two valid numerical algorithms to perform linear regression for real-time streaming data.

4.3 Sliding Window Method for Online Stepwise Regression

4.3.1 Partitioning Trajectories

Sliding window method is a commonly used method for online time sequence segmentation Keogh et al. (2004). The window defines the range of the data being

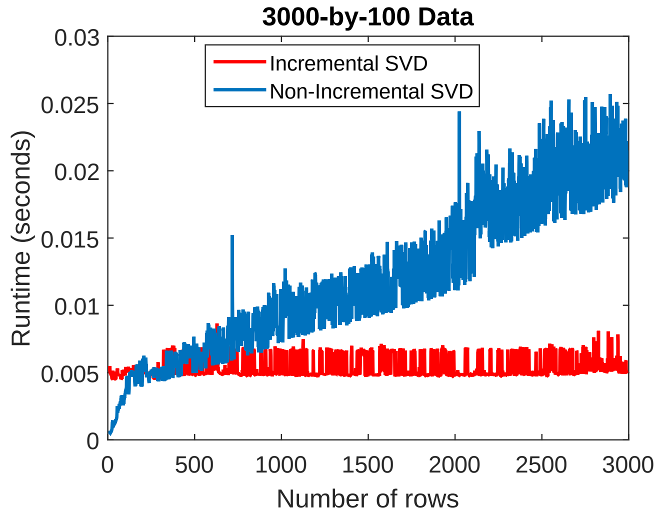


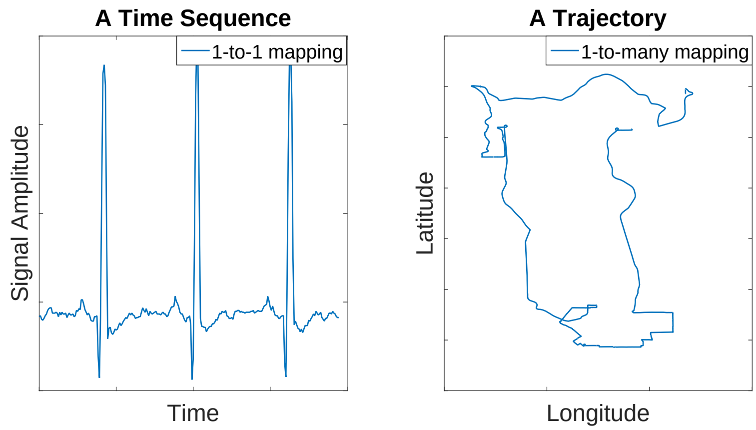
Figure 4.4: Runtime comparison of non-incremental and incremental SVD.

evaluated at the current step. The window should guarantee we have enough data points to solve the regression function. For the linear regression based partition, we can specify "start" as 1 and "end" as the rank of the data set. While the prediction error by the optimal regression model is smaller than the maximum error specified by the user, the window is allowed to expand by adding 1 to "end". When the error exceeds the tolerance, the data covered by the window is identified as a segment. The algorithm then proceeds by setting "start" as "end", and "end" as "end+rank". For historical data analysis, it stops when all the data points have been evaluated, but for active data stream, the algorithm keeps evaluating new inputs. Sliding window method has been used for time sequence partitioning, with the dependent variable the magnitude of the signal and independent variable the time stamp. We here use it without time stamp: since the trajectory shape is about latitude and longitude positions, we make the latitude the dependent variable and longitude the independent variable.

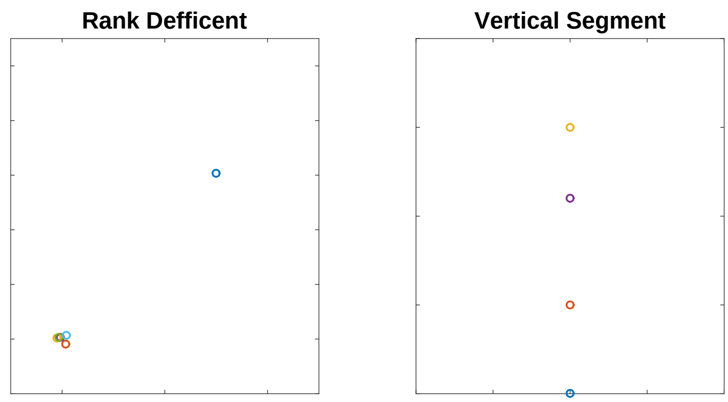
A time sequence is a one-to-one mapping, i.e. time always increases and one time stamp is mapped to only one signal value, while a two-dimensional trajectory is a one-to-many mapping, i.e. there could be more than one latitude values associated with the same longitude (See Figure 4.5a). This difference results in two special challenges when applying time sequence segmentation techniques to trajectory partitioning. Fortunately, both challenges can be identified and fixed. For one thing, a vehicle might stop at one location (See Figure 4.5b left), either because of congestion, traffic light, or temporarily parking. In this case the data matrix will contain some identical rows, which makes it rank-deficient and the regression will be ill-conditioned. We can easily fix this by looking at the rank of the data matrix for the beginning part of a segment, and if it's deficient, just wait till enough data is collected to make it full rank. The other challenge is if a vehicle is traveling roughly along the longitude line direction (south-north), as shown in Figure 4.5b right, the slope will be close to infinite. Calculation with really large and small floating numbers will bring in various unexpected errors Goldberg (1991), so to avoid this we check the condition number of data matrix and if the condition number is close to infinity, we connect the start and end points to represent this segment without going through the regression process. Considering these two special cases, Algorithm 6 presents the sliding window procedures for online trajectory partitioning. The regression module is from the incremental linear regression algorithms (Incremental Householder QR or SVD) in Section 4.2. Figure 4.6 presents a cloud based compressed sensing system using the sliding window algorithm. (Icons used in the figure are from Wikimedia (2016); IconFinder (2016); Icons-Land (2016), used with permission.)

4.3.2 Error Options

The $MaxError$ parameter, the maximum error on a segment, determines where the trajectory gets partitioned. There are many options to measure the error. The commonly used ones are the Mean Squared Error (MSE), Cumulative Squared Error (CSE), and Maximum Individual Error (MAX). $MSE = \frac{1}{K}\sum\limits_{i=1}^{K}\left(\hat{y}_i - y_i\right)^2$, $CSE = \sum\limits_{i=1}^{K}\left(\hat{y}_i - y_i\right)^2$ and $MAX = \max(|\hat{y}_i - y_i|)$ (K is the number of points in that segment, $\hat{y}_i = Aw$ is the predicted value at point i by the regression model.) MSE offers an upper bound for the mean of the prediction error $E\left(|\hat{y}_i - y_i|\right) = \frac{1}{K}\sum\limits_{i=1}^{K}|\hat{y}_i - y_i| \leq$



(a)



(b)

Figure 4.5: Differences between time sequence segmentation and trajectory partitioning.

$\frac{\sqrt{K}}{K}\sqrt{\sum_{i=1}^{K}|\hat{y}_i - y_i|^2} = \sqrt{MSE}$, while MAX is the upper bound of individual prediction errors $|\hat{y}_i - y_i| \leq MAX$.

4.3.3 A Case Study

Figure 4.7 is an example of segmenting a real GPS trajectory. The data is provided by Paul Newson and John Krumm from Microsoft Research, Seattle Newson and Krumm (2009). We choose CSE as the error measure, and $MaxError$ is 10^{-8}. Figure 4.8 compares the sampled trajectories using different $MaxError$ values. As $MaxError$

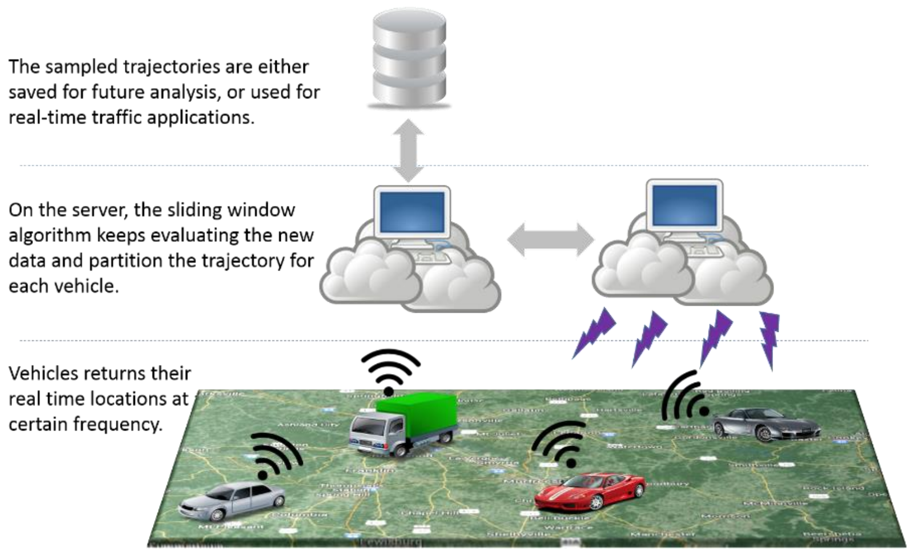


Figure 4.6: A cloud based real-time trajectory collection and processing system.

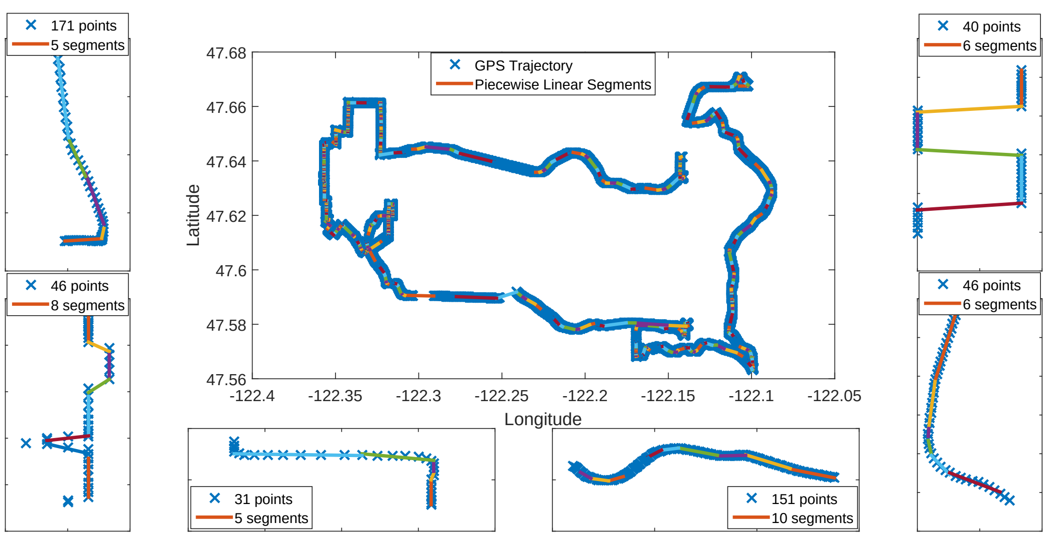


Figure 4.7: An example of trajectory partitioning (CSE, $MaxError = 10^{-8}$).

goes higher and higher, there are more and more points removed, however, we can tell that the shape is still preserved well, because the segmentation skips only the points that are col-linear with others, while the "points of interest", the turning points that determines the shape, are still preserved.

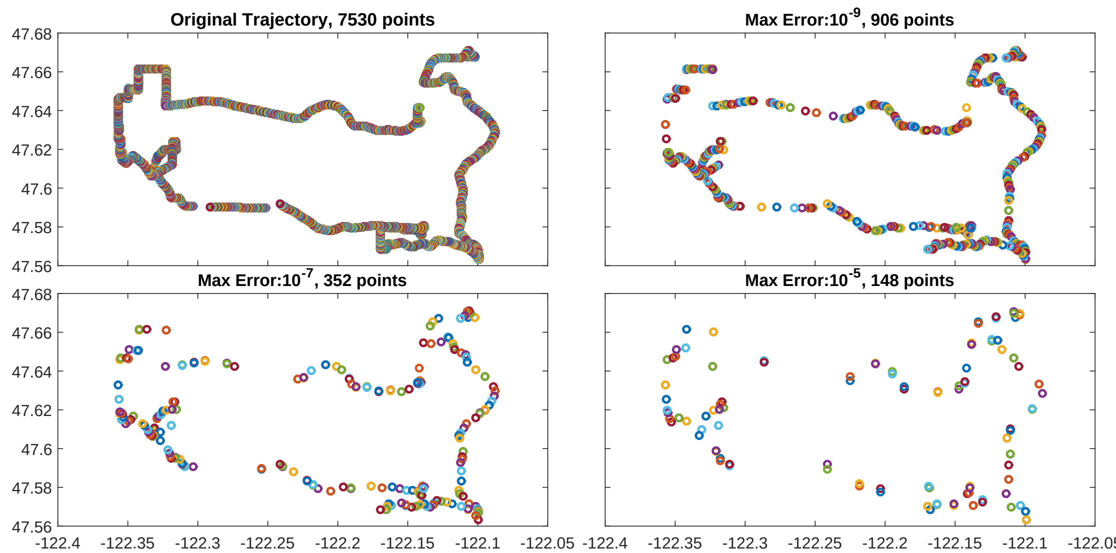


Figure 4.8: Downsampled trajectory using different $MaxError$ values (CSE).

4.4 Discussion

4.4.1 Accuracy Control and Compression Power

We use Compression Power (CP, also known as Compression Ratio Basnayake et al. (2011)) to measure the data reduction ability of the algorithm. CP is simply the ratio of the numbers of points before and after the compression. A smaller $MaxError$ value will partition the trajectory more frequently, preserves more points and so has a lower compression power, while a bigger $MaxError$ allows more points to lie in the same segment and so results in a higher compression power. We are concerned about CP because it represents how much storage and computational resources we can save, however, we are also concerned about the accuracy of the sampled trajectory, i.e., how authentic the sampled data is compared to the original data. Although a low $MaxError$ value reduces the trajectory size by a few thousand times, a large bias off the true locations could make it impossible to reconstruct a vehicle's trajectory.

Since our segmentation algorithm is based on linear regression, a statistical sound way to measure the prediction quality at an individual point is the 95% Confidence Interval (CI). We then use the length of the 95% CI to represent the maximal possible

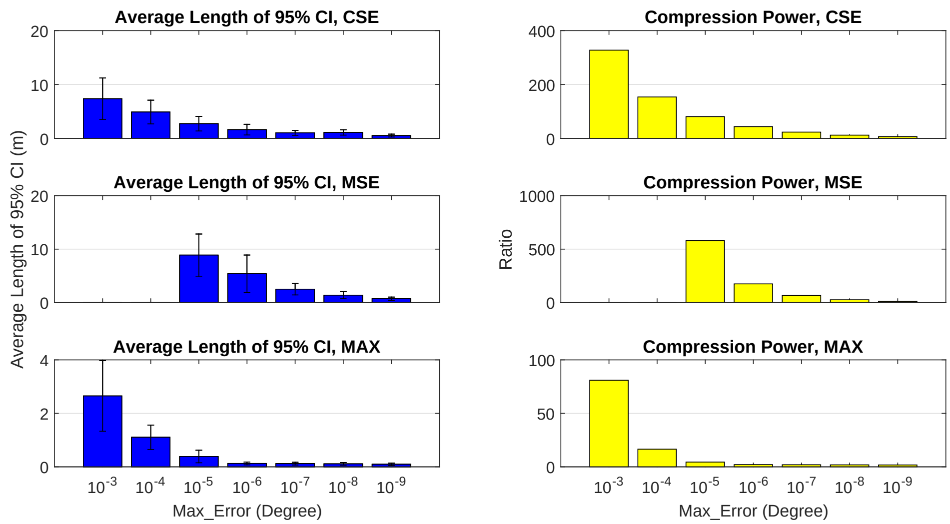


Figure 4.9: Left: The association between CI and $MaxError$. Right: The association between CP and $MaxError$. Each row uses a different error option

error for 95% of all the predictions. 4.9 presents the association between CI's, CPs and Max_Error under different error options (CSE, MSE, MAX). Since our data are longitudes and latitudes, the direct CI of the prediction is a latitude range, which is in degrees, however, since we are more used to interpret length, we can easily convert it to meters: $l = \frac{6.371 \times 10^6 \theta}{360}$.

Here are our observations:

(1) To achieve the same CI's length and compression power, the $MaxError$ values for three types of error options generally have the following relationship: $MAX > CSE > MSE$. This is not hard to understand, because CSE is cumulative error while MSE is the average, so to partition a trajectory to the same number of segments, we need a higher CSE than MSE. On the other hand, MAX is the maximum individual error, a small MAX will be higher than the prediction errors at most points and so will over-partition the trajectory. Only if we allow a higher maximum error, we will be able to avoid over-partitioning the trajectory.

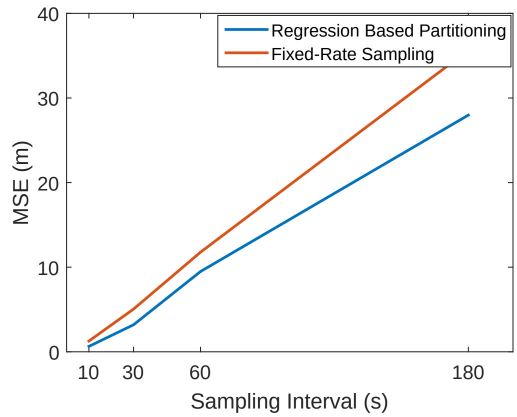
(2) The three types of error options offer different ranges of "good $MaxError$ values". By saying a $MaxError$ is good, we mean it leads to both a narrow CI range and a large CP.

As shown in Table 4.2, CSE has the widest range ($10^{-4} \sim 10^{-8}$) of good $MaxError$ values, while MSE ($9 \times 10^{-7} \sim 10^{-9}$) and MAX ($10^{-3} \sim 5 \times 10^{-5}$) are a lot narrower. In practice, a wider parameter range means it is easier to tune up. With the best parameter range hidden in a narrow range, it becomes easy to miss the best range.

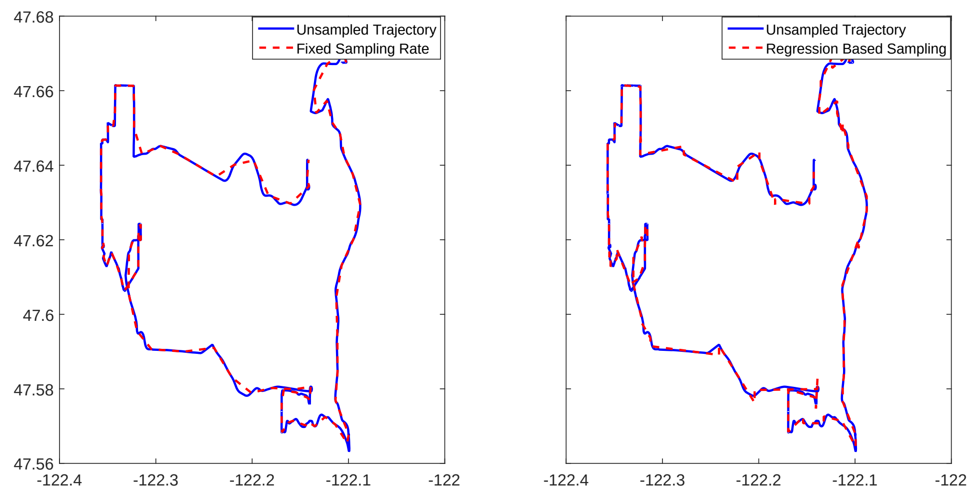
Notice that the good $MaxError$ values in this study are not guaranteed to apply to other trajectory data. GPS devices have different accuracies Ahmed-Zaid et al. (2011); Basnayake et al. (2011), trajectories collected by a high precision GPS device are less noisy than those by lower precision devices, and so need a small $MaxError$ value to capture the turning points. Different applications might have different requirements on the compression ratio and the accuracy of the sampling. For a specific application and GPS device, one can use some trajectory samples, try different $MaxError$ values and record the CI lengths and compression ratios, and decide the optimal value to use.

4.4.2 Comparison with Fixed-Rate Sampling

We compare the accuracies of our stepwise regression based sampling method with fixed-rate sampling. Given a fixed sampling rate, we adjust the $MaxError$ value in our regression based method to make the two achieve about the same compression power. We connect the sampled points to get the reconstructed trajectory, and compare it with the true trajectory (unsampled). We use $MSE = \frac{1}{K}\sum_{i=1}^{K}(\hat{y}_i - y_i)^2$ as the accuracy measure, where the true trajectory contains K points, and $\hat{y}_i$ is the predicted longitude value of point i by the reconstruction, and y_i is its true longitude. Figure 4.10a shows that no matter what the sampling rate we use, regression based sampling is always more accurate than fixed rate sampling. Figure



(a) Trajectory reconstruction accuracy using the two methods



(b) Differences of the reconstructed trajectories with true trajectory

Figure 4.10: Comparison of regression based partitioning and fixed-rate sampling.

4.10b is a visualization of the difference, where we can see clearly that fixed-rate sampling misses many "points of interests" and makes the reconstructed trace visibly different from the actual trajectory.

4.5 Conclusion and Future Work

We tackle the big data explosion challenge in transportation. While GPS trajectories offer a lot of valuable information to support both real-time applications and various research studies, the increasing volume makes data collection, transferring, storing and processing a big challenge to traditional storage and computing resources. We look into how trajectory data can be collected instantaneously, economically and authentically. We have shown that Row-wise Givens rotation enables QR decomposition to be implemented incrementally, and incremental SVD can accommodate new-coming data at a constant cost. Both of these factorizations are good options for online linear regression. Overcoming two special challenges, the sliding window method and incremental linear regression work together to partition and sample the vehicle trajectory in a real-time manner. The performance of this method is evaluated based on both compression power and the accuracy. By choosing a proper maximum error value, the algorithm can achieve both a significant compression power (e.g. over 10:1) and a small bias (e.g. less than 5 meters). Regression and sliding window based segmentation always gains a better accuracy than fixed-rate sampling when reconstructing the traces.

There are a lot more to investigate to push this method forward. We have shown many applications utilizing trajectory data. It is worth looking into how these applications' performances vary with the data accuracy level. If the application does not require high precision data, our method can compress the trajectory at a large ratio, however, if a big compression ratio makes the application unreliable, the segmentation would be ineffective and should not be used. Our method should be validated on an application-by-application basis.

Algorithm 5: Incremental SVD for linear regression

Incremental−SVD

Input : new data $X_{(m+1)}$, y_{m+1}, stored $U_{m\times m}$, $S_{m\times(n+1)}$ and $V^T_{(n+1)\times(n+1)}$ from last step

Output: w

Step 1: $m = m+1, (m \geq n)$

Pad a zero row to U :

$$U = \begin{bmatrix} U \\ \mathbf{0} \end{bmatrix}$$

Step 2: Update B:

$$B_{(m+1)\times 1} = \begin{bmatrix} B_{m\times 1} \\ y_{m+1} \end{bmatrix}$$

Step 3: Define:

$$a = [\ 0 \quad \cdots \quad 0 \quad 1\ ]^T_{1\times(m+1)}$$

$$b = \begin{bmatrix} 1 & x_1^{(m+1)} & x_2^{(m+1)} & \cdots & x_n^{(m+1)} \end{bmatrix}$$

Step 4: Calculate:

$$g = U^T a, p = a - Ug, R_a = \|p\|, P = R_a^{-1} p$$

$$h = V^T b, q = b - Vh, R_b = \|q\|, Q = R_b^{-1} q$$

$$K = \begin{bmatrix} S_{1:m,1:m} & \mathbf{0}_{m\times 1} \\ \mathbf{0}_{1\times m} & 0 \end{bmatrix} + \begin{bmatrix} g \\ R_a \end{bmatrix} \begin{bmatrix} h \\ R_b \end{bmatrix}^T$$

Step 5: SVD on K:

$$[\ U_t \quad S_t \quad V_t\ ] = \mathrm{svd}(K)$$

Update U, S and V:

$$U = \begin{bmatrix} U & P \end{bmatrix} U_t$$

$$S = S_t$$

$$V = \begin{bmatrix} V & Q \end{bmatrix} V_t$$

Step 7: Solve w:

$$Sw = VU^T B$$

Return w

Algorithm 6: Sliding Window Method for Online Trajectory Partitioning

```
Trajectory−Partitioning (Lon, Lat, MaxError)
    Input  : Trajectory data flow Lon and Lat, maximum regression
             prediction error MaxError
    Output: Downsampled trajectory
    start ← 0
    end ← rank
    while end < length of data or data stream is active do
        if Lon(start : end) is rank-deficient then
            continue
        else if Lon(start : end) has a large condition number then
            w ← Inf
            error ← 0
        else
            [w, error] ← Incremental_Regression(Lon(start : end))
        end
        if error < MaxError then
            end ← end + 1
        else
            result.push_back(start, end, w)
            start ← end
            end ← start + rank
        end
    end
```

Table 4.2: Good $MaxError$ values

	CSE					MSE				MAX		
$MaxError$	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	9×10^{-7}	10^{-7}	10^{-8}	10^{-9}	10^{-3}	10^{-4}	10^{-5}
CI(m)	4.89	2.73	1.63	1.03	1.11	4.26	2.52	1.39	0.73	2.65	1.11	0.75
CP	153.67	50.87	43.78	21.39	12.05	175.12	66.64	26.61	11.48	80.97	16.51	10.62

Chapter 5

Hierarchical Network Clustering: Energy-Efficient Infrastructure Deployment on Road Network

5.1 Introduction

The past few years have seen exceptional research accomplishments in artificial intelligence, cognitive science, computer vision, sensors, communication, etc. Lake et al. (2016). They are reforming the way we sense the physical world, acquire data, build intelligent control strategies and predict the future. A new generation of technology is launched and these unprecedented technologies are also impacting transportation domain and leading us to a new era of traveling and traffic monitoring. Drones are being used to monitor real time traffic conditions. The advances of deep learning in computer vision will offer a drone the "superpower" to "watch" the traffic scene in the sky and extract the exact instantaneous travel data on a network scale, which will further facilitate many remarkable real time applications that we have been dedicated to for decades, such as automated traffic monitoring and seamless emergency response, accurate travel time predictions, personalized routing recommendation, etc. In special cases, such as manhunts for terrorists, jail breakers and other escaping criminals, full coverage of the road network using drones of "sharp" eyes will dramatically increase the chances of success and save law enforcement resources. Coverage of road networks with drones have applications in other situations as well, such as military surveillance, emergency evacuations, etc. The other upcoming momentous technology is the connected and autonomous vehicle (CAV) Zorbas and Douligeris (2011), which is a gathering of all kinds of cutting edge technologies in computer vision, machine learning, control, sensing, communication, and so forth. Dedicated Short Range Communications (DSRC) is used for vehicle-to-vehicle and vehicle-to-infrastructure real time information sharing Ahmed-Zaid et al. (2011). Its instant acquisition of network connection, low latency (millisecond-level), high reliability in inclement weather and other features makes DSRC so far the best choice for active safety purpose Zorbas and Douligeris (2011); Kenney (2011). The next generation 5G cellular network, although not sure if is able to replace DSRC for active safety communication purpose, will at least be a supplement to DSRC to provide non-safety related on-board Internet accesses such as in-vehicle entertainment Zorbas and Douligeris (2011). Another possible necessity of 5G cellular in vehicular networks is the support for the communication between individual vehicles and a remote centralized server. Behind an autonomous vehicle are the supporting algorithms, databases and endless computations. The computations take place in a centralized server instead of on board. For example, deep learning enables powerful machine vision to detect the complicated and dynamic traffic scenes involving vehicles, bikes, pedestrians, roadways, signs and so on. The essential algorithm is the convolutional neural network (CNN), which has millions of parameters stored in large scale graphs Krizhevsky et al. (2012) that can only be accommodated by data centers, but not an on-board computer. With that being said, high speed and reliable communication covering the entire road network is necessary to guarantee the proper functionality of the CAV system.

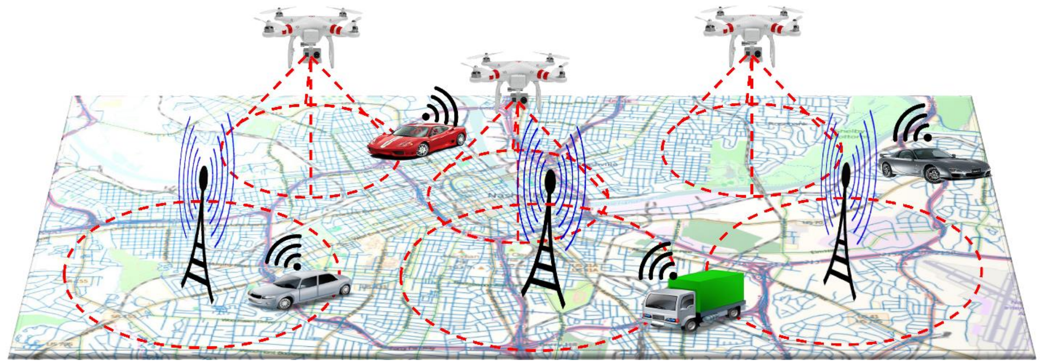


Figure 5.1: Applications of new technologies in transportation system.

These new technologies brings in new challenges when we deploy them in practice. Figure 5.1 pictures the deployment of the new technologies to serve our road network. To cover all the roads over a large area, we need to decide how many drones/helicopter to deploy and the regions each of them is in charge of. Similarly, for the 5G network to support CAVs, we need to decide where and how many communication base stations to build. On one hand, sufficient infrastructures (drones, base stations) should be distributed to cover road network completely in order to provide consecutive and reliable services. On the other hand, the overall "cost" of the system should be minimized. The cost consists of two parts: the installment and maintenance cost, and the energy/power cost. Generally, there is a trade-off between the quantity of infrastructures and their individual coverage ranges. More infrastructure deployments means higher installment and maintenance cost, but each of them have a smaller range to cover, which cuts down fuel/energy consumption. The optimal configuration selects a moderate quantity with moderate coverage ranges to achieve a minimal overall cost.

Coverage problem is a well-studied problem in wireless sensor network (WSN) domain. A number of literatures have offered thorough overviews on a broad scope of sub-topics in WSN coverage, including sensing and communication geometrical models, design requirements, coverage algorithms, sensor activity scheduling Cardei and Wu (2006); Deying and Liu (2009); Mulligan and Ammari (2010); Wang (2011); Mahfoudh and Minet (2008), etc. Sensors are distributed to monitor the targets in a known or unknown environment. The sensors can be deployed deterministically in a completely known environment, but in most applications, people rarely have enough knowledge of the monitored targets and the sensors are distributed to the space randomly through airplanes or other tools. The positions of the randomly distributed sensors can be obtained through equipped positioning modules. A sensor basically has two states: active and sleep. An active sensor senses the states of the targets, disseminates the signal to the server or its neighbor sensors. A sensor in sleep mode does not sense but can get reactivated to work mode. Normally, there are more than enough sensors to cover each target. The lifetime of the sensor network is constrained by the battery life of the individual sensors, and the objective of a coverage algorithm is to make the system energy-efficient, i.e. to maximize the WSN lifetime through matching sensors with the targets they are in charge of, and designing active-sleep schedules. There are two types of targets: area and points. Area coverage normally first divides the space into smaller "fields" using approaches like Voronoi diagram, and then applies point coverage techniques Mulligan and Ammari (2010). For point coverage, "cover sets" is a commonly used approach. One class of "cover sets" method divides sensors into disjoint subsets. A disjoint subset can individually cover the entire target set, and one sensor belongs to only one disjoint subset. A "disjoint set cover (DSC)" method finds the maximal number of disjoint sets from a given sensor set, because the more disjoint sets there are, the more available shifts can be scheduled to work, and so the longer the network lifetime will be. Cardei and Du proved that DSC problem is NP-Complete, they then converted it to a maximum flow problem and solved it using mixed integer programming Cardei and Du (2005). Cardei and Wu attacked a similar problem but instead of having fixed sensing ranges, the sensors have adjustable sensing ranges and the objective is generate a maximal number of disjoint sets, each with the smallest possible sensing range to conserve power Cardei et al. (2005b). It was again formulated as a linear integer programming problem and solved with both mathematical programming and two different heuristics. The authors concluded that adjustable sensing range makes a big difference in extending the network lifetime Cardei et al. (2005b). Cardei and Thai studied the "maximum cover problem", which is a similar setting but a sensor is allowed to work for multiple disjoint sets Cardei et al. (2005a). This is based on the observation that sensors are not distributed evenly in the target space, and so some are more critical than others to cover the targets. Instead of making every disjoint work for the same period of time, this model makes the active time of each disjoint cover set variable. With the same objective of maximizing network lifetime, the model constraints the total service time of every sensor such that they do not run out of battery. Experiments conducted have shown that the new algorithm can further extend a WSN's lifetime. To consider a more practical and dynamic situation where the targets might have different priorities of being covered and sensors' battery level varies, Zorbas et al proposed a cover set creation strategy based on a cost function where factors such as a sensor's sensing range, a target's critical factor, remaining battery of the sensor etc. determines which targets the sensor will cover in the next cycle Zorbas et al. (2010). DSC algorithms work for problems on small spatial scale. For sensors distributed on a large scale space that exceeds their sensing range, there might not be many disjoint sets. Non-disjoint cover sets method can handle larger scale problems. An individual non-disjoint cover set cannot independently monitor the full target set, but a selected combination of them can do it. Besides, a sensor can belong to more than one non-disjoint sets. Representative studies of non-disjoint coverage include determining the minimum breach using integer programming Cheng et al. (2005, 2007), the connected coverage and efficient scheduling Zhao and Gurusamy (2008); Kasbekar et al. (2011); Yang et al. (2006), etc.

Coverage problems are studied in other domains as well. Given a set of point targets, and another set of discs, the Minimum Unit Disc Coverage problem, chooses the minimum number of discs to cover the entire target set Acharyya et al. (2012); Fu et al. (2007). The problem is applied to facility and service location selection, such as choosing minimal number of locations to build fire stations but ensuring easy access for everybody in a county. Other problems includes edge covering problem in graph theory, polygon covering, clique cover, art gallery problem Wikipedia (2014); Sun et al. (2008), etc. Most of the coverage problems discussed above originate from to the classical set covering problem in computer science. According to Sun et al. (2008); Wikipedia (2014); Har-Peled and Lee (2012); Mecke and Wagner (2004), this class of problem can be formulated as follows: A finite set of targets $P = \{p_1, p_2, \cdots p_n\}$, a set of covers $S = \{s_1, s_2, \cdots s_k\}$, where each of the element covers one or more elements of P, and the associated cost set for the covers $C = \{c_1, c_2, \cdots c_k\}$. The objective is to determine an optimal subset S^* of S, such that S^* covers the entire set P with minimal total cost. Although there are many variants based on different contexts, most of these problems can be solved using linear integer programming Wikipedia (2014).

The road network coverage problem we are studying is different from traditional coverage in mainly two ways: (1) In traditional coverage problems, the cover set is given. In WSN coverage, sensors' locations are known; in minimum disc coverage, the centers and radii of the candidate discs are given; in set covering, the cover set is known. Solving these coverage problems is to choose the optimal candidates from the cover set. In our road network coverage, however, the candidate base locations are not available. This adds a lot more variability and complexity to the optimization problem. (2) The cost set cannot be pre-defined. Due to lack of the candidate covers, we are not able to list the costs of which cover covers which segments of the network. In the next Section, we will start with formulating our problem as an optimization problem, and then show that the lack of the cover and cost sets makes it impossible to solve it using integer programming techniques. In Section 5.4 and 5.5, we will present two hierarchical heuristic algorithms to approximate the optimal solution. Session 5.6 is a case study using real world network. We also analyze the complexities and compare the run-time performances of the two algorithms. To conclude the paper in Section 5.7, we overview other practical considerations in road network coverage applications, and how the framework in this study can be applied to those scenarios.

5.2 Problem Formulation

We first clarify the terms we will be using. We will use a graph data structure to represent the road network, so we will be calling it graph or network. Since the terms vertex and edge are preferred in graph theory/algorithms, we will call a road segment an edge or link, and call the intersection of two road segments a vertex or node. We use the word "base" to refer to drones, communication base stations and any other types of infrastructures that are used to cover the road network in a specific application.

The road network is represented as a graph $G = (E, V)$, where E is its edge set and V the vertex set. We are to cover G with K bases, and each of them has an adjustable coverage range r. Due to the power/sensing range limit, r has an upper bound R. Let's try to formulate it to be an integer programming problem. Variable x_{ij} represents the i^{th} edge e_i is covered by the j^{th} base b_j. The objective is to minimize the total coverage areas:

$$\min_{x_{ij}} \quad \sum_{j=1}^{K} S(b_j) \tag{5.1a}$$

$$\text{s.t.} \quad S(b_j) = S(\bigcup_{i=1}^{|E|} e_i x_{ij}), \quad j = 1, 2..., K, \tag{5.1b}$$

$$\sum_{j=1}^{K} x_{ij} \geq 1, \quad i = 1, 2..., |E|, \tag{5.1c}$$

$$S(\bigcup_{i=1}^{|E|} e_i x_{ij}) \leq \pi R^2, \quad j = 1, 2..., K, \tag{5.1d}$$

$$x_{ij} = \{0, 1\}, \quad i = 1, 2..., |E|, j = 1, 2..., K \tag{5.1e}$$

Remark:

(1) In constraint 5.1b, since the bases' locations and areas are not given, and they are determined by the edges the base covers. The coverage area of the base can be calculated as the area of the bounding circle the edges it covers.

(2) Constraint 5.1c guarantees every edge should be covered by at least one bases.

(3) Constraint 5.1d interprets the maximum sensing range for all bases.

(4) Constraint 5.1e forces x_{ij} a 0-1 integer variable.

The difficulty of solving the problem using integer programming techniques is that the mapping between the objective function and the variables is not explicitly defined. We all know that, for a linear integer programming problem, the objective is a linear combination of the independent variables; for a nonlinear integer programming, the objective function is a nonlinear combination of the independent variables. The commonly used search strategies in solving nonlinear optimization involve gradient and hessian, which is based on a well-defined and differentiable objective function. However, there is not a universal mathematical formula between the coverage area of a base $S(\bigcup_{i=1}^{|E|} e_i x_{ij})$ and the edges/vertices it covers, as we will show in section 5.3. Evaluating the coverage area for a set of edges is an iterative process, and so trying to find the optimal value for the objective function becomes an enumeration process, which is other words a brute force way. In Section 5.4 and 5.5 we will present two heuristics to approximate the optimal solution, but before that, we first present the steps of getting the minimum bounding circle of a set of edges. Our purpose is to show that this is an iterative process and there is no definable formula.

5.3 Minimum Bounding Circle of a Set of Edges

The minimum bounding circle of an edge set is equivalent to that of the vertices. Determining the minimum bounding box is a fundamental problem in computational geometry. An efficient way is to first get the convex hull of the points. Since most commonly seen shapes such as circle, rectangle, triangles, and ellipses are convex, the bounding box is determined by the points on the convex hull, and so the points inside of the hull can then be ignored to significantly reduce the computational cost. We then can iteratively determine the parameters of the bounding shape. For example, to determine the minimum bounding circle, there are three parameters: center of

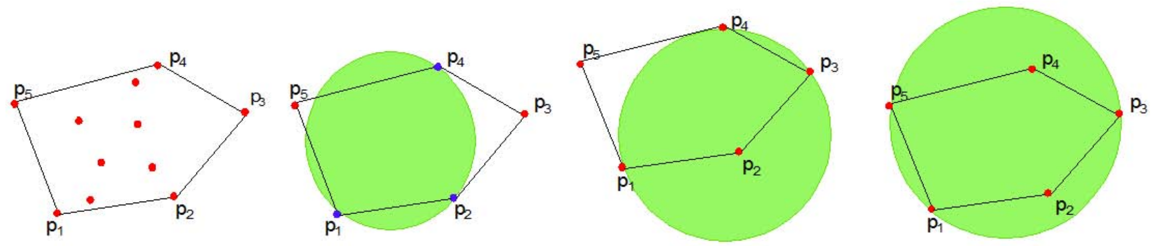


Figure 5.2: An example of getting minimum bounding circle of a set of points.

the circle (c_x, c_y), and the radius R. We define two sets to store the points, set S_1 storing the points included on and inside the circle, and set S_2 containing the points outside the current circle. Initially all points on the convex hull are in S_2. We initialize the problem by randomly picking three points from S_2 to solve for a circle C and move these points to S_1. We then check if other points are inside C and update S_1 and S_2. If S_2 is empty, then C is the minimum bounding circle we are looking for, otherwise we move the furthest point (to the current circle) from S_2 to S_1 and recalculate a circle C to cover S_1. Since C keeps expanding to include new points, the process eventually terminates when all points are included inside C. We will refer this process as *Min_Bound_Circle(edges)* in later sections of this article. Figure 5.2 shows a simple example of getting the minimum bounding circle following this process: (1)Get convex hull. $S_1 = \emptyset$, $S_2 = \{p_1, p_2, p_3, p_4, p_5\}$. (2) Initialization. $C = Circle(p_1, p_2, p_4)$. $S_1 = \{p_1, p_2, p_4\}$, $S_2 = \{p_3, p_5\}$. p_3 is the furthest from C. (3) Update C. C includes p_3. $S_2 = p_5$. (4) Update C. C includes p_5. $S_2 = \emptyset$. Algorithm terminates.

Note that although we use circle as the coverage shape in our study, this convex hull based process applies to other shapes as well. Different shapes have different equations to solve. For example, if the coverage shape is not circle but an ellipse, which has four parameters: center (c_x, c_y), and the axis lengths a and b. We then need four points and four equations to solve for the initial ellipse and updating it when merging a new point.

5.4 A Hierarchical Heuristic for Road Network Coverage

Earlier we have shown there is no good way to get the theoretical optimal solution to the network coverage problem, and the best we can do is to approximate it. We approximate the optimal settings of the bases using a hierarchical clustering approach. Given a set of points, hierarchical clustering has an objective of minimizing the total within-cluster variance. Ward's method starts with each point being a cluster. In each iteration, two clusters that will lead to minimum increase of the total within-cluster variance will be chosen and merged Wikipedia (2014). This agglomerative hierarchical clustering framework can be applied to determining base locations and coverage ranges in our road network coverage application. Our objective is to minimize the total coverage area. Here a cluster is not a subset of points, but the bounding circle of a subset of edges. Using the hierarchical clustering framework, we initially create the clusters from each road segment of the road network, then in each step, find and merge two clusters that lead to minimum increase of total coverage area. The algorithm will terminate when the number of clusters reaches the requested quantity, or keeping merging will violate the constraints, for example merging any two clusters will make the radius larger than the maximum radius R. The bases should be set at the centers of the cluster circles and their coverage ranges are determined by the radius.

Table 5.1 shows the "cluster" data structure. Algorithm 7 shows the initialization of the *cluster_list*. A cluster initially comes from a single edge. A cluster has its "*closest_cluster*", which is obtained, as shown in Algorithm 7, by finding the cluster that can be merged with a minimum area enlargement. The affiliated minimum area enlargement is named "cost" here. Algorithm 8 presents the merging process. In every iteration, a *merge_cluster* is selected, which induces a minimum increase of total coverage area, while considering the maximum radius constraints and clusters broader than R will not be selected to expand again.

When a cluster merges its *closest_cluster*, a few updates follows. For one thing, the *closest_cluster* will be removed from the *cluster_list* because it has been merged. Before the removal, we first need to merge *closest_cluster*'s edges into the *merge_cluster*'s edges list. The expansion makes *merge_cluster* bigger and have an updated bounding circle and *closest_cluster*. For another, there might be other clusters that had *closest_cluster* as their *closest_cluster*, since *closest_cluster* is not a valid cluster any more, all those clusters will have to update their *closest_cluster(s)*. The clusters keep expanding until they all reach the maximum radius limit, and eventually no cluster will be able to expand, so the algorithm terminates with the minimum number of cover circles.

Let's analyze the computational complexity of this heuristic approximation process. Suppose the size of the network's edge set $|E| = n$. In the initialization, *Get_Closest_Cluster(cluster, cluster_list)* is a $O(n)$ process that obtains the *closest_cluster* for each cluster by searching over the entire *cluster_list*. The initialization part has a $O(n^2)$ complexity. In the cluster merging process, *Get_Merge_Cluster(cluster_list, R)* iterates through the *cluster_list* and has a $O(n)$ complexity. *Min_Bound_Circle(merge_cluster.edges)* is a local operation because it involves the edges of only the *merge_cluster*, not of other clusters, so its contribution to the overall complexity can be ignored compared with $O(n)$. The merging process is controlled by a *while()* loop, and if there are K clusters merged successively, which means K iterations, then the complexity is $O(nk)$, the worst case is $O(n^2)$ since $K \leq n$. So this is an $O\{n^2\}$ algorithm. The initialization has comparable complexity as the merging iterations. This algorithm works fine for a small scale road network, however, the fact is that even a county level road network can have over a few thousand edges. Our experiments show that this algorithm runs too slow for a medium scale network. So we proceed to seek a lower complexity.

Table 5.1: Data structure of a cluster

Filed Name	Description
id	equal to the id of the initial edge
edges	a list of the edges covered by the cluster
closest_cluster	the cluster to merge with minimum area enlargement
cost	the area enlargement when merging
center	(longitude, latitude)
radius	radius of the bounding circle

Algorithm 7: Initializing *cluster_list*

Initialization $(E, V, \varepsilon, minpts)$
  Input : Edge set E, Graph G
  Output: Initialized *cluster_list*
  foreach *edge* $e \in E$ **do**
    $id \leftarrow e.id$
    $edges \leftarrow \{e\}$
    $center \leftarrow e.midpoint$
    $radius \leftarrow e.length/2$
    $cluster\_list.add(\textbf{cluster}(id, edges, center, radius))$
  end
  foreach $cluster \in cluster\_list$ **do**
    $[closest\_cluster, cost] \leftarrow \textbf{Get\_Closest\_Cluster}(cluster, cluster\_list)$
  end

Get_Closest_Cluster $(cluster, cluster\_list)$
  $closest\_cluster \leftarrow \underset{c \in cluster\_list, c \neq cluster}{\arg\min} EnlargeArea(cluster, c)$
  $cost \leftarrow EnlargeArea(cluster, closest\_cluster)$

5.5 An Accelerated Heuristic Utilizing Local Search

The main reason the above algorithm is has a time consuming $O(n^2)$ is that *Get_Closest_Cluster()* compares a cluster with the entire cluster set, which initially is the edge set E. When we have a large road network, E is huge and searching over it makes the algorithm slow. The fact is that for an individual edge, its closest edge should most likely be within the ones directly connected to it, so a global search is unnecessary. In this session, we present an accelerated implementation utilizing the connections between the road segments.

Algorithm 8: Hierarchical Merging of *cluster_list*

```
Hierarchical_Merge (cluster_list, R)
    Input  : Initialized cluster_list, constraint R
    Output: Final cluster_list
    /* Determine which cluster will merge with its closest_cluster
       */
    merge_cluster ← Get_Merge_Cluster(cluster_list, R)
    while merge_cluster is not empty do
        /* Merge merge_cluster and its closest_cluster                */
        merge_cluster.edges ← merge_cluster.edges ∪ cloest_cluster.edges
        [merge_cluster.center, merge_cluster.radius] ←
         Min_Bound_Circle(merge_cluster.edges)
        cluster_list.remove(closest_cluster)
        /* if merge_cluster is also the closest_cluster of any
           other clusters, update their closest_cluster                */
        foreach clhster ∈ cluster_list do
            if cluster.closest_cluster is closest_cluster then
                [cluster.closest_cluster, cluster.cost] ←
                 Get_Closest_Cluster(cluster, cluster_list)
            end
        end
        merge_cluster ← Get_Merge_Cluster(cluster_list, R)
    end

Get_Merge_Cluster (cluster_list, R)
    min_cost ← infinity foreach cluster ∈ cluster_list do
        /* Get the new radius if cluster merges its cloest_cluster, the
           merge is forbidden if this radius exceeds R                */
        [center, radius] =
         Min_Bound_Circle(cluster.edges ∪ cluster.closest_cluster.edges)
        if radius > R then
            Continue
        else if area < min_cost then
            min_cost ← radius
            merge_cluster ← cluster
    end
```

An efficient and commonly used representation of a graph $G = (V, E)$ in computer memory is the adjacent list $G = \{s_i \rightarrow (e_j, t_j, w_j) | i = 1, ... |V|, j = 1, ... |N_{s_i}|\}$, where s_i is a source node, connected to target node t_j through edge e_j that carries a weight w_j. N_{s_i} represents the set of neighbors nodes connected to source s_i. In some applications such as GPS navigating, a road network is a directional graph, i.e. a vehicle drives from a source to a target, and if a road is two-way, there will be another edge going from the target to the source. Since our coverage application does not differentiate traffic flow directions, we do not keep the concept "source" and "target", so all edges are bidirectional, i.e. when there is a connection $s_i \rightarrow (e_j, t_j, w_j)$, there is also a $t_j \rightarrow (e_j, s_i, w_j)$. This simplification gives each cluster an equal chance to reach out and merge with its neighbors.

The adjacent list provides an easy way to extract the adjacent edges for each edge e. We denote the adjacent edges(clusters) as adj. For each node v, we get its neighbors $G(v) = \{(e_j, t_j, w_j) \,|j = 1... |N_v|\}$, then $\forall j = 1, ... |N_v|$, $e_j.adj = \{e_k \,|k = 1, ... |N_v| \,, k \neq j\}$. An edge becomes an adjacent edge of all other edges connected to the same node. Compared with the previous algorithm, the "cluster" data structure has one new field: the adjacent clusters list adj.

The *Get_Closest_Cluster(cluster,adj)* function becomes a local search within *adj*:
$closest\_cluster = \underset{c \,\in\, cluster.adj}{\arg\min} \; EnlargeArea(cluster, c)$

Because of the new field *adj*, each iteration in the cluster merging process now has two extra updates. First, whenever a *closest_cluster* gets merged by the *merge_cluster*, in addition to updating the *merge_cluster* itself, the *merge_cluster* and the *adj* of *closest_cluster* are connected now and should become each other's adjacent clusters. Second, the *closest_cluster*, which is not an active cluster any more, might still be in the *adj* of other clusters. One straightforward way is to parse the entire cluster list to get it removed from the *adj* of each cluster, but this definitely adds to the complexity of the algorithm. We notice that *adj* is only used in the *Get_Closest_Cluster* step, so we will avoid its effects in a light-weight way: in *Get_Closest_Cluster* step, we first check if an adjacent cluster inside *adj* list is still a valid cluster in the *cluster_list*. If it has already been removed, it will not be considered for *closest_cluster* voting.

This version of the clustering algorithm has a dramatic improvement in computational complexity. *Get_Closest_Cluster()* is a local search on the adj list of each edge. We assume on average one edge has M connections, then for an n-edge network,

Algorithm 9: Hierarchical Merging of *cluster_list* Utilizing Local Search

```
Get_Merge_Cluster (cluster_list, R)
    Input  : Initialized cluster_list, constraint R
    Output: Final cluster_list
    /* Determine which cluster will merge with its closest_cluster
       */
    merge_cluster ← Get_Merge_Cluster(cluster_list, R)
    while merge_cluster is not empty do
        /* Same routine as in Algorithm 8                          */
        merge_cluster.edges ← merge_cluster.edges ∪ cloest_cluster.edges
        [merge_cluster.center, merge_cluster.radius] ←
         Min_Bound_Circle(merge_cluster.edges)
        cluster_list.remove(closest_cluster)
        foreach cluster ∈ cluster_list do
            if cluster.closest_cluster is closest_cluster then
                [cluster.closest_cluster, cluster.cost] ←
                 Get_Closest_Cluster(cluster, cluster_list)
            end
        end
        /* update merge_cluster's adj                              */
        merge_cluster.adj ← merge_cluster.adj ∪ cloest_cluster.adj
        /* update the adj of the closest_cluster's adjacent
           clusters                                                */
        foreach cluster ∈ closest_cluster.adj do
            cluster.adj ← cluster.adj ∪ merge_cluster
        end
        merge_cluster ← Get_Merge_Cluster(cluster_list, R)
    end
```

the initialization step has a complexity of $O(nM)$. The merging process is still a $O(nK)$ process because of the while loop and *Get_Merge_Cluster()*. So the worst case complexity of the algorithm is still $O(n^2)$ when $K = n$. However, with a fixed maximum radius constraint R, a larger network will need more bases to cover, and so the number of clusters merged in the process will be far less than n. With $K << n$, the complexity of the algorithm when used on a large network will be much lower than $O(n^2)$. For the previous version in Section 5.5, the complexity is dominated by the initialization step, so the complexity is always higher than no matter what scale the network has. The change from a global search of *closest_cluster* to a local one over adjacent clusters list makes it significantly faster, as we will present in the following case study.

5.6 Case Study

We first select two local road networks to test the algorithms. Network 1 is from urban area where the roads are dense, the network has 331 segments. Network 2 is a suburban one, with 690 segments. We set the maximum radius for network 1 as 600 meters and 6000 for network 2. These two numbers are selected only for visualization purpose, because we want to show a proper number of circles in each case so readers can see clear results. In real applications, the maximum radius should be based on the infrastructure itself, e.g. the vision range of the drone and the communication range of the base stations, etc. Figure 5.3 shows the original road network, base settings and associations between the total coverage area and the number of bases deployed.

On one hand, the algorithm determines the minimum number of bases (clusters) needed to cover the entire network. For network 1, at least four circles are need with a maximum radius of 600 meters; for network 2, five circles whose radius is shorter than 6 km are needed at minimum. On the other hand, since the algorithm is a hierarchical process, we can also record the total coverage areas when changing the number of bases. For a dense network, if we use too many bases, because of the overlaps over each other, the total area is large; as the number of bases decreases, the overlaps get reduced and so does the total coverage area. However, when only a few bases are used, they cover not only the road network, but also a lot of extra space where no road is distributed, so the total area rises. So the minimum coverage area is reached when the number of bases is in the middle. The situation is different for the less-dense suburban network. The overlap is not obvious, so deploying more bases does not help reduce overlaps. Similar to the dense urban network, the total coverage area increases as the number of bases drops towards the minimum, for the same reason of covering extra space.

In real world, some applications might prefer the minimum number if the coverage area is not a concern, for example, when using drones to monitor traffic, the cameras' vision range is fixed, and our only goal is to find the minimum number of drones needed. But in other applications, where energy is a concern, we have to consider both the quantity and the energy consumption, which is related to the coverage areas. For example, when building communication base stations, there is a trade-off between how many stations to build and how large each station will cover. Using the minimum quantity, compared with more, means each of them covers a larger region and so consumes more power, so an objective function including the cost from both aspects is needed and the algorithm provides the coverage-area-versus-quantity data to obtain the optimal solution.

We further compare the run-time performances of the two versions of algorithms. Five networks of different scales are used and the number of edges ranges from 196 to 4011. We specially look at their total run-time and the time cost in the initialization step. Table 5.2 shows the results. It shows that accelerated version always outperforms the original version. One big difference between them is that over 40% of the run-time for the original version is spent on the initialization step, while the percentage of the initialization run-time of the accelerated version is a single digit. For the original version, as the network gets bigger, initialization run-time counts more and more in total run-time, while it is the opposite for the accelerated version. This is consistent with our earlier complexity analysis of the two algorithms, that for the original version, initialization step dominates the algorithm's run-time with a $O(n^2)$ complexity; while for the accelerated version, the initialization step has only a linear $O(nM)$ complexity and the major run-time factor is the merging process.

5.7 Conclusion and Potential Extensions

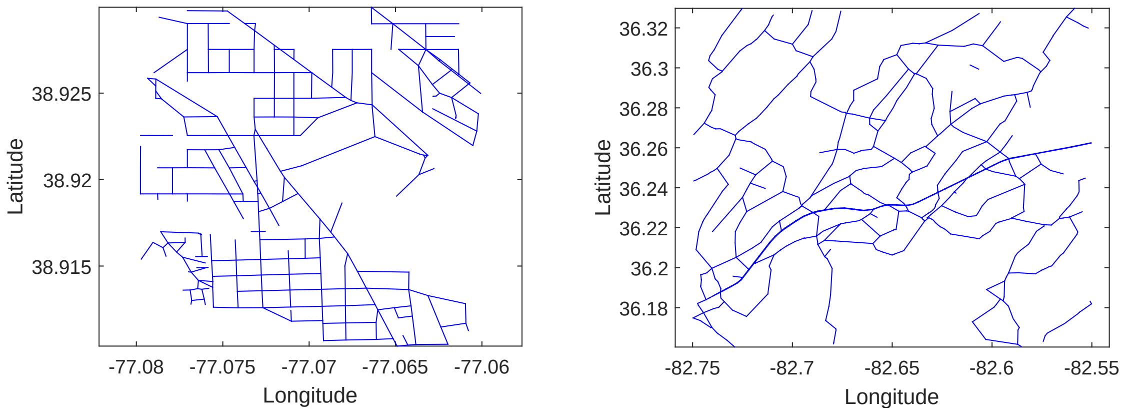
The emerging technologies enabled by recent advances in artificial intelligence, such as drones with intelligent recognition abilities, connected and autonomous vehicles,

Table 5.2: Run-time comparison of two algorithms on five networks

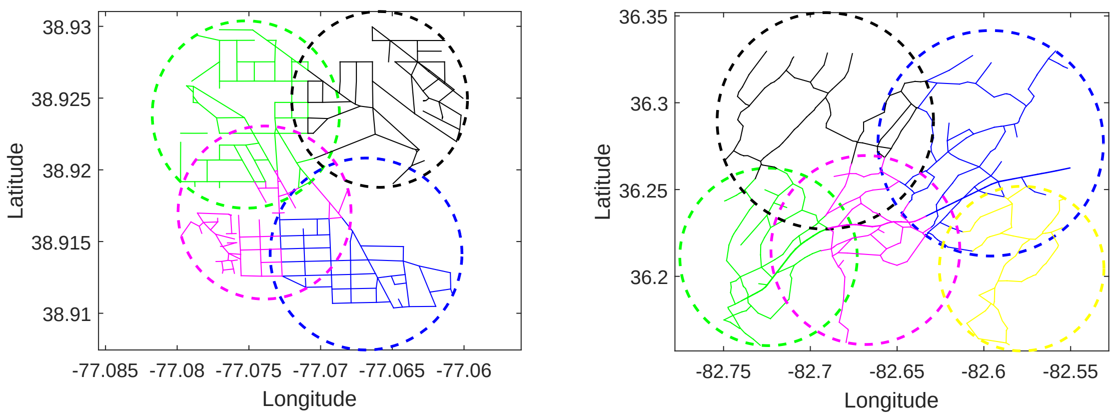
	Original version			Local search version		
Number of edges	Total (s)	Init (s)	Init/Total	Total (s)	Init (s)	Init/Total
196	132.416	57.568	43.47%	8.713	0.760	8.72%
331	389.776	174.493	44.77%	19.495	1.423	7.30%
690	1653.785	905.602	54.76%	38.167	1.969	5.16%
1688	–	–	–	231.695	5.180	2.24%
4011	–	–	–	1113.946	11.401	1.02%

are changing the ways we acquire real time traffic information, monitor and control traffic flow, and potentially redefines the roles of human beings when interacting with a vehicle. These new applications impose new demands of facility deployment and distribution algorithms over road networks. We have considered the road network coverage problem with an objective of minimizing coverage areas. Although there have been many coverage algorithms in other domains, the road network coverage is different mainly because there is no candidate base locations available. We have also shown that the problem cannot be solved using integer programming techniques because the objective function cannot be explicitly defined by the independent variables. We then propose two heuristic algorithms to approximate the optimal solution. The algorithms are similar to agglomerative hierarchical clustering where clusters starts as the individual edges of the network and merge to their "closest" clusters. The first version of the algorithm selects the closest neighbor by searching the entire cluster set; the second version accelerates itself by building and maintaining the adjacent cluster list and search locally. We conducted case studies using real networks. The accelerated version always outperforms the original version and is remarkably faster in initialization. In some distribution problems where energy is not a concern, the algorithm offers the minimum number of bases to cover the entire road network; for problems where both the number of bases and the coverage areas affect the overall cost, the algorithm support optimized decision making by providing complete information of the total coverage areas of all possible numbers of bases.

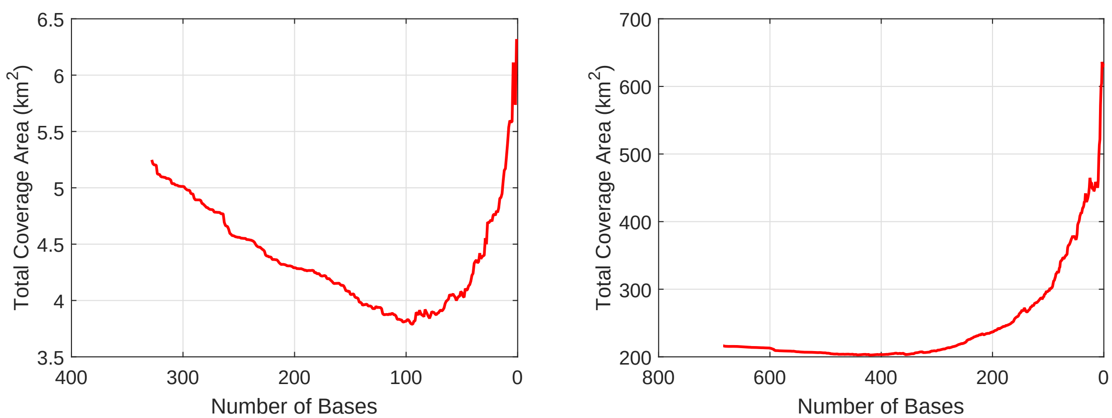
We consider this study as a preliminary framework for possibly a series of road network coverage problems. We can predict but have not yet looked into many practical scenarios. For example, we didn't consider usability the generated cover circles. If the center happens to be in water or inside a building, it would be impossible for us to set up a communication station. A further question following this is should we resolve situations like this by modeling other geographical components into the problem itself and how to interpret them as constraints? Or should we develop a "correction" approach to rectify the infeasible solutions afterwards? Other scenarios such as distributing heterogeneous facilities (instruments) instead of identical ones, i.e. the vision ranges are different for drones, communication base stations have different ranges, etc., and this obviously adds more variability to the problem. For another example, we did not consider the bandwidth constraints for the communication bases station application, however, in reality, there might be an upper bound on the number of devices (which is vehicles here) being hooked up with each station. In this case, we should not just use the coverage area as our optimization objective. Instead, we need to merge the edges based on the dynamic traffic density on each edge. There are more issues such as whether the cover range is adjustable, and if they are discrete ranges instead of arbitrary continuous range, etc. Any specific constraints on the configuration will need a customized modification on the merging criteria in the algorithm. We will pursue these in the future based on the real world requirements.



(a) Raw road networks



(b) Clustering Results



(c) Total coverage area versus number of bases

Figure 5.3: Case study on two road networks. Left: network 1 - urban, 331 edges, $R = 600$ m. Right: network 2 - suburban, 690 edges, $R = 6000$ m.

Chapter 6

Hierarchical Clustering: A General Solution to Vehicle Routing Problems

6.1 Introduction

Vehicle routing problems (VRP) is one of the most influential research problems, and serve as the underlying support for many logistics and transportation applications. The VRPs has many variations depending on the specific application scenario. In this section we will overview the history of various vehicle routing problems, and famous solution methods.

6.1.1 Capacitated Vehicle Routing Problem (CVRP)

The classical Travel Salesman Problem (TSP) is the origin and the simplest form of vehicle routing problems: Given the locations of multiple cities, a salesman needs to cover each of them exactly once, with the shortest travel distance. Multiple Travel Salesman Problem (mTSP) is similar, but it allows more than one salesman to finish the task. Bektas offered a thorough overview of mTSP problem and its exact and heuristics solution methods Bektas (2006); Kara and Bektas (2006). In 1959, Dantzig and Ram generalized TSP problem and applied it to Truck Dispatching Problem: one or more trucks are sent out to pick up goods from a bunch of stations, each of which has a certain quantity of goods, and the trucks have limited capacities Dantzig and Ramser (1959). The goal is to find the best matching and route so the total travel distance is minimized. Since then Truck Dispatching Problem has spurred decades of other studies of more complicated and practical configurations and formulations. A more general name "Vehicle Routing Problem (VRP)" has been used. Capacitated Vehicle Routing Problem (CVRP) is among the most frequently studied problems of this class. Clarke and Wright created "Savings" methods in 1964 Clarke and Wright (1964). It starts from generating short routes. "Saving" is defined as the decrease of travel distance when merging two shorter routes. The method keeps merging the route pairs that causes the largest saving, till no merging is feasible (all vehicles are filled up). Miller created the "Sweep" method, in which the customers are paired with vehicles based on their locations in a polar coordinate system, whose center is the vehicle's origin depot Miller (1970). Instead of using polar shape, Foster and Ryan advanced Miller's method to petal like space, and named their method Petal Method Foster and Ryan (1976); Ryan et al. (1993), which was reported to perform more accurate and faster than Sweep Laporte et al. (2000). Christofides and Eilon designed 3-optimal method, which was claimed to perform much faster than Savings Christofides and Eilon (1969). Besides Savings and Sweep, another class of heuristic algorithm for CVRP is two phase method: cluster first to partition the space and then find optimal local routing. The most famous two phase method is Fisher and Jaikumar's Generalized Assignment Algorithm Acharyya et al. (2012), where the space is divided into cones and the nearest customer inside each cone to the vehicles is chosen as a seed to initialize a route. Every passenger chooses the most convenient route to insert, which causes the minimum distance increase and the vehicle is not filled up. Another well-known two phase algorithm is the cyclic transfer algorithm Thompson and Psaraftis (1993).

6.1.2 Vehicle Routing Problem with Pickup and Delivery (VRPPD)

CVRP applies to problems like logistics distributing, delivering goods to stores /customers' houses, etc. For taxi scheduling, ride sharing systems, this is not a proper model because a passenger has both a pick up location and delivery location, while in CVRP, every customer has only one service location. Therefore, there has been another type of vehicle routing model Vehicle Routing Problem with Pickup and Delivery (VRPPD). Since pickup location and delivery location are not related spatially, they are not necessarily next to each other and can be far away, VRPPD has higher complexity than CVRP. The single location based methods overviewed above cannot be applied to VRPPD directly. Katoha and Yano studied the one-vehicle-multiple-passenger tree shaped network routing problem with pick and delivery demands Katoh and Yano (2006). Although their two-approximation method seems to work well on tree shaped network, it unfortunately does not apply to general graph/network, which is what the real transportation network is. Tzoreff et al studied the same problem on other special shaped networks such as cycles, warehouse shapes, etc Tzoreff et al. (2002). Gribkovskaia et al studied another restricted configuration where all delivery loads come from the vehicle depot and all loads picked up will be sent back to the same depot Gribkovskaia et al. (2001). As the author pointed out in the original paper, this assumption does not describe many real applications and definitely does not fit the ride sharing case. Gribkovskaia et al developed a general mixed linear integer programming model for single-vehicle-multiple-customer VRPPD and used the Tabu search heuristics to find the approximated solution Gribkovskaia et al. (2007). Nagy and Salhi formulated the most general multi-vehicle-multi-customer VRPPD model, where pickup and delivery locations, capacity constraints, pickup and delivery orders are included Nagy and Salhi (2005). They also offered a thorough overview and classification of previous models on VRPPD.

6.1.3 Vehicle Routing Problem with Time Window (VRPTW)

CVRP and VRPPD are only focused on geographical locations, however, in practical situations, customers might request service to happen only within a certain time window, or can not be later than some time point. For example, a customer can require a piece of furniture to be deliver between 5 p.m. to 7 p.m. To accommodate time window factor, there is a new type of routing problem called Vehicle Routing Problem with Time Window (VRPTW). Solomon studied VRPTW and came up with a two phase algorithm: First do a nearest neighbor search to attach a customer to its nearest vehicle (although because of the constraint of capacity, a customer might not always gets assigned to the nearest vehicle), then do the one-vehicle-multiple-vehicle routing inside each cluster Solomon (1987). Cordeau et al. formulated VRPTW as a network flow problem, and solved it using different optimization approaches, including branch and cutting, column generation and Lagrangian relaxation Cordeau and Groupe d'études et de recherche en analyse des décisions (Montréal, 2000(@). Braysy and Gendreau overviewed the approximated solution methods for VRPTW, including route construction methods (similar to Solomans two phase method), solution improvement method (slightly and iteratively tune a given route), Tabu search, genetic algorithm, simulated annealing etc. Bräysy and Gendreau (2005,?). Braysy and Gendreau also benchmarked all the algorithms using Solomon's 56 test cases Solomon (2005).

6.1.4 Vehicle Routing Problem with Pickup and Delivery with Time Window (VRPPDTW)

VRPPDTW is the pickup and delivery location enabled version of VRPTW. It is among the most complicated variations of VRPs. Four types of constraints are supported: capacities, time windows, pickup and delivery locations, and order (pick up happens before delivery). Because of the added complexity, the modelling and solution methods become more advanced. The most frequently cited literature on VRPPDTW is Cordeau's mixed linear integer programming formulation of VRPPDTW and his branch and cut solution to it Cordeau (2006). Spoke and Cordeau later came up with an enhanced branch-and-cut-and-pricing solution to further improve the solution Ropke and Cordeau (2009). The formulation of VRPPDTW is a three-index model and even increasing the number of vehicles and passenger just slightly could cause a dramatic increase in the dimension of solution space and so the computational time. Other researchers have tried other solution methods, such as the state-space-time scheme introduced by Yang Yang and Zhou (2014) and Mahmoudi Mahmoudi and Zhou (2016). However, all solution methods for VRPPDTW so far is still computationally challenged. According to the most recent result reported in Mahmoudi and Zhou (2016), to compute a 50-passenger-15-vehicle case, it takes almost two hours.

The focus of this chapter is to present a general heuristic solution to the VRP family. Our method will be much faster than the theoretical solutions (optimization model based), and is easy to implement. Since the VRP is a big family, we only choose VRPPD and VRPPDTW to illustrate our algorithm.

6.1.5 Objectives in VRPs

There are many different types of objectives one can define when formulating a VRP problem. A VRP forumulation also depends on the interest of the specific application. According to Agatz et al. (2012), there are mainly three types of objectives in VRPs: minimizing system-wide vehicle miles, minimizing system-wide travel time, minimizing number of vehicles needed. These three objectives are shared among all VRP variations. The most recent logistic applications have raised higher and higher requirements on time factor, i.e. to meet the customers' time window requests. Examples of this type of services include UberEats, UberEverything, Amazon Fresh, Amazon Now, etc. Not being able to meet the "deadline" will mean that customers will be disappointed, which in turn hurts the reputation of that service and directly affect customer quantity and so profits. To address this issue, the author in his Master thesis Zhang (2016) has formulated the delay minimization problem. So for VRPPD, we will apply our greedy and hierarchical solution to three types of objectives, while in VRPPDTW, we apply that to four types of objectives.

6.2 VRPPDTW Formulation

Cordeau first formulated VRPPDTW in 2006 Cordeau (2006) and the model has been adopted to solve many related problems, such as truck dispatching, patient transformation in hospital networks Hanne et al. (2009); Beaudry et al. (2010), vehicle customer matching in taxi-sharing Ma et al. (2013); Huang et al. (2014), selecting locations for electric vehicle charge stations, theater, military supply bases, etc. Moccia (2004); Burks Jr (2006). A few comprehensive overviews on this class of research can be found in Furuhata et al. (2013); Parragh et al. (2008); Berbeglia et al. (2007). In this study, our model development is largely based on Cordeau's formulation with slight simplification.

We first describe the configuration of VRPPDTW problem. There are m vehicles in-service. We define the vehicle set $V = \{1, 2, ...m\}$. Every vehicle has a origin depot and destination depot. There are n requests(can be packages, passengers, etc.) awaiting pickup and delivery. Every request comes with a pickup location, a destination, and a pickup time window, and a delivery time window. The pickup location set is defined as $P = \{1, 2, ...n\}$, delivery location set $D = \{n+1, n+2, ...2n\}$. For each vehicle k, we index it origin and destination depots as 0 and $2n+1$, respectively. We then define another location set $N = P \cup D \cup \{0, 2n+1\}$. Every pickup and delivery location has a time window, defined as (e_i, l_i), meaning the early and latest service time. Notice that in some applications this can be simplified, such as in some scenarios, we might only need a lasted pickup/delivery time, or we might only care about delivery time, etc. But we here keep both time points for both pickup and delivery, to make the model complete. Table 6.1 presents the symbols we

Table 6.1: Symbols and definitions

Symbol	Type	Source	Definition
m	Integer	Given	Number of vehicles
n	Integer	Given	Number of request
q_i	Integer	Given	Loads at each stop. Positive for picking up, negative for delivery.
e_i	Double	Given	Lower bound for time window at location i.
l_i	Double	Given	Upper bound for time window at location i.
t_{ij}^k	Double	Given	Travel time of vehicle k from location i to location j.
c_{ij}^k	Double	Given	Travel distance between location i to location j.
C^k	Integer	Given	Capacity of vehicle k
x_{ij}^k	Binary	Variable	$x_{ij}^k = 1$ if vehicle k travels from location i to j, $x_{ij}^k = 0$ otherwise.
B_i^k	Double	Variable	Arrival time when vehicle k gets to location i.
Q_i^k	Integer	Variable	Load in vehicle k after it departs location i.
d_i^k	Double	Variable	Stop time of vehicle k at location i.

Objective 1: Minimizing system-wise vehicle miles.

$$\min \quad \sum_{k\in V}\sum_{j\in N}\sum_{i\in N} c_{ij}x_{ij}^k \tag{6.1}$$

Objective 2: Minimizing system-wise travel time.

$$\min \quad \sum_{k\in V}\sum_{j\in N}\sum_{i\in N} t_{ij}^k x_{ij}^k \tag{6.2}$$

Objective 3: Minimizing number of vehicles needed.

$$\min \quad \sum_{k\in V}\sum_{j\in N} x_{0j}^k \tag{6.3}$$

Objective 4: Minimizing service delay (pickup delay and delivery delay).

$$\min \quad \sum_{i\in P}\left(\lambda_1 max\left(0, \left(\sum_{k\in V}\left(B_i^k \sum_{j\in N} x_{i,j}^k\right)\right) - l_i\right) + \lambda_2 max\left(0, \left(\sum_{k\in V}\left(B_{i+n}^k \sum_{j\in N} x_{i+n,j}^k\right)\right) - l_{i+n}\right)\right) \tag{6.4}$$

will use, both given information and variables. Four types of objectives can be found in 6.1, 6.2, 6.3, 6.4. The constraints are from 6.5 to 6.21.

$$x_{ii}^k = 0, \quad \forall i \in N \tag{6.5}$$

$$x_{i+n,i}^k = 0, \quad \forall k \in V, \forall i \in P \tag{6.6}$$

$$x_{0,i+n}^k = 0, \quad \forall k \in V, \forall i \in P \tag{6.7}$$

$$\sum_{k \in V} \sum_{j \in N} x_{ij}^k = 1, \quad \forall i \in P \tag{6.8}$$

$$\sum_{j \in N} x_{ij}^k - \sum_{j \in N} x_{i+n,j}^k = 0, \quad \forall k \in V, \forall i \in P \tag{6.9}$$

$$\sum_{j \in N} x_{0j}^k = 1, \quad \forall k \in V, \forall i \in P \tag{6.10}$$

$$\sum_{j \in N} x_{0j}^k = 1, \quad \forall k \in V \tag{6.11}$$

$$\sum_{i \in P} x_{i+n,2n+1}^k = 1, \quad \forall k \in V \tag{6.12}$$

$$\sum_{j \in N} x_{ji}^k - \sum_{j \in N} x_{ij}^k = 0, \quad \forall k \in V, \forall i \in P \cup D \tag{6.13}$$

$$B_j^k \geq (B_i^k + d_i^k + t_{ij}^k) x_{ij}^k, \quad \forall k \in V, \forall i \in N, \forall j \in N \tag{6.14}$$

$$B_{i+n}^k \geq B_i^k, \quad \forall k \in V, \forall i \in P \tag{6.15}$$

$$Q_i^k \geq 0, \quad \forall k \in V, \forall i \in N \tag{6.16}$$

$$Q_i^k \geq q_i, \quad \forall k \in V, \forall i \in N \tag{6.17}$$

$$Q_i^k \leq C^k, \quad \forall k \in V, \forall i \in N \tag{6.18}$$

$$Q_i^k \leq C^k + q_i, \quad \forall k \in V, \forall i \in N \tag{6.19}$$

$$Q_j^k \geq (Q_i^k + q_i) x_{ij}^k, \quad \forall k \in V, \forall i \in N, \forall j \in N \tag{6.20}$$

$$e_i \leq B_i^k \leq l_i, \quad \forall k \in V, \forall i \in N \tag{6.21}$$

Objectives 6.1 and 6.2 are very similar. To get the total travel distance, we integrate on the product of the individual distance (or travel time) on each edge and the indicator variable x_{ij}^k. For objective 6.2, the total number of vehicles in use, we simply count how many vehicles starts from its origin depot, and if one vehicle k is not in use, $x_{0j}^k = 0, \forall j \in N$. Objective 6.4 is formulated by adding up the pickup and delivery delays of every passenger, since these two parts might not be equally critical, we weigh them by coefficients λ_1 and λ_2 respectively. We only count if the arrival time is later than the requested, so the outside *max* operators filters out those that are on time. $\sum_{k\in K}\left(B_i^k \sum_{j\in N} x_{i,j}^k\right)$ is the actual arrival time at location i. From 6.9 we know that for any $i \in P$, there is only one pair of j and k to make $\sum_{k\in K}\sum_{j\in N} x_{ij}^k = 1$, which is because there is only one vehicle to pick up a passenger at his/her origin and departs to only one direction, so the dual summation will remain as B_i^k where vehicle k is the one that has picked up passenger i.

Objective 6.4 is not a completely linear formulation. The nonlinear terms $B_i^k \sum_{j\in N} x_{i,j}^k$, $B_{i+n}^k \sum_{j\in N} x_{i+n,j}^k$, the *max* operator can all be linearized using the big-M notation. Details can be found in Zhang (2016).

For the constraints, 6.5 to 6.7 set constraints on x_{ij}^k based on service order requirements: a vehicle cannot travel back to itself, a vehicle cannot travels from an order' destination to its origin, and a vehicle cannot travel from its origin depot directly to an order's destination. 6.8 describes that exactly one vehicle picks up an order. 6.9 together with 6.8 describes the an order is picked up and delivered by the same vehicle. 6.10 enforces that a vehicle always starts from its origin depot. 6.11 enforces a vehicles to go back to its destination depot after delivering the last order. 6.13 is the flow conservation at any pickup and delivery location. 6.14 describes the arrival time constraint at two locations: if a vehicle travels from one location to another, then it arrives at the second location later than the first location. 6.15 states that a vehicle always arrives at an order's delivery location later than the pickup location. This is a constraint that Cordeau's formulation did not use. Cordeau used some advanced order constraint techniques to enforce the orders. We found that it is much easier to just add a constraint on B_i^k and B_{i+n}^k. 6.16 to 6.19 restricts the lower and upper bounds of the load of a vehicle. At a pickup location, $q_i \leq Q_i^k \leq C^k$, while at a drop-off location, $0 \leq Q_i^k \leq C^k + q_i$. 6.20 describes the load relationship at two locations, logically same as 6.14. 6.21 enforces the pickup and delivery action to happen within the corresponding time window. The problem behind Objectives 6.1, 6.2 and 6.3 are different from the one behind Objective 6.4. To minimize system-wide travel distance or time, or the number of vehicles to fulfill the deliveries, the underlying assumption is that we are able to find a solution that satisfies all the constraints. For the service delay minimization problem, we have assumed there is no perfect solution to satisfy all time windows, and so for Objective 6.4, we do not include the time window constraint 6.20.

The models can be solved as a mixed integer programming (MIP) problem using the optimization solvers such as Gurobi Optimization et al. (2012), CPLEX CPLEX (2009), etc. As we will show later in Section 6.1 that solving the problems with standard optimization packages is much slower than our heuristic algorithm that we will introduce in the next section.

6.3 Greedy and Hierarchical Clustering Based Heuristics for VRPPDTW

Although standard optimization packages can always find the optimal solutions, the slow speeds prohibits it from being deployed to some real applications where real-time response is more critical than the absolute accuracy. In this section, we present a general Greedy and Hierarchical clustering based method, that can quickly find near-optimal solutions for VRPPDTW of all four types of objectives introduced earlier.

6.3.1 Algorithm Design

The logic is quite straightforward. It is an agglomerative process, where in each iteration one order is assigned to one vehicle. When deciding which vehicle to pick up and deliver that order, we use a greedy strategy, i.e. the combination of that order and that vehicles will cause the minimum increase of our objective functions, which are the costs.

The "best" has double meaning here. On one hand, no matter which vehicle the order is eventually assigned to, considering there might have already been orders assigned to this vehicle, the new order should then placed at the optimal service order inside the vehicle, which in other words, is that inserting this new order will, again, causes the minimum increase to our objective function. Let's see an example, before adding order o_j to vehicle v_k, v_k has already been assigned to serve orders $\{o_1, o_2, ...o_m\}$ following the most efficient (which means leading to minimum objective function value) route: $R = \{+o_2 + o_1, -o_1, +o_3, -o_2, ...\}$. (Note: here we use $+o_2$ to represent picking up order o_2 at its pickup location and $-o_2$ delivering order o_2 at the dropoff location.) Adding o_j to R is a process of inserting $+o_j$ and $-o_j$ to existing route R, with the constraint that $+o_j$ must come earlier than $-o_j$ because delivery happens physically after pickup. The goal is to find out which new route carries the least cost. This can be done by enumerating all possible new routes after inserting $+o_j$ and $-o_j$, and comparing their new costs. This process can be implemented as a dual loop, where the outside loop is on the possible insertion spot s for delivery $-o_j$, so $s = 0, 1, 2, ...m + 1$. The inside loop is on the insertion spot t for pickup $+o_j$, and because $+o_j$ has to be in front of $-o_j$, so $t = 0, 1, 2, ...s$. The complexity of this step is $O(q^2)$, where q is the capacity of the vehicle. Algorithm 12 details this in-vehicle sorting process.

On the other hand, assigning an order to different vehicles will produce different cost increases, so we assign it to the vehicle that will cause the minimum overall cost.

Since every assignment contributes a minimal possible cost increase, the total cost should also be minimal. However, just like any other greedy methods that gets easily trapped in a local optimum Tabatabaei et al. (2012), the greedy agglomeration step above is also "short sighted" that is insufficient to achieve global optimum. Although every order attaches to the best available vehicle and follows the best service route to achieve minimum objective value, the result is not necessarily a global minimum. This is because a later assigned order does not have as many vehicles to choose from as an earlier assigned order. A later assigned order might be assigned to a vehicle that causes a large although "best-at-that-point" cost, just because by the time that order gets to choose, that is the only vehicle available. In other words, if an earlier assigned order does not attach to the best vehicle at that moment, but instead chooses

Algorithm 10: Monte-Carlo

```
Monte−Carlo (oList, vList, M)
    Input  : order list oList, vehicle list vList, number of simulations M
    Output: Best routes {R}. Optimal cost min_cost
    min_cost ← Infinity
    for i ← 1 to M do
        oList ← Shuffle(oList)
        cost, {R'} ← Match(oList, vList)
        if cost < min_cost then
            min_cost ← cost
            {R} ← {R'}
        end
    end
    return min_cost, {R}
```

a sub-optimal one, and leaves the "best" vehicle for a later assigned order, which will find it more convenient to use, the overall cost might be smaller than the other way around.

Our strategy is to use Monte Carlo simulation to address the"short sight" issue. Since the order of the order list matters, we can shuffle the order list and repeat the greedy clustering steps using the shuffled order list. Shuffling the order list is essentially to get a new random permutation. The randomness of the orders and sufficient number of simulations aim to give every order an equal chance to be combined with the best vehicle that can achieve minimal system cost.

Up to now, we have introduced the development idea behind our greedy clustering and Monte Carlo simulation based algorithm for VRPPDTW. Algorithm 10, 11 and 12 jointly present the implementation.

6.3.2 Cost Function Implementation

Notice that the algorithm presented above is good for all four types of objectives. The difference of four objectives is in the implementations of **get_cost()** subroutine, i.e. calculating the cost/objective function value given a route. For example, a route $R = \{+o_2 + o_1, -o_1, +o_3, -o_2 ...\}$. If this is a vehicle mileage minimization problem

Algorithm 11: Match

```
Match (oList, vList, M)
    Input  : order list oList, vehicle list vList
    Output: Best routes {R'}. Optimal cost min_cost
    min_cost ← 0
    foreach unassigned order o ∈ oList do
        min_inc_cost ← Infinity
        foreach vehicle v ∈ vList do
            if v is not full then
                inc_cost, tmp_R ← Best_Insertion(v.R, o)
                if inc_cost < min_inc_cost then
                    min_inc_cost ← inc_cost
                    R' ← tmp_R
                end
            end
        end
        Mark o as assigned
        {R'}.append(R')
        min_cost ← min_cost + min_inc_cost
    end
    return min_cost, {R'}
```

6.1, it is easy to determine the mileage at each location on the mile. The cost is then the mileage at the end of the route. Same thing for travel time minimization problem 6.2. For minimizing number of vehicles in use 6.3, the cost is either 0, or 1. If this is the first time use the vehicle, then it is 1, if this vehicle has been used for other orders, then it is 0. It is a little tricky to calculate the cost for delay minimization problem 6.4. Suppose there are q served in the route, then we are able to determine the time stamp at each location on the route, $T = \{t_1, t_2, ..., t_{2q}\}$. There are two ways to calculate the delay: (1) Count only the late deliveries. No matter how early all other orders are delivered, we count only the lately delivered ones. $D = \sum_{i=1}^{q} (\lambda_1(t_i - l_i)I_i + \lambda_2(t_{i+q} - l_{i+q})I_{i+q})$, where $I_i = \begin{cases} 0, & if\ t_i \leq l_i \\ 1, & if\ t_i > l_i \end{cases}$. (2) Enable margins, count late deliveries as positive delays and early deliveries as negative delays, $D = \sum_{i=1}^{q} \lambda_1(t_i - l_i) + \lambda_2(t_{i+q} - l_{i+q})$. Our experiments in Zhang (2016) showed that

Algorithm 12: Best-Insertion

Best−Insertion (R, o_j)

Input : Existing Route R, new order o_j

Output: Best new route R^+ after inserting $+o_j$ and $-o_j$. Increased cost *inc_cost*

$inc\_cost \leftarrow 0$

for $i \leftarrow$ 0 *to* ***length***(R) **do**

  for $j \leftarrow$ 0 *to* i **do**

    $R' = [R(1:j)\ +o_j\ R(j+1:i)\ -o_j\ R(i+1:end)]$

    if o_j *meets time window constraints 6.21* ***or*** *this is for Objective 6.4* **then**

      $delta\_cost = \textbf{get\_cost}(R') - \textbf{get\_cost}(R)$

      if $delta\_cost < inc\_cost$ **then**

        $inc\_cost = delta\_cost$

        $R^+ = R'$

      end

    end

  end

end

return R^+, inc_cost

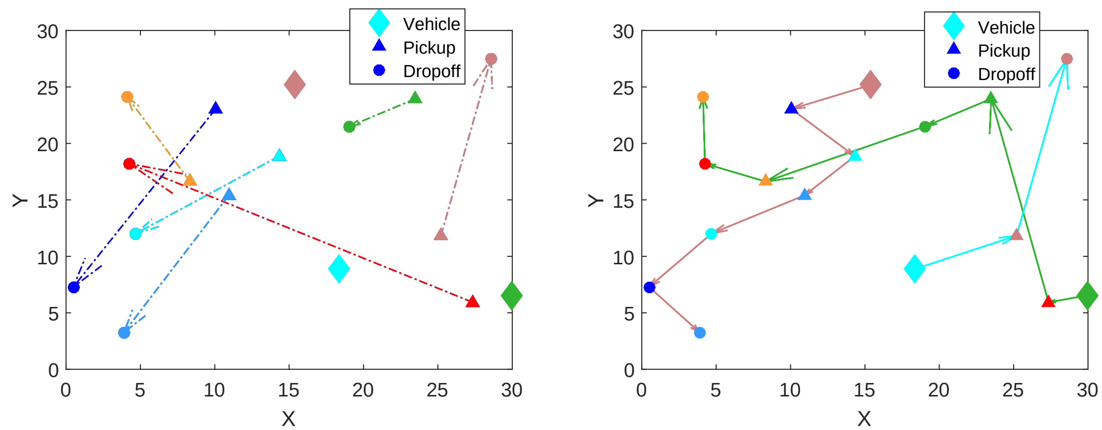
(2) is the correct way to calculate delay cost, in the algorithm itself, to avoid filling vehicles one by one.

6.4 Case Study

We give two groups of case studies. Since Objectives 6.1 and 6.2 takes the same form, we will only implement our algorithm on 6.1. We are also interested in delay minimization, so we have a second group of case study on 6.4.

6.4.1 Case Study on Minimizing Travel Distance

We compared our algorithm with optimization package Gurobi on a few small cases: o5v3(5 orders, 3 vehicles), o6v3, o7v3, o10v4, o15v6. The results show that our algorithms got exactly the same solutions as given by Gurobi. The difference is that



(a) Configuration **(b)** Best solution

Figure 6.1: Travel distance minimization on a o7v3 case.

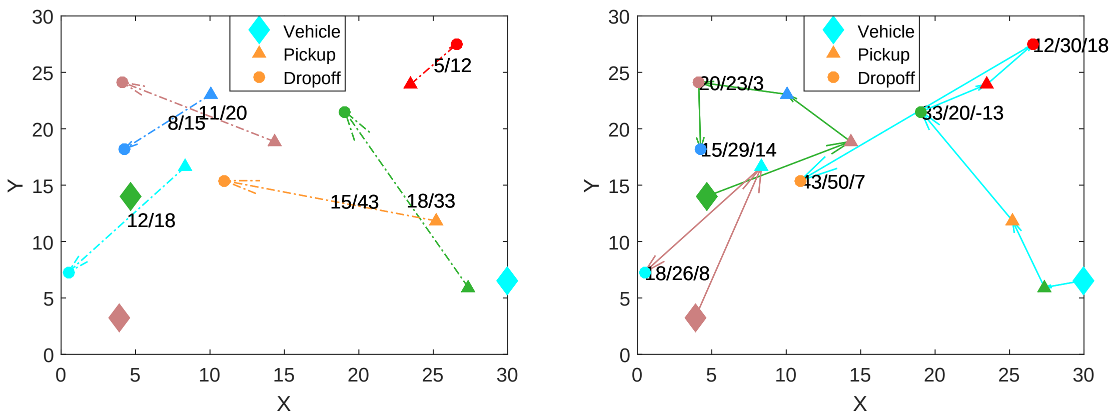
Table 6.2: Run-time Comparison for travel distance minimization

Case	Run-time (s)	
	Gurobi	Clustering Algorithm
o5v3	0.72	0.71
o6v3	1.33	1.27
o7v3	15.71	1.15
o10v4	117.52	2.36
o15v6	3750.00	5.19

our algorithm is much faster than Gurobi. Figure 6.1 visualizes a solution on a o7v3 case. Table 6.2 compares the run-time of the two solution methods.

6.4.2 Case Study on Minimizing Service Delay

We conduct the same run-time comparison as in Section 6.4.1, as shown in Table 6.3. Figure 6.2 is a visualization of a o6v3 case. Besides, we conduct performance evaluation on much larger cases, as shown in Table 6.4 and Figure 6.3. Note that Gurobi is not able to solve these larger cases any more. Using 32 processors, we able to obtain pretty good solutions within a reasonable run-time through sufficient number of Monte Carlo simulations. In Figure 6.3, as we invest more and more Monte



(a) Configuration

(b) Best solution

Figure 6.2: Service delay minimization on a o6v3 case.

Table 6.3: Run-time Comparison for service delay minimization

Case	Run-time (s)	
	Gurobi	Clustering Algorithm
o4v2	1.93	1.14
o5v3	15.19	1.63
o6v3	490.75	2.05
o6v4	100.24	2.67
o7v3	2910.36	3.06
o7v4	3779.94	3.30

Carlo simulations, the recorded best objective value keeps dropping, till it stays the same. In Zhang (2016), we have looked into how many simulations will be sufficient to guarantee a near-optimal solution, and found out that it is about linearly related to the product of number of vehicles and number of orders.

6.5 Conclusion and Future Work

We have presented a hierarchical clustering strategy to solve different types of vehicle routing problems. The basic idea of this algorithm is to let every “order” to select the “best” vehicle, such that their combination, not only satisfies all the constraints,

Table 6.4: Run-time of service delay minimization on larger cases

Case	Run-time(s) (1000 simulations on 32 processors)
o50v20	3.65
o100v50	17.38
o200v100	99.22
o300v150	219.73

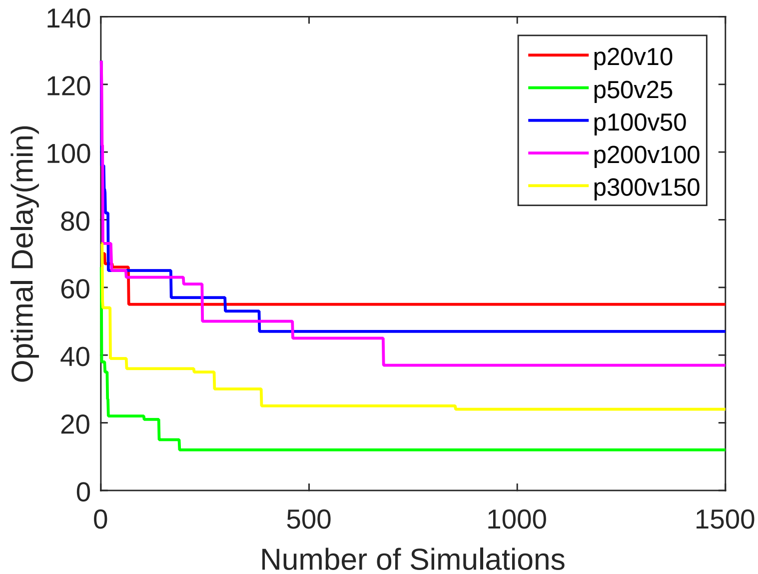


Figure 6.3: Relationship between optimal objective value and the number of simulations.

but also causes as little increase to our objective value as possible. Shuffling the order list can overcome the "short-sight" issue. We implement on two (or three) types of objective functions: minimizing system-wise travel distance/travel time, and minimizing service delay. We have conducted sufficient case studies on both small scale system, where we have been able to compare the algorithm with Gurobi's optimal solution, and also larger scale cases. We see that on small cases, the results obtained from the clustering algorithm are exactly the same as those given by Gurobi, while the computational cost has been reduced by over three orders of magnitudes. While on large cases, even though we are not able to compare with Gurobi's results, the convergence of the results makes us strongly believe the algorithm gives near-optimal solution, still within reasonable computational time. Note that although all our experiments have been on VRPPDTW formulation, the algorithm can easily be adapted to other formulations of VRP, since those formulations have less constraints and makes it earlier to implement the clustering algorithm.

Our algorithm is a contribution to both the academic research in the VRP domain, and also engineering solutions that can be implemented in real-time, large scale practical applications. Delivery services such as UberEats, UberEverything, Amazon Now, Amazon Fresh might also find our study related.

Chapter 7

Conclusions

We have witnessed rapid advances in the domain of Intelligent Transportation System over the past few years. Massive mobile data set become available and are spurring new research that applies statistical inference and machine learning to extract supportive traffic information and better predict its dynamics. Connected and driverless cars tend to be the future and combines different disciples, including computer science and transportation in an unprecedentedly close way. New business operations, such as vehicle hailing, short term delivery services, are good examples how new technologies can re-define our traffic and logistics system. Clustering algorithms, a unique method that reveals the association among data, have not been fully exploited in the new transportation context. We have argued that: 1) Clustering algorithms were designed for “points”, two dimensional, three dimensional, or higher dimensional, where features are treated homogeneously. 2) Data in transportation field are far more complicated than being a “point”. Data might have underlying constraints, such as accidents happens on road network, vehicle trajectory are mutually constricted points. Some data are geometric shapes such as the road network itself. Some data are essentially heterogeneous , such as locations with time order constraints. We push the frontier of applying clustering algorithms in transportation domain by re-examining the mathematical formulation and philosophy of the classical clustering algorithms, and reformulating them to incorporate transportation-specific information. Doing so, we are able to apply these algorithms to solve more types of problems besides clustering geo-locations.

Concretely, in Chapter 3, we re-designed DBSCAN algorithm by giving it routing ability. This expands it from a n-dimensional space to a network space, which is the space of traffic events. Our case studies validated its capability of providing more accurate clustering results for accident hotspot detection, among many other hotspot discovery applications.

In Chapter 4, we explored the clustering algorithms' potential of data compression. Our online regression based algorithm were able to selectively collect GPS location and achieve over 100 times of compression power while maintaining an acceptable accuracy.

In Chapter 5, we clustered over road network itself. Road network itself is a type of data, and in the context of allocating resources over an arbitrary network, hierarchical clustering offers a fast and effective heuristics to find the locations.

In Chapter 6, we presented a type of hierarchical clustering strategy that handles mutually constrained location and time information in a vehicle routing problem. This strategy excels itself over traditional optimization model plus solver approach, in terms of its much lower computational cost and fine accuracy. It is also very easy to implement and supports different types of vehicle routing problems.

The studies included in this dissertation focus more on examining the new possibility of solving transportation problems using reformulated clustering algorithms, rather than the specific problems themselves. The goal is to create these new methods, and implement them in an efficient and portable way, such that people find it easy to use to solve more in-depth problems. That being said, there will be many problems/applications that can find our methods helpful. For example, for the Dijk-DBSCAN algorithm, although we solely conducted a accident hotspot detection case study, one might use it to detect congestion hotspot, hazard roads for pedestrians and bicyclist, etc. The hierarchical clustering algorithms over road network, might be extended to allocating e-Bike or electric vehicle return or charge station, by changing the model itself but using the same clustering strategy. Similarly, the hierarchical clustering algorithm in the VRP problem, could possibly have unlimited extensions, considering in different applications, the constraints could be largely different, but can all find their solutions in our vehicle-order matching framework.

We believe our work will help other people solve their own problems.

Bibliography

2016, B. C. C. (2016). Bristol city council 2016. 25

Acharyya, R., Das, G. K., et al. (2012). Unit disk cover problem. *arXiv preprint arXiv:1209.2951*. 51, 68

Aerts, K., Lathuy, C., Steenberghen, T., and Thomas, I. (2006). Spatial clustering of traffic accidents using distances along the network. In *Proc. 19th Workshop of the International Cooperation on Theories and Concepts in Traffic Safety*. 3

Agatz, N., Erera, A., Savelsbergh, M., and Wang, X. (2012). Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303. 71

Ahmed-Zaid, F., Bai, F., Bai, S., Basnayake, C., Bellur, B., Brovold, S., Brown, G., Caminiti, L., Cunningham, D., Elzein, H., et al. (2011). Vehicle safety communications–applications (vsc-a) final report: Appendix volume 1 system design and objective test. Technical report. 26, 42, 48

Anbaroglu, B., Heydecker, B., and Cheng, T. (2014). Spatio-temporal clustering for non-recurrent traffic congestion detection on urban road networks. *Transportation Research Part C: Emerging Technologies*, 48:47–65. 3, 10

Anderson, T. K. (2009). Kernel density estimation and k-means clustering to profile road accident hotspots. *Accident Analysis & Prevention*, 41(3):359–364. 3, 10

Bartin, B., Ozbay, K., and Iyigun, C. (2007). Clustering-based methodology for determining optimal roadway configuration of detectors for travel time estimation.

Transportation Research Record: Journal of the Transportation Research Board, (2000):98–105. 10

Basnayake, C., Lachapelle, G., and Bancroft, J. (2011). Relative positioning for vehicle-to-vehicle communications-enabled vehicle safety applications. In *Proceedings of the 18th ITS World Congress, Orlando, Florida*, pages 1–16. 40, 42

Beaudry, A., Laporte, G., Melo, T., and Nickel, S. (2010). Dynamic transportation of patients in hospitals. *OR spectrum*, 32(1):77–107. 72

Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219. 68

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., and Laporte, G. (2007). Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31. 72

Bills, S. D. (2009). Examining hotspots of traffic collisions in redlands, california. 9

Birant, D. and Kut, A. (2007). St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221. 10

Brand, M. (2006). Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30. 35

Bräysy, O. and Gendreau, M. (2005). Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118. 70

Brecheisen, S., Kriegel, H.-P., and Pfeifle, M. (2006). Parallel density-based clustering of complex objects. *Advances in Knowledge Discovery and Data Mining*, pages 179–188. 20

Brent, R. P., Luk, F. T., and Van Loan, C. (1982). Computation of the singular value decomposition using mesh-connected processors. Technical report, Cornell University. 35

Brinkmann, A. (2017). Co-scheduling: Prospects and challenges. *Co-Scheduling of HPC Applications*, 28:1. 20

Bulllock, S. S. (2005). Qr factorizations using a restricted set of rotations. *Electronic Transactions on Numerical Analysis*, 21:20–27. 33

Burks Jr, R. E. (2006). An adaptive tabu search heuristic for the location routing pickup and delivery problem with time windows with a theater distribution application. Technical report, DTIC Document. 72

Calabrese, F., Di Lorenzo, G., Liu, L., and Ratti, C. (2011). Estimating origin-destination flows using opportunistically collected mobile phone location data from one million users in boston metropolitan area. *IEEE Pervasive Computing*, 99. 2

Cardei, M. and Du, D.-Z. (2005). Improving wireless sensor network lifetime through power aware organization. *Wireless Networks*, 11(3):333–340. 50

Cardei, M., Thai, M. T., Li, Y., and Wu, W. (2005a). Energy-efficient target coverage in wireless sensor networks. In *INFOCOM 2005. 24th annual joint conference of the ieee computer and communications societies. proceedings ieee*, volume 3, pages 1976–1984. IEEE. 51

Cardei, M. and Wu, J. (2006). Energy-efficient coverage problems in wireless ad-hoc sensor networks. *Computer communications*, 29(4):413–420. 49

Cardei, M., Wu, J., Lu, M., and Pervaiz, M. O. (2005b). Maximum network lifetime in wireless sensor networks with adjustable sensing ranges. In *Wireless and Mobile Computing, Networking and Communications, 2005.(WiMob'2005), IEEE International Conference on*, volume 3, pages 438–445. IEEE. 50, 51

Chawla, S., Zheng, Y., and Hu, J. (2012). Inferring the root cause in road traffic anomalies. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 141–150. IEEE. 25

Cheng, M. X., Ruan, L., and Wu, W. (2005). Achieving minimum coverage breach under bandwidth constraints in wireless sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2638–2645. IEEE. 51

Cheng, M. X., Ruan, L., and Wu, W. (2007). Coverage breach problems in bandwidth-constrained sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 3(2):12. 51

Christofides, N. and Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, 20(3):309–318. 68

Cici, B., Markopoulou, A., Frias-Martinez, E., and Laoutaris, N. (2014). Assessing the potential of ride-sharing using mobile and social data: a tale of four cities. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 201–211. ACM. 2

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581. 68

Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586. 71, 72

Cordeau, J.-F. and Groupe d'études et de recherche en analyse des décisions (Montréal, Q. (2000). *The VRP with time windows*. Montréal: Groupe d'études et de recherche en analyse des décisions. 70

CPLEX, I. I. (2009). V12. 1: Users manual for cplex. *International Business Machines Corporation*, 46(53):157. 76

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management science*, 6(1):80–91. 68

Deying, L. and Liu, H. (2009). Wireless networks: Research, technology and applications, chapter sensor coverage in wireless sensor networks. 49

Di Martino, F., Loia, V., and Sessa, S. (2008). Extended fuzzy c-means clustering algorithm for hotspot events in spatial analysis. *International Journal of Hybrid Intelligent Systems*, 5(1):31–44. 10

DiTech (2016). Solve ditech's demand-supply prediction challenge. 26

Divya, G., Robinson, R. R., and Selvan, K. (2014). Suitability of clustering algorithms for crime hotspot analysis. *International Journal of Science, Engineering and Computer Technology*, 4(7):231. 10, 11

Downs, A. (2005). *Still stuck in traffic: coping with peak-hour traffic congestion.* Brookings Institution Press. 3

Egecioglu, Ö. and Srinivasan, A. (1995). Givens and householder reductions for linear least squares on a cluster of workstations. 31

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231. 12

Faria, A. M., Cintra, M. E., de Castro, A. F., and Lopes, D. C. (2014). Criminal hot spot detection using formal concept analysis and clustering algorithms. *Encontro Nacional de Inteligência Artificial, ENIAC*, pages 85–90. 10

Foster, B. A. and Ryan, D. M. (1976). An integer programming approach to the vehicle scheduling problem. *Journal of the Operational Research Society*, 27(2):367–384. 68

Fu, B., Chen, Z., and Abdelguerfi, M. (2007). An almost linear time 2.8334-approximation algorithm for the disc covering problem. In *International Conference on Algorithmic Applications in Management*, pages 317–326. Springer. 51

Furuhata, M., Dessouky, M., Ordóñez, F., Brunet, M.-E., Wang, X., and Koenig, S. (2013). Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological*, 57:28–46. 72

Giannotti, F., Nanni, M., Pedreschi, D., Pinelli, F., Renso, C., Rinzivillo, S., and Trasarti, R. (2011). Unveiling the complexity of human mobility by querying and mining massive trajectory data. *The VLDB JournalThe International Journal on Very Large Data Bases*, 20(5):695–719. 24, 25

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48. 37

Golub, G. H. and Van Loan, C. F. (2012). *Matrix computations*, volume 3. JHU Press. 31

Gribkovskaia, I., Halskau, Ø., Laporte, G., and Vlček, M. (2007). General solutions to the single vehicle routing problem with pickups and deliveries. *European Journal of Operational Research*, 180(2):568–584. 69

Gribkovskaia, I., Halskau, O., and Myklebost, K. N. B. (2001). Models for pick-up and deliveries from depots with lasso solutions. *NOFOMA2001, Collaboration in logistics: connecting islands using information technology*, pages 279–293. 69

Hanne, T., Melo, T., and Nickel, S. (2009). Bringing robustness to patient flow management through optimized patient transports in hospitals. *Interfaces*, 39(3):241–255. 72

Har-Peled, S. and Lee, M. (2012). Weighted geometric set cover problems revisited. *Journal of Computational Geometry*, 3(1):65–85. 52

Herrera, J. C., Work, D. B., Herring, R., Ban, X. J., Jacobson, Q., and Bayen, A. M. (2010). Evaluation of traffic data obtained via gps-enabled mobile phones: The mobile century field experiment. *Transportation Research Part C: Emerging Technologies*, 18(4):568–583. 2

Herring, R., Hofleitner, A., Abbeel, P., and Bayen, A. (2010). Estimating arterial traffic conditions using sparse probe data. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 929–936. IEEE. 2

Holmes, M., Gray, A., and Isbell, C. (2007). Fast svd for large-scale matrices. In *Workshop on Efficient Machine Learning at NIPS*, volume 58, pages 249–252. 35

Huang, Y., Bastani, F., Jin, R., and Wang, X. S. (2014). Large scale real-time ridesharing with service guarantee on road networks. *Proceedings of the VLDB Endowment*, 7(14):2017–2028. 72

Hunter, T., Herring, R., Abbeel, P., and Bayen, A. (2009). Path and travel time inference from gps probe vehicle data. *NIPS Analyzing Networks and Learning with Graphs*, 12(1). 2

IconFinder (2016). Data, database, storage, tape icon. 37

Icons-Land (2016). Transport. 37

Kara, I. and Bektas, T. (2006). Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of Operational Research*, 174(3):1449–1458. 68

Kasbekar, G. S., Bejerano, Y., and Sarkar, S. (2011). Lifetime and coverage guarantees through distributed coordinate-free sensor activation. *IEEE/ACM transactions on networking*, 19(2):470–483. 51

Katoh, N. and Yano, T. (2006). An approximation algorithm for the pickup and delivery vehicle routing problem on trees. *Discrete Applied Mathematics*, 154(16):2335–2349. 69

Kenney, J. B. (2011). Dedicated short-range communications (dsrc) standards in the united states. *Proceedings of the IEEE*, 99(7):1162–1182. 48

Keogh, E., Chu, S., Hart, D., and Pazzani, M. (2004). Segmenting time series: A survey and novel approach. *Data mining in time series databases*, 57:1–22. 35

Kidane, P. and Bonds, A. (2015). An analysis of hierarchical clustering algorithms for hotspot detection in geographical request maps. 10

Kim, J. and Mahmassani, H. S. (2015). Spatial and temporal characterization of travel patterns in a traffic network using vehicle trajectories. *Transportation Research Procedia*, 9:164–184. 10

Kim, K. and Yamashita, E. Y. (2007). Using ak-means clustering algorithm to examine patterns of pedestrian involved crashes in honolulu, hawaii. *Journal of advanced transportation*, 41(1):69–89. 3

Kortge, J. and Zhang, J. (2005). System and method for providing safety-optimized navigation route planning. US Patent App. 11/117,794. 2

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. 48

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2016). Building machines that learn and think like people. *arXiv preprint arXiv:1604.00289*. 47

Lam, S. H. M. and Toan, T. D. (2008). Short-term travel time prediction using support vector regression. In *Transportation Research Board 87th Annual Meeting*, number 08-0670. 5

Laporte, G., Gendreau, M., Potvin, J.-Y., and Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research*, 7(4-5):285–300. 68

Lawson, A. B. (2010). Hotspot detection and clustering: ways and means. *Environmental and Ecological Statistics*, 17(2):231–245. 10

Leon, S. J., Björck, Å., and Gander, W. (2013). Gram-schmidt orthogonalization: 100 years and more. *Numerical Linear Algebra with Applications*, 20(3):492–532. 31

Li, R. and Rose, G. (2011). Incorporating uncertainty into short-term travel time predictions. *Transportation Research Part C: Emerging Technologies*, 19(6):1006–1018. 5

Li, X., Han, J., Lee, J.-G., and Gonzalez, H. (2007). Traffic density-based discovery of hot routes in road networks. In *International Symposium on Spatial and Temporal Databases*, pages 441–459. Springer. 10

Lopez, X. M., Debeir, O., Maris, C., Rorive, S., Roland, I., Saerens, M., Salmon, I., and Decaestecker, C. (2012). Clustering methods applied in the detection of ki67 hot-spots in whole tumor slide images: An efficient way to characterize heterogeneous tissue-based biomarkers. *Cytometry Part A*, 81(9):765–775. 10

Lou, Y., Zhang, C., Zheng, Y., Xie, X., Wang, W., and Huang, Y. (2009). Map-matching for low-sampling-rate gps trajectories. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 352–361. ACM. 26

Ma, S., Zheng, Y., and Wolfson, O. (2013). T-share: A large-scale dynamic taxi ridesharing service. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 410–421. IEEE. 72

Mahfoudh, S. and Minet, P. (2008). Survey of energy efficient strategies in wireless ad hoc and sensor networks. In *Networking, 2008. ICN 2008. Seventh International Conference on*, pages 1–7. IEEE. 49

Mahmoudi, M. and Zhou, X. (2016). Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on state–space–time network representations. *Transportation Research Part B: Methodological*, 89:19–42. 71

Map, O. S. (2016). Map, o.s. open street map public traces. 25

Mecke, S. and Wagner, D. (2004). Solving geometric covering problems by data reduction. In *European Symposium on Algorithms*, pages 760–771. Springer. 52

Miller, L. R. (1970). Heuristic algorithms for the generalized vehicle dispatch problem. 68

Moccia, L. (2004). *New optimization models and algorithms for the management of maritime container terminals*. PhD thesis, PhD thesis, Universita degli studi della Calabria. 72

Mokotoff, E., Jimeno, J. L., and Gutiérrez, A. I. (2001). List scheduling algorithms to minimize the makespan on identical parallel machines. *Top*, 9(2):243–269. 20

Montazeri-Gh, M. and Fotouhi, A. (2011). Traffic condition recognition using the k-means clustering method. *Scientia Iranica*, 18(4):930–937. 10

Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., and Damas, L. (2013). Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402. 2

Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., and Damas, L. (2016). Time-evolving od matrix estimation using high-speed gps data streams. *Expert Systems With Applications*, 44:275–288. 24, 25

Mulligan, R. and Ammari, H. M. (2010). Coverage in wireless sensor networks: A survey. *Network Protocols and Algorithms*, 2(2):27–53. 49, 50

Nagy, G. and Salhi, S. (2005). Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European journal of operational research*, 162(1):126–141. 69

Newson, P. and Krumm, J. (2009). Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 336–343. ACM. 26, 38

Optimization, G. et al. (2012). Gurobi optimizer reference manual. *URL: http://www. gurobi. com*, 2:1–3. 76

Ossen, S. and Hoogendoorn, S. (2008). Validity of trajectory-based calibration approach of car-following models in presence of measurement errors. *Transportation Research Record: Journal of the Transportation Research Board*, (2088):117–125. 26

Pan, B., Zheng, Y., Wilkie, D., and Shahabi, C. (2013). Crowd sensing of traffic anomalies based on human mobility and social media. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 344–353. ACM. 25

Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2008). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1):21–51. 72

Patwary, M. A., Palsetia, D., Agrawal, A., Liao, W.-k., Manne, F., and Choudhary, A. (2012). A new scalable parallel dbscan algorithm using the disjoint-set data structure. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 62. IEEE Computer Society Press. 20

Phillips, P. and Lee, I. (2006). Mining positive associations of urban criminal activities using hierarchical crime hot spots. In *Intelligence and Security Informatics*, pages 127–132. Springer. 10

Prasannakumar, V., Vijith, H., Charutha, R., and Geetha, N. (2011). Spatio-temporal clustering of road accidents: Gis based analysis and assessment. *Procedia-Social and Behavioral Sciences*, 21:317–325. 3, 9

Punzo, V., Ciuffo, B., and Montanino, M. (2012). Can results of car-following model calibration based on trajectory data be trusted? *Transportation Research Record: Journal of the Transportation Research Board*, (2315):11–24. 26

Roh, G.-P. and Hwang, S.-w. (2010). Nncluster: An efficient clustering algorithm for road network trajectories. In *International Conference on Database Systems for Advanced Applications*, pages 47–61. Springer. 10

Ropke, S. and Cordeau, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286. 71

Ryan, D. M., Hjorring, C., and Glover, F. (1993). Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society*, 44(3):289–296. 68

Schlaich, J., Otterstätter, T., and Friedrich, M. (2010). Generating trajectories from mobile phone data. In *Proceedings of the 89th annual meeting compendium of papers, transportation research board of the national academies*. 1

Shah, S., Bao, F., Lu, C.-T., and Chen, I.-R. (2011). Crowdsafe: crowd sourcing of crime incidents and safe routing on mobile devices. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 521–524. ACM. 2

Shahsavari, B. and Abbeel, P. (2015). Short-term traffic forecasting: Modeling and learning spatio-temporal relations in transportation networks using graph neural networks. 5

Shirai, M., Hirota, M., Ishikawa, H., and Yokoyama, S. (2013). A method of area of interest and shooting spot detection using geo-tagged photographs. In *COMP@ SIGSPATIAL*, pages 34–41. 10

Shladover, S. E. and Tan, S.-K. (2006). Analysis of vehicle positioning accuracy requirements for communication-based cooperative collision warning. *Journal of Intelligent Transportation Systems*, 10(3):131–140. 26

Sohn, K. and Kim, D. (2008). Dynamic origin–destination flow estimation using cellular communication system. *IEEE Transactions on Vehicular Technology*, 57(5):2703–2713. 25

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265. 70

Solomon, M. M. (2005). Vrptw benchmark problems. 70

Sun, M.-T., Yi, C.-W., Yang, C.-K., and Lai, T.-H. (2008). An optimal algorithm for the minimum disc cover problem. *Algorithmica*, 50(1):58–71. 52

Sun, Z. and Ban, X. J. (2013). Vehicle trajectory reconstruction for signalized intersections using mobile traffic sensors. *Transportation Research Part C: Emerging Technologies*, 36:268–283. 1

Systems, E. E. I. (2016). Rtms by eis. 1

Tabatabaei, S. S., Coates, M., and Rabbat, M. (2012). Ganc: Greedy agglomerative normalized cut for graph clustering. *Pattern Recognition*, 45(2):831–843. 77

the NYC Taxi and Commission, L. (2016). Nyc taxi trip data. 6

Thompson, P. M. and Psaraftis, H. N. (1993). Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations research*, 41(5):935–946. 68

Truong, L. T. and Somenahalli, S. V. (2011). Using gis to identify pedestrian-vehicle crash hot spots and unsafe bus stops. *Journal of Public Transportation*, 14(1):6. 5, 9

Tzoreff, T. E., Granot, D., Granot, F., and Sošić, G. (2002). The vehicle routing problem with pickups and deliveries on some special graphs. *Discrete Applied Mathematics*, 116(3):193–229. 69

Wang, B. (2011). Coverage problems in sensor networks: A survey. *ACM Computing Surveys (CSUR)*, 43(4):32. 49

Wen, H., Sun, J., and Zhang, X. (2014). Study on traffic congestion patterns of large city in china taking beijing as an example. *Procedia-Social and Behavioral Sciences*, 138:482–491. 10

Wikimedia (2016). Cloud computing icon.svg. 37

Wikipedia (2014). Covering problems. 52, 56

Won, J.-I., Kim, S.-W., Baek, J.-H., and Lee, J. (2009). Trajectory clustering in road network environment. In *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*, pages 299–305. IEEE. 10

Yang, C., Duraiswami, R., Gumerov, N. A., Davis, L. S., et al. (2003). Improved fast gauss transform and efficient kernel density estimation. In *ICCV*, volume 1, page 464. 10

Yang, L. and Zhou, X. (2014). Constraint reformulation and a lagrangian relaxation-based solution algorithm for a least expected time path problem. *Transportation Research Part B: Methodological*, 59:22–44. 71

Yang, S., Dai, F., Cardei, M., Wu, J., and Patterson, F. (2006). On connected multiple point coverage in wireless sensor networks. *International Journal of Wireless Information Networks*, 13(4):289–301. 51

Yuan, J., Zheng, Y., Xie, X., and Sun, G. (2011). Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM. 25

Yuan, J., Zheng, Y., Xie, X., and Sun, G. (2013). T-drive: Enhancing driving directions with taxi drivers' intelligence. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):220–232. 25

Zhang, B., Xing, K., Cheng, X., Huang, L., and Bie, R. (2012). Traffic clustering and online traffic prediction in vehicle networks: A social influence perspective. In *Infocom, 2012 Proceedings IEEE*, pages 495–503. IEEE. 10

Zhang, X. and Rice, J. A. (2003). Short-term travel time prediction. *Transportation Research Part C: Emerging Technologies*, 11(3):187–210. 5

Zhang, Y. (2016). Scheduling for timely passenger delivery in a large scale ride sharing system. unpublished thesis. 72, 75, 79, 82

Zhao, Q. and Gurusamy, M. (2008). Lifetime maximization for connected target coverage in wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 16(6):1378–1391. 51

Zorbas, D. and Douligeris, C. (2011). Connected coverage in wsns based on critical targets. *Computer Networks*, 55(6):1412–1425. 48

Zorbas, D., Glynos, D., Kotzanikolaou, P., and Douligeris, C. (2010). Solving coverage problems in wireless sensor networks using cover sets. *Ad Hoc Networks*, 8(4):400–415. 51

Appendix

Appendix A

Proof for Lemma 4.0.1

A.1

We only need to prove that

$$R^{(m)} = G^{(m,n)} \cdots G^{(m,2)} G^{(m,1)} \cdots G^{(n+1,n)} \cdots G^{(n+1,2)} G^{(n+1,1)} \cdots G^{(n,n-1)} \cdots G^{(n,2)} G^{(n,1)} \cdots G^{(3,2)} G^{(3,1)} G^{(2,1)} A$$

is upper triangular, i.e. $r_{ij} = 0$ if $1 \le j < i \le n$ or $n < i \le m$.

We use induction method. We make m our induction variable.

(1) When $m = 2$, $n = 1$. $R^{(2)} = G^{(2,1)}A$. Assume $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, then $G^{(2,1)} = \begin{bmatrix} \frac{a_{11}}{\sqrt{a_{11}^2+a_{21}^2}} & \frac{a_{21}}{\sqrt{a_{11}^2+a_{21}^2}} \\ \frac{-a_{21}}{\sqrt{a_{11}^2+a_{21}^2}} & \frac{a_{11}}{\sqrt{a_{11}^2+a_{21}^2}} \end{bmatrix}$, so $R^{(2)} = \begin{bmatrix} \sqrt{a_{11}^2+a_{21}^2} & \frac{a_{11}a_{12}+a_{21}a_{22}}{\sqrt{a_{11}^2+a_{21}^2}} \\ 0 & \frac{-a_{21}a_{12}+a_{11}a_{22}}{\sqrt{a_{11}^2+a_{21}^2}} \end{bmatrix}$ is upper triangular.

(2) When $m = k$, $n < k$, we assume $R^{(k)} = G^{(k,n)} \cdots G^{(k,2)} G^{(k,1)} \cdots G^{(3,2)} G^{(3,1)} G^{(2,1)} A$ is upper triangular.

(3) When $m = k+1$, let's prove $R^{(k+1)} = G^{(k+1,n)} \cdots G^{(k+1,2)} G^{(k+1,1)} R^{(k)}$ is still upper triangular. Since from $R^{(k+1)}$ is derived from $R^{(k)}$ by zeroing out the elements in $k+1$ row one by one and from left to right, so we will again use induction to prove

that $R^{(k+1)}|_l = G^{(k+1,l)} \cdots G^{(k+1,2)} G^{(k+1,1)} R^{(k)}$ has the following form:

$$r_{ij}|_l = 0 \quad \text{if} \quad 1 \le j < i \le n \quad \text{or} \quad n < i \le k \quad \text{or} \quad i = k+1,\ 1 \le j \le l \tag{A.1}$$

(3.1) For step 1, $r_{ij}|_1 = \begin{cases} r_{ij} & , i \ne 1, k+1 \\ r_{1j}\cos(\theta) - r_{k+1,j}\sin(\theta) & , i = 1 \\ r_{1j} sin(\theta) + r_{k+1,j} cos(\theta) & , i = k+1 \end{cases}$. So for $1 \le j < i \le n$ and $n < i \le k$, $r_{ij}|_1 = r_{ij} = 0$, while $i = k+1$, $j = 1$, $r_{ij}|_1 = r_{11}\frac{-r_{k+1,1}}{\sqrt{r_{11}^2 + r_{k+1,1}^2}} + r_{k+1,1}\frac{r_{11}}{\sqrt{r_{11}^2 + r_{k+1,1}^2}} = 0$. So statement A.1 holds.

(3.2) Suppose for step l, statement statement A.1 holds as well, i.e.

$$R^{(k+1)}|_l = \begin{array}{c|c} \begin{bmatrix} r_{11}|_l & r_{12}|_l & \cdots & r_{1l}|_l & r_{1,l+1}|_l & \cdots & r_{1,n}|_l \\ 0 & r_{22}|_l & \cdots & r_{2l}|_l & r_{2,l+1}|_l & \cdots & r_{2n}|_l \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & r_{nn}|_l \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & r_{k+1,l+1}|_l & \cdots & r_{k+1,n}|_l \end{bmatrix} & \begin{array}{c} 1 \\ \\ l \\ n \\ \\ l \\ \\ k+1 \end{array} \end{array}$$

(3.3) We now prove for step $l+1$, A.1 still holds, i.e. $r_{ij}|_{l+1} = 0$ if $1 \le j < i \le n$ or $n < i \le k$ or $i = k+1$, $1 \le j \le l+1$. We know that $l+1 \le n$. $r_{ij}|_{l+1} = \begin{cases} r_{ij}|_l & , i \ne l+1, k+1 \\ r_{l+1,j}|_l \cos(\theta) - r_{k+1,j}|_l \sin(\theta) & , i = l+1 \\ r_{l+1,j}|_l\, sin(\theta) + r_{k+1,j}|_l\, cos(\theta) & , i = k+1 \end{cases}$

For $1 \le j < i \le n$, we know $r_{ij}|_l = 0$. If $i \ne l+1$, $r_{ij}|_{l+1} = r_{ij}|_l = 0$. If $i = l+1$, since $j < i$, so $j < l+1$, and so $r_{l+1,j}|_l = 0$. For $j < l+1$, i.e. $j \le l$, from (3.2), we know that $r_{k+1,j}|_l = 0$, so $r_{l+1,j}|_{l+1} = r_{l+1,j}|_l \cos(\theta) - r_{k+1,j}|_l \sin(\theta) = 0 - 0 = 0$.

For $n < i \le k$, $r_{ij}|_{l+1} = r_{ij}|_l = 0$

For $i = k+1$, $1 \le j < l+1$, $r_{l+1,j}|_{l+1} = r_{k+1,j}|_l\, sin(\theta) + r_{k+1,j}|_l\, cos(\theta) = 0 - 0 = 0$.

For $i = k+1$, $j = l+1$, $r_{l+1,j}|_{l+1} = r_{l+1,l+1}|_l \frac{-r_{k+1,l+1}}{\sqrt{r_{l+1,l+1}^2|_l + r_{k+1,l+1}^2|_l}} + r_{k+1,l+1}|_l \frac{r_{l+1,l+1}}{\sqrt{r_{l+1,l+1}^2|_l + r_{k+1,l+1}^2|_l}} = 0$.

Therefore, A.1 stands for $l+1$.

Above all, we have proved that in $R^{(k+1)}|_l = G^{(k+1,l)} \cdots G^{(k+1,2)} G^{(k+1,1)} R^{(k)}$, $r_{ij}|_l = 0$, if $1 \leq j < i \leq n$ or $n < i \leq k+1$ or $i = k+1$, $1 \leq j \leq l$. If $l = n$, we have $r_{ij} = 0$ if $1 \leq j < i \leq n$ or $n < i \leq k+1$, which means $R^{(k+1)}$ is upper triangular. Hence, (3.1) to (3.3) completes that proof that the reduction

$$R^{(m)} = G^{(m,n)} \cdots G^{(m,2)} G^{(m,1)} \cdots G^{(n+1,n)} \cdots G^{(n+1,2)} G^{(n+1,1)} \cdots G^{(n,n-1)} \cdots G^{(n,2)} G^{(n,1)} \cdots G^{(3,2)} G^{(3,1)} G^{(2,1)} A$$

transform A to an upper triangular matrix $R^{(m)}$, and so is a valid sequence for QR decomposition.

Vita

Yang obtained his Bachelor degree from Beijing Jiaotong University, where he graduated first place, and was a three-time recipient of the National Scholarship. He started graduate study at the University of Tennessee, Knoxville from August 2014. In 2016, he earned a M.S. in Computer Engineering. He is expected to get his PhD in August 2017, together with a M.S. in Statistics.