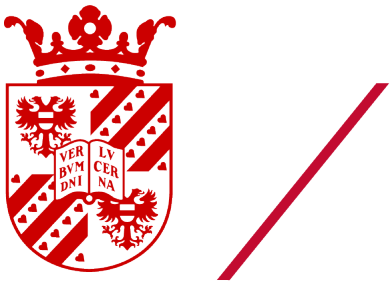


university of groningen

faculty of science and engineering

Thomas Titherington

Mobility traces analysis: Studying mobility patterns, allowing insight into the operation of a smart city.

Now that smartphones and smart devices are ubiquitous in everyday life, these devices are constantly monitoring their diverse environments. From your mobile phone calculating the number of steps you have walked today, to a climate control system found in your home. The advancements and maturation of these devices now allow us to collect and analyse a huge amount of varied data, leading to interesting and sometimes surprising results. This project analyses the T-Drive data set provided by the Microsoft Research team, which contains the one-week trajectories of 10,357 taxis. The project proposes a framework that can be used to study large vehicular trajectory data sets, providing inferences on the mobility patterns within. Specifically, we show how these patterns can be used in the context of a smart city by using the taxi trajectories to create bus stops.

Date
July 2020
Supervisors
Boris Koldehofe & Viktoriya Degeler

Contents

1 Introduction **2**

2 Project Description **3**

3 Literature Survey **5**

3.1 Related Work 5

3.2 Methods 6

3.3 Methods Evaluation 7

4 Approach **10**

4.1 Data 10

4.2 Architecture 10

4.2.1 Pre-processing 11

4.2.2 Stay points 11

4.2.3 Clustering 12

5 Results **17**

5.1 Stay Points 17

5.2 Clustering 18

6 Conclusion **22**

7 Future Work **23**

8 Appendix **25**

1 Introduction

Twenty-five years ago, the world was introduced to its first smartphone - the IBM Simon. At the time, the word smartphone would not have meant a great deal to anyone. Contra to just over ten years later, when the first iPhone was launched, the term smartphone took on new meaning. Jump to today, and its developing further, as well as its associated fields. Smart devices such as these have come so far, now collecting a hugely diverse range of data. This now gives us the opportunity to analyse this data, allowing inferences to be made and applied to different contexts that were not possible before.

In the context of a modern city, this could have many beneficial impacts upon the inner workings of the city. As the number of inhabitants of the worlds cities increases at a rapid rate, more focus must be placed on efficiently using space and reducing emissions, allowing the city to become long term oriented. One can start to solve these enigmatic problems by using large data sets collected by smart devices. For example, collecting and analysing traffic data within the city can be used to streamline public transport services. Ensuring that bus routes truly visit locations that are frequently visited by the cities population. Additionally, being sustainable is more relevant today than ever before. Grouping similar trips that are made often can help to reduce the amount of vehicles on the road, which clearly has instrumental benefit on the environment as well as the traffic flow and congestion within the city. Consequently, the work reported in this project is motivated by the importance of using large trajectory data sets in the context of a smart city to solve the emerging problems outlined above.

In this project we demonstrate how one can use trajectory data-sets to identify key areas within a geographical region. We further show how these key areas can be used to identify optimal locations for bus stops. Finally we establish a clustering approach that is suitable for large trajectory data-sets, and one that can be used in an online fashion.

The main focus of the project, however, is on the clustering techniques that can be used effectively on large trajectory data sets, and to illustrate how this can be done. What we identify in this project is a clustering approach that identifies similarities between trajectories. Further showing how trajectory data can be used to identify popular geographic regions and apply these findings to create bus stops. We show why such a clustering approach is well suited to large trajectory data-sets. Mostly importantly, one of the biggest advantages of this approach is that it is completely scalable and can be applied in an online fashion as it does not require the whole-data set to have been processed or loaded in advance. As a result, the overall processing pipeline can be executed in parallel. Additionally, the technique described here can be generalised for use on different trajectory data-sets. For instance, using GPS trajectories from a set of private vehicles or even trajectories described by location cookies collected by smart phones.

When examining existing research, there appears to be a research gap. This gap lies between collecting and analysing big data within a city, and its application to other comparable scenarios within the city. By using the data generated by a cities population we can expose the potential weaknesses in the public services, once problems are identified they can be improved and services can be tailored to meet the populations needs. In summary, this project aims to fill that gap, bypassing the need for surveys or customer reviews, this idea lets the data speak for itself. Furthermore, this approach presents a clustering technique that can be applied in an online fashion, which sets it apart from existing research into mining large trajectory data-sets.

2 Project Description

In order to grasp the problem that this project addresses, its best to understand what a trajectory data-set consists of. As such, data-sets that share these common attributes can be used by the framework presented in this project. The term *large trajectory data-sets* is used frequently throughout the project, typically this is a trajectory set with millions of points and thus one in which the entire data-set cannot be loaded into main memory.

GPS point: A GPS point p, is found in a GPS log and consists of the following properties. $p = (t, lng, lat, dt)$ where t is the corresponding taxi identification number, lng the longitude, lat the latitude, dt the date and time.

Trajectory: A trajectory is a list of GPS points ordered with respect to date and time. $t = \{p_0, p_1, ..., p_n\}$ such that $p_0.dt < p_1.dt < ... < p_n.dt$, where n is the number of points in a trajectory.

Now that we have a better understanding of what the project entails, we can formulate the research goals precisely as follows.

Research goals:

1. *Given a set of vehicle trajectories, segment each trajectory into parts that represent key points within a journey.*
2. *Given the key segments of vehicle trajectories within a geographic range, we wish to identify areas of high significance at various spatial and temporal scales.*
3. *Show how these regions of high significance can be used to create bus stops.*
4. *Identify a suitable clustering approach that is scalable and efficient for use on very large trajectory data-sets.*

Elaborating on the research goals outlined, the contribution of this research lies in the following areas:

- Understanding the benefits and drawbacks of using different clustering approaches on large trajectory data-sets and illustrating why one approach is better suited than another.
- Presenting a scalable and efficient spatial clustering approach for use on large trajectory data-sets. One that takes advantage of strong data locality thus minimising IO operations, and resulting in low page faults.
- Demonstrating that we can use a spatial clustering approach to identify the similarities in one trajectory data-set and further apply the findings to the creation of bus stops.

Upon inspection of the research goals, it can be seen that each goal poses numerous challenges. With regard to the segmentation of a vehicles trajectory, we must define exactly what constitutes a key point within a trajectory. Furthermore, how does one identify this? What makes a geographical region significant, more importantly what characteristics must it possess? Since traffic flow and congestion are heavily influenced by the time of day and occur within some geographic range, it is important that spatial and temporal constraints are identified. Clearly defining and bounding variables ensure that the results produced are accurate and applicable to the real world. Finally, we want to ensure that the bus stops we propose fulfil their function and will be used by the public. There are a great deal of influencing factors here, including, but not limited to, the locations of the bus stops, the distance between bus stops on a route, the distance a client has to walk to board a bus and the number of clients a bus stop services. In

the Approach section of the project, the aforementioned problems and challenges are discussed in detail and strategies for solving them are introduced.

The taxi trajectory data set used here, is the T-Drive data set [1][2]. It contains the trajectories of 10,357 taxis in Beijing over the course of one week in February 2008, and is provided by the Microsoft Research team.

3 Literature Survey

3.1 Related Work

As the popularity and adoption of smartphones and smart devices increases, so does the amount of data collected by these devices. Due to the rise in big data within the geographical movement area, new methods are needed to address this quantitative approach [3]. J.A. Long et al. is critical of traditional methods used to analyse movement tracking, stating that new structures and methods should be considered. Now that the range of data collected has become incredibly diverse, and the technology used to capture them becoming increasing prevalent in the market, methods for extracting useful information are lagging behind. He addresses seven types of methodologies that can be used to analyse movement data. Specifically, the identification of patterns and clusters in large geographical movement data sets focus on one of space, time, or space-time. Methods used to identify spatial clusters first look at space and then time. The simplest techniques for the detection of clusters on spatial movement data sets generally require that trajectories from individual paths be represented as spatial points. Other spatial first methods look to define regions of interest and identify times at which movement trajectories are clustered in these geographical regions. On the other hand, a temporal first approach can be taken - we first look at time and then space. [4] For instance, we filter the data set based on temporal constraints and run spatial clustering on the filtered dataset. This filtered data set then captures a specific time frame. Furthermore he states that developing predictive frameworks for movement data is an area that should be studied in future research as research within the field is lacking.

As mentioned, the maturation of these sensors and smart devices allow us to collect a large array of data. An example of which is the M-Atlas project carried out by F.Giannotti et al. [5] M-Atlas is a query driven system that is focused on trajectory data, it uses the GPS data collected from tens of thousands of private cars in the city of Milan and Pisa. The project is interested in peoples travel patterns and the flow of certain groups, specifically, how large events and attractions within a city influence mobility, for instance a football match. The system also is used to predict and characterise traffic jams and congestion. M-Atlas uses a density based type of clustering [6], allowing the interpretation of groups of objects having similar properties and behaviours. However, since trajectories of moving entities are structurally complex objects, G. Andrienko et al. argue that they cannot be represented in a traditional form - points in a multi-dimensional space of properties - as this is very computationally intensive and not scalable to large data-sets. Their solution, is to combine clustering and classification. In short, a subset of objects with clustering applied is taken. Then a classifier is built, which is used for attaching new objects to the existing clusters. This can be modified by an analyst if needed. The classifier is applied to the whole data-set, meaning each object is attached to one of the clusters or left unclassified if it does not fit into any clusters. This procedure can be repeated to the remaining unclassified objects. This type of geographical clustering would be classed as spatial first method of classifying movement data, in terms of J.A Long et al. [3]

P. S. Castro, et al. [7] state, in their mobility analysis paper, that the digital footprint left by the increasingly popular smart devices are a reflection of the economical and societal interactions of a community. As a result, GPS traces allow us to reveal many valuable facts about social and community dynamics. Within, they separate existing work into three categories: social dynamics, traffic dynamics and operational dynamics. In the area of social dynamics, the ability to detect the most frequented locations in a city can be useful for urban planning, public transport, tourism and security companies. In [7] this ability is called *Hotspot Extraction*. Since taxis provide a precise indication of peoples desired destinations, they paint a more accurate representation of a cities attractions compared with public transportation. There has been substantial research on using GPS trajectories from personal devices to detect such significant locations within a geographical region. Ashbrook and Starner [8] characterise hotspots based on GPS signal over a time frame. The idea being that an area with faded GPS signal would indicate a place of significance, similar to what would occur when inside a building. However this research is quite dated (2003) and cellar networks have improved massively since then. Alvares et al. [9] define stops in trajectories as polygons in R^2 where certain points in a trajectory spend a minimum amount of time. Another idea pre-

sented by Palma et al. [10] cluster sections of trajectories based on their speed, combining and building on the two previous ideas. Finally, Zheng [11] defines stay points as geographic areas where the GPS tracer has remained for a certain time and distance threshold. This solution presented by Zheng, appears to encapsulate the ideas from previous research into a single simple method.

3.2 Methods

In the data mining field there are many different approaches to grouping and classifying data. Cluster analysis is one method, in which there are a number of strategies. In the review [12], numerous clustering techniques are introduced, the main algorithms addressed are:

1. Partitional clustering - which attempts to decompose the data-set into a set of disjoint clusters using a certain criterion function.
2. Hierarchical clustering - an iterative approach that builds a hierarchy of clusters. Hierarchical strategies can usually be split into two categories. Agglomerative or bottom-up approaches where smaller clusters are merged into larger ones as one moves up the hierarchy. Conversely, a Divisive approach splits up larger clusters producing a tree of smaller clusters. Results of which are commonly presented in the form of a Dendrogram, that illustrates how the clusters are related to one another.
3. Density-based clustering - which clusters neighbouring objects of the data based on density conditions.
4. Grid-based clustering - space is quantised into a finite number of cells and then, operations are done on the quantised space.

Each method has its own particular merits and flaws, as such, one type of clustering approach may be better suited to certain situations over others. The algorithms that are listed in this section are by no means an exhaustive list, the algorithms discussed are those that are relevant to the project.

K-Means clustering is a partitional clustering algorithm and is one of the simplest and well known clustering methods. Simply put, it aims to group n observations into k clusters so that each observation in the set belongs to the cluster with the nearest cluster centre, known as a cluster centroid. In order to achieve this, K-Means tries to minimise the within-cluster sum of squares:

$$\sum_{i=1}^{N} \mathbf{min}_{\mu_j \in C}(||x_i - \mu_j||^2)$$

Where $\mu_j \in C$ is the set of k clusters and $x_i \in X$ is the set of n observations.

The within-cluster sum of squares essentially represents how closely related the data is, as a result by minimising this function we ensure that an observation vector is assigned to the closest cluster centroid, up to a certain threshold value. If we are clustering points in two dimensions, by minimising the within-cluster sum of squares we are minimising the average distance between points and their cluster centroids. At each iteration in the algorithm we update each cluster centroid value based on the mean of points assigned to that cluster and thus must recalculate this within-cluster sum of squares using the updated values. This process of recalculation is repeated until some threshold is met, i.e. the within-cluster sum of squares criterion reaches some tolerance level.

Hierarchical clustering, specifically BIRCH [13] is a hierarchical clustering algorithm that makes use of a Clustering Feature Tree (CFT). Essentially, over a single pass of the data, the data set is transformed

into a CFT. Here we are grouping points into a tree, and only storing what is required for us to make all necessary calculations on the data vectors, nothing more. Within the CFT we store summaries about the clusters within, these are known as Cluster Features (CF). These CF entries are what make up the nodes in the tree and can be defined as follows.

$$CF = (N, \vec{LS}, SS)$$

Where N is the d-dimensional data points within that cluster. $\vec{LS} = \sum_{i=1}^{N} \vec{X_i}$ is the linear sum of points in the cluster. $\vec{SS} = \sum_{i=1}^{N} (\vec{X_i})^2$ is the square sum of the data points with the cluster.

To build the CFT we need a couple of parameters that will dictate the shape of the tree. The *branching factor* can be seen as the order of the tree, that is, how many cluster features are allowed within a node. The second parameter is the *threshold*, this indicates the maximum allowable radii for clusters in the tree.

As mentioned above, M-atlas [5] uses density based clustering, more precisely it uses the clustering algorithm OPTICS [14]. In order to understand OPTICS I will first introduce its ancestor DBSCAN. [15] Density-Based Spatial Clustering of Applications with Noise, commonly known as DBSCAN, identifies areas in the data-set with a high density of observations, along with areas that have a low density of observations, sorting these into clusters of varying shapes. The main concept behind DBSCAN is the notion of *core samples* - samples that are in areas of high density. Clusters created by DBSCAN can be understood as a set of these core samples. Neighbours of the core sample are those that are within a distance of *eps*. What DBSCAN is trying to do is extend this set of core samples at each iteration, checking which points meet both a distance and density condition. The algorithm can be abstracted as follows:

1. Find the points in *eps* neighbourhood of every point and identify core points.
2. Try to extend this neighbourhood by finding the connected components of core points, ignoring all non-core points.
3. For each non-core point found, assign it to a nearby cluster if the cluster is an *eps* neighbour. Otherwise, it is considered noise.

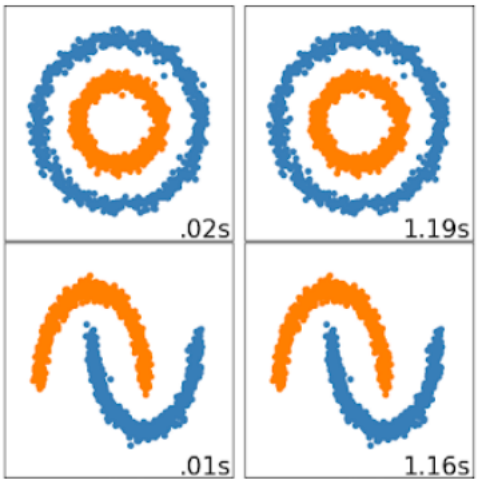
OPTICS was created to address the issues with DBSCAN. Allowing for varying density between clusters unlike DBSCAN which assumes constant density, and eliminating the distance *eps* parameter.

3.3 Methods Evaluation

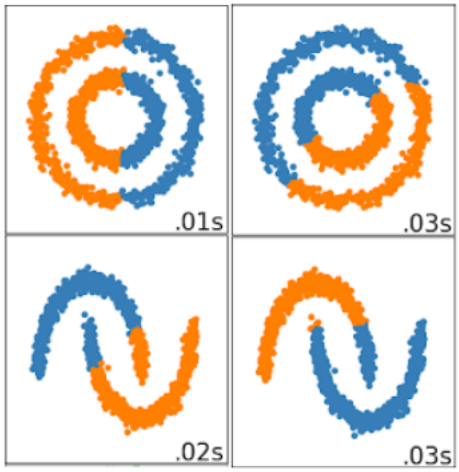
In order to identify the clustering technique that is best suited here, we must consider what criteria are important in this context. We can classify the criteria into two categories. For a bus stop to be convenient, it needs to serve as many clients as possible as well as minimising the distance that a client has to walk to it. These are *problem constraints*. Since we are dealing with large trajectory data-sets with potentially tens of millions of points - the approach must be scalable. When you deal with such a large number of points it is not always possible to load the whole data-set into main memory and so this is an important requirement of the chosen algorithm. Additionally, we want to minimise the number of times we read the data, and take advantage of how the data is stored. These can be see as *data constraints*.

Due to their focus on core samples, both DBSCAN and OPTICS deal with outliers very well, meaning that in some cases data pre-processing is not required. In addition, the number of clusters you wish to find does not have to be defined in advance, unlike K-Means. Despite this, they do have their drawbacks. Both these algorithms can assign points that are far away to the same cluster, as long as the density between

Figure 1: Two clusters formed shown by blue and orange colours, in two separate point distributions (top and bottom). - Scikit-learn developers. 2020



(a) DBSCAN (left) and OPTICS (right).



(b) K-Means (left) and BIRCH (right).

them is maintained. This is shown by the blue outer circle in Figure 1a. This may become a problem if we want to create bus stops. For example, a long line of points with relatively high density will be assigned to the same cluster. This means that people will potentially have to walk a long way to the location of the bus stop. Moreover, it could be the case that the centre of some other cluster will end up being closer to them. Contrast this to the results produced by K-Means and BIRCH - Figure 1b. Since neither algorithms are density based, the density of the distribution of points is not considered. What determines the grouping of clusters here is the distance metric specified by the algorithm, in two dimensions that could be Euclidean distance. This rules out both our density based approaches, neither can guarantee that our problem constraints will be satisfied.

This leaves us with K-Means and BIRCH. Due to its efficient simplicity the K-Means algorithm is relatively easy to implement. However, K-Means requires the number clusters to be known in advance, which can be a problem if little is known about the data set in advance, although there are methods to address this problem, such as the Elbow method. Having said that, the real issue with K-Means is not that we need to know the amount of clusters in advance, but that it is not well suited to large data sets. This is

because at each iteration we need to calculate the within-cluster sum of squares as well as the new centroid values of each cluster, requiring us to iterate over all points in the data-set. Due to its efficient representation of clusters, BIRCH on the other hand, does not suffer from the same problem. It is actually possible to only read the data-set once, inserting each point into the tree. The efficiency that we want to harness from BIRCH lies within the use of the Cluster Feature Tree, as such in this project we make use of the Cluster Feature Tree.

4 Approach

4.1 Data

The data-set used in this project is the T-Drive [1][2] data-set from the Microsoft Research team. It contains the GPS trajectories of 10,357 taxis in the city of Beijing during the period of Feb. 2nd to Feb. 8th 2008. The total number of points in this dataset is around 15 million and the total distance of the trajectories reaches to 9 million kilometres. The average distance interval is about 623 meters with an average time interval of 177 seconds. Each taxis trajectories out of the 10,357 are split into separate CSV files. The number of GPS points within a trajectory significantly varies, there are many files that contain no entries, some contain a handful of points (under twenty) and others contain hundreds of points over the course of a particular week in February.

To store the data-sets produced at various stages in the pipeline I use a HDF5 store. Note that, it is not necessary to store the outcome at each stage in the pipeline, the results can just be passed to the next stage. Once the transformed data points reach the final stage and are added to the tree, the tree can be written to disk. This is useful if you wish to query the tree at a future point in time. In the HDF system, data is stored hierarchically, similar to organisation of the UNIX file system. Consequently, here we split the HDF5 file into parts that correspond to a step in the pipeline described in the upcoming section.

The HDF5 format is extremely portable, allowing content to be read and written on machines with different architecture. This is possible because the HDF5 format stores metadata about the data-sets it contains, for instance the size, type and Endianness. The main reason that it is used in this project however, is because both read and write times are much faster compared with other alternatives such as SQLite. HDF5 uses chunked compression which means the file can be read in chunks, only decompressing the chunk the user wants to read, instead of the entire file or database. Additionally, HDF5 files are stored in continuous fashion on the disk decreasing read and write time, ensuring that the our approach makes use of strong data locality in turn minimising page faults.

4.2 Architecture

As mentioned in the problem description, the research goals pose numerous challenges and questions that must be addressed by a framework such as this. For instance, we can start with the question, how can we identify if two trajectories are similar? The first law of geography [16] states that *everything is related to everything else, but near things are more related than distant things.* With this principle in mind, we can plot two trajectories with respect to their longitude and latitude. When examining their individual points, we may see that some points share longitude and latitude coordinates that are geographically close to each other. Using the first law of geography we could stop here and say that trajectories with geographically close points should be considered similar. But what do we mean by geographically close? Furthermore, we are only considering two variables here, if we introduce a third - time - are these trajectories still similar? This appears to be somewhat subjective.

The problem here, however is not that simple. Even if we deem these trajectories to be similar, what is this really telling us? We need to think about the points that make up a trajectory. Points alone do not indicate significant geographical areas, all they tell us, is the objects (taxis) position at some recorded time. As a result, we are not just looking at points but the correlation between them. What is presented in this section is how one can take a set of trajectories and identify the correlations within. Each step defined here makes up the overall processing pipeline as a result this approach can be executed in parallel. The biggest advantage, however is that it can be applied in an online fashion as it does not require the whole data-set to have been loaded or processed in advance.

4.2.1 Pre-processing

This step is designed to filter outliers present in the dataset, removing points that should not belong in the dataset. For instance, it is not uncommon for GPS trackers to record points that appear to be inconsistent with the trajectory of an object. Figure 2 illustrates this idea. In order to remove this noise in the data, we can apply a filter, mentioned here [11], which works as follows.

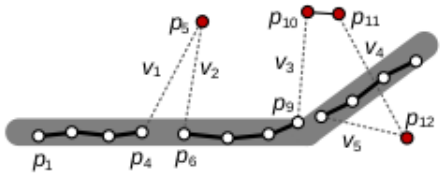


Figure 2: Noise Points - Yu Zheng. 2015. Trajectory data mining: An overview.

We calculate the speed between two points based on the time interval and distance between a point and its successor. This is called a segment. Segments with a speed larger than a certain threshold will be removed. For example, a threshold of 170 km/h. If a segment is removed, we must check its neighbours that fall within a certain distance. If they fall within this distance they should also be removed. This second distance check is designed to remove points such as p_{10} and p_{11} in Figure 2.

4.2.2 Stay points

The second role of data pre-processing is to transform the input data into a format that can be used by the future steps in the sequence. At present, all the data-set contains is coordinates of taxis at certain time intervals. This is rather limited. In order to give these points value we can further distinguish between them.

If we want to identify key areas within a trajectory, we must think about the attributes of a trajectory. Currently, each spatial point in a taxis trajectory is assumed to be of equal importance. However, some of these points may denote locations where a taxi has remained for a significant period of time. For instance, a taxi may be stuck in traffic at rush hour or a taxi may be dropping a passenger off at some tourist attraction. These points can be grouped and categorised as stop points - locations where a taxi has remained for a significant amount of time. This is what Zheng refers to as *Stay Points* [11], stating that two types of stay points can be identified. One is a single point location - *Stay Point 1* - where someone has remained at the same point for a period of time. *Stay Point 2* - where a number of points fall within a certain radius, relative to a centre point and a time threshold. Figure 3 shows the two types of stay points described. The terms *stop points* and *stay points* are synonymous in this project.

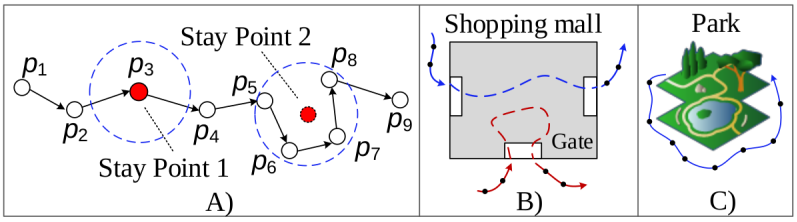


Figure 3: Stay Points - Yu Zheng. 2015. Trajectory data mining: An overview.

In the context of taxi trajectories, discerning between the types of stay points is not important since both

can indicate a significant location. Furthermore, the first type of stay point is unlikely to occur, since a GPS tracker usually generates different readings even when positioned in the exact same location. This gives rise to the following definition.

Stay points: A stay point $S = (t, lng, lat, arvl, dept)$ where t is the corresponding taxi identification number, lng the longitude, lat the latitude, $arvl$ is the date-time at which the taxi arrived at the location and $dept$ is the date-time when the taxi left that location. A stay point represents a spatial region of significant importance and consists of grouped points within a trajectory. As such, we can identify a stay point using the following criteria.

$$S = p_1 \overset{\Delta t_1 + \Delta d_1}{\rightarrow} p_2 \overset{\Delta t_2 + \Delta d_2}{\rightarrow} \ldots \overset{\Delta t_{N-1} + \Delta d_{N-1}}{\rightarrow} p_N$$

Where $p_i,\ 0 < i \leq N$ represents a point a taxi has visited. $\forall j < i :\ \Delta t_j > T_t$ and $\Delta d_j < T_d$

In order to detect stop points, we can set a threshold with respect to the difference in time T_t and distance T_d. This means all points within the threshold are candidate stop points. Adapting the stay point detection algorithm stated in [17], we obtain the stop point detection function that produces a set of stop points for each taxi and stores them in the format specified in the definition above. Once filtering has been applied and stop points have been detected. We can move on to the next step in the pipeline.

For our findings to be applicable to reality we need to ensure that temporal constraints and spatial constraints are well defined and maintained. Our clustering approach deals with this spatial constraint, grouping points in R^2 that are spatially close. Before we can cluster the data-set we need to deal with the third variable - time. If we were to run the clustering approach on the whole data-set, that is, GPS points over seven days, although it will produce results that appear to identify key areas of the region, how applicable are they to reality and what can we infer from them? Since we're clustering over such a large timeframe its hard to see the change over time and we cannot say that the areas identified on a Friday are the same as those visited on a Tuesday. Consequently, we need to first filter the stay points created based on some temporal constraint, the choice of which is up to the analyst. For instance, one might wish to identify the impact that rush hour has on the areas visited, and so one would choose a very specific constraint, i.e. 5:00 - 6:00 P.M on a Tuesday or even the same one hour time frame on each day of the week. In this project however, the stay points are separated based on the day of the week.

4.2.3 Clustering

As we illustrated at the start of this section, judging the similarity between two trajectories is somewhat subjective and non-binary. From the GPS trajectories, we cannot say for definite that two trajectories are identical, hence similarity is a spectrum. This actually makes a clustering approach well suited to this application as, in short, clusters of higher similarity are created and grouped together.

Creating the CF Tree

In our methods evaluation section we demonstrated the importance of using a clustering technique that is well suited to a given scenario. Consequently it was decided that we will make use of the Cluster Feature Tree behind the BIRCH algorithm and so, the clustering approach that is used here is a variant on the Clustering Feature Tree. To understand the motivation behind this clustering approach described here and how it differs on that of regular BIRCH clustering, let us discuss the differences in more detail. BIRCH comprises of multiple phases, each designed to refine the CF Tree and the clusters produced.

- **Phase 1:** is the initial step and consists of loading the data-set, requiring us to pass over the whole data-set and build the CF Tree.

- **Phase 2:** is used to condense the size of the tree so that there are less clusters found at the leaf nodes. The reason for this step is to prepare the tree for Phase 3. Some clustering algorithms have constrains on the size of the data-set that can be clustered, as such, it is an optional step.
- **Phase 3:** applies another clustering algorithm to the clusters found at the leaves of the tree. According to [13] the undesirable effect of the skewed input order, and splitting triggered by page size (the branching factor) causes us to be unfaithful to the actual clustering patterns in the data. Re-clustering here is said to remedy these undesirable effects.
- **Phase 4:** essentially re-clusters the data-set. It uses the centroids of the clusters produced in Phase 3 as seeds and redistributes the data points to its closest seed. What this step ensures is that if there are any copies in the data i.e. duplicate points, they will all be moved to the same cluster. Which can not be guaranteed in each of the other steps. Additionally, this step allows points assigned to a cluster to move to another cluster, if it is deemed to be closer.

Since we are looking for a clustering approach that is robust and suitable for use on very large trajectory data-sets using an online clustering approach, we need to minimise unnecessary data reads and computationally expensive operations. To ensure accurate clusters are formed, BIRCH relies on each step of the pipeline, as each step helps to refine the CF Tree and improve the quality of the clusters produced. However, each step in the pipeline comes at a cost. If we can build a tree with a structure that ensures the observations and underlying clusters within are accurately represented by the tree, then subjecting the tree to additional refinement steps will likely have little effect.

Phase 1 is essential, without this step, no clusters are produced. Phase 2 and Phase 4 on the other hand are optional. Phase 2 is comparable to lossy compression, we are inserting all data points into the tree and only keeping the clusters formed at the leaf nodes of the tree. This stage is optional, in some cases we may want to reduce the number of clusters that are formed at this stage, but we do not need to here. Phase 4 gives points the opportunity to move clusters, as such, this stage requires an additional pass of the entire data-set and can be costly when dealing with large data-sets. According to [13] the inaccuracies that Phase 4 deals with are in reality rare and minor, therefore this stage is optional. As introduced above, Phase 3 further applies another clustering algorithm to the clusters found at the leaf nodes in the tree, specifically its purpose is to deal with potential inaccurate clusters produced by the original tree created in Phase 1. In [13] they do not elaborate on what they mean by skewed input order. However, if we study the part of the CF Tree algorithm that deals with insertions, we can see that the order in which we enter points does indeed have an effect on the clusters produced. In practice though, the main influence on the resulting clusters is by the threshold and branching factor. They are what determine the shape of the tree and ultimately, the clusters produced. To better understand this effect, let us consider an example.

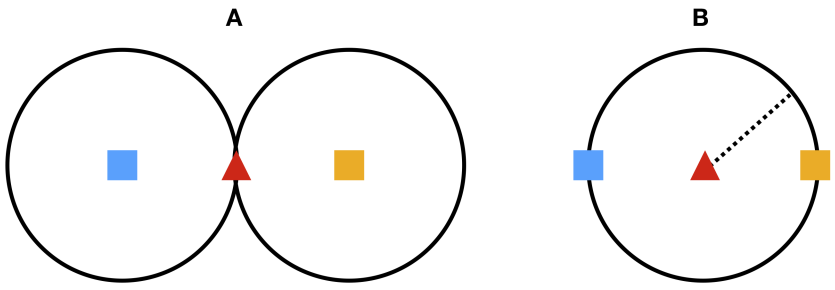


Figure 4: (A) Red triangle is the incoming point, both squares are established cluster centroids. (B) Both squares represent incoming points, red triangle is an established cluster centroid.

Observe the incoming point in Figure 4 A. The cluster in which the point joins is determined by the threshold, in this situation we have two overlapping clusters. According to the chosen threshold, the point could belong to either cluster as it is on the fringe of both circles. Based on the distance metric that is chosen

(e.g. euclidean in R^2), we choose the cluster centroid that is closest to the incoming cluster. If we consider a different order of insertion shown in Figure 4 B, it is still possible that cluster overlapping will occur, as points here are on the fringe of the cluster. As such they may create a new cluster feature entry all together or might be absorbed by the existing cluster. Again, this all depends on the threshold value chosen, which is represented by the dashed line in the diagram. What this illustrates is that by choosing our threshold and branching factor parameters carefully we can prevent the problems caused by the order of point insertions, therefore we can bypass the need for Phase 3. Before reaching Phase 3, if we have a CF Tree that is representative of the true clusters within the data-set we circumvent the need for additional clustering in Phase 3. In order to do so, we must first think about how both the threshold and branching factor influence the shape of the tree and the clusters produced.

As introduced in the methods survey, CF Tree is a height balanced tree with two parameters. The *threshold* constraint states the maximum allowable radii for each cluster in the tree, in other words, requiring all clusters within the tree to be smaller than a certain radius value. This means that a low threshold will lead to *cluster splitting* because a lower radius value will force the tree to create more clusters. A high threshold will lead to *clustering combining* by increasing the range of acceptable points within each potential cluster. For instance, sometimes a point can be added to a cluster when it lies on the fringe of the threshold, even though it is potentially part of a separate cluster - as we showed in Figure 4. Additionally, a *branching factor* must be chosen, this is the maximum number of cluster feature entries allowed within a node, essentially the order of the tree. Accordingly, the height of the tree produced is significantly influenced by this value. A high branching factor will create a shallow tree as more cluster entries are allowed within a node, thus a low branching factor will increase the height of the tree. In addition, splitting can also occur from overlapping clusters at different layers in the tree, and can be dealt with by choosing a high branching factor. However, a lower branching factor allows us to improve the granularity, by increasing the height of the clustering feature tree.

Splitting and combining often co-occur and are determined by the threshold. More specifically, these two properties influence both the radius R of the clusters and the distance between them D. Since the threshold dictates the cluster radius we could actually find a good estimate of this value before creating the tree. Using the ideas in [18] as a basis for this argument, if we take random sub-samples of the stay point data-set and run the K-Means algorithm on this data-set we can use the average radius between the clusters produced to choose our threshold. We can obtain the average radius using the within-cluster sum of squares introduced in the methods survey.

$$\overline{R} = \frac{\sum_{i=1}^{N} ||x_i - \mu_j||^2}{K}$$

Concretely, we do as follows.

1. Find K by running the Elbow-method on N random sub-samples.
2. Run K-Means on a random sub-sample and return the average cluster radius.
3. Use the returned average cluster radius as our threshold value.

Inserting points

Once we've created the tree. We can read the stay points calculated in the previous step and insert them into the tree in chunks. As such, the data-set can be read in one single pass. To insert a point into the tree we need to find its closest cluster, to do so lets look at the structure of the tree in greater detail.

Each non-leaf node contains at *most* B entries of the form $[CF_i, child_i]$ where $0 < i \leq B$. CF_i is a Cluster Feature and $child_i$ is the child node that the cluster feature points to. A non-leaf node represents a

cluster that summarises all the sub-clusters of its children. A leaf node contains at most B clusters, and must satisfy the threshold. That is, all cluster entries in a leaf node must have a radius less than T. When we insert a point we must descend the tree, at each layer checking the cluster that is closest to the incoming point, until we hit a leaf node. At this point we either add it to an existing cluster if adding this new point continues to satisfy T, otherwise we create a new cluster feature entry. This new entry is just the point itself, as such its centroid is simply the points coordinates and its radius is zero. This cluster feature creation also occurs when the tree is empty and we add our first point. The first cluster created is the first point that arrives.

When inserting, we read the stay points calculated in the previous step and insert them into the tree in chunks. Therefore, the data-set can be read in one single pass and so there is no requirement for the whole data-set to be loaded into main memory at one time. We choose a page size P, that allows the program to load P sized chunks of the data-set into memory and insert the points within into the tree. We continue this process until we have reached the end of the data-set or we know there will be no more incoming points (in the case the algorithm is implemented in an online fashion).

Saving the tree

When clustering has finished, we can save the clusters to disk. Clusters can be written to a store, here we use a HDF5 store with a structure similar to that of the Cluster Feature.

$$< cluster\_id, layer, n, ls_x, ls_y, ss, radius, centroid_x, centroid_y >$$

Storing clusters in such a way makes it well suited to queries. For example, we can query the tree at a chosen layer to see the order in which the clusters were visited by a certain taxi. A visualisation of this query is shown in Figure 5.

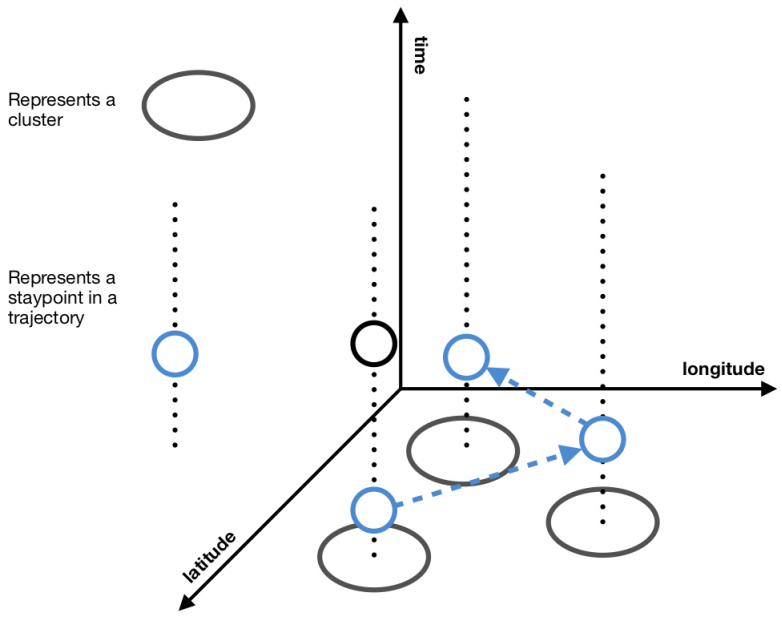
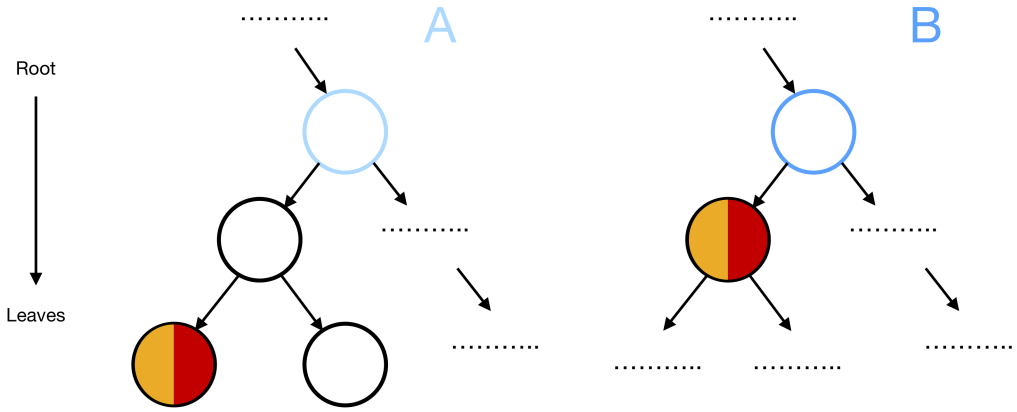


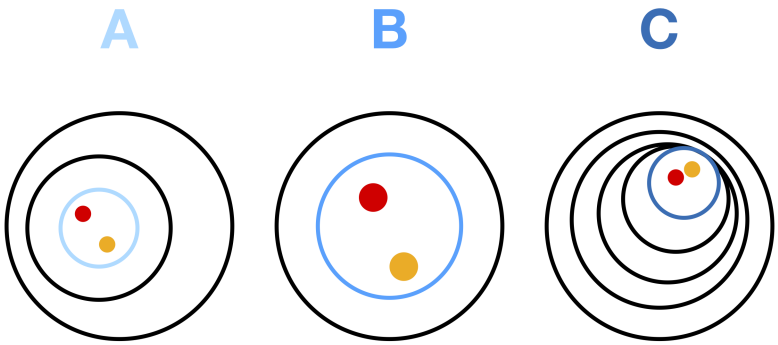
Figure 5: Visualisation of the sequence of clusters visited by a taxi.

Due to its hierarchical nature, stay points are hierarchically clustered using the CF Tree. Consequently similar stay points from different taxis will be assigned to the same clusters spanning different layers. Clus-

ters at the leaves of the tree are the most specific, taxis that share the same cluster(s) at this bottom level as closely related. In contrast to taxis that share the same cluster(s) at a higher level in the tree (closer to the root). As a result, each level in the tree describes the relationship and similarity between trajectories at a different spatial scale. The closer to the bottom of the tree, the closer the spatial scale. Figure 6 visualises this point.



(a) CFT view



(b) Spatial view

Figure 6: Visualisation of clusters - points that share clusters lower in the tree are closer related than points higher up the tree. (spatially)

5 Results

5.1 Stay Points

When identifying candidate stay points, we must compare each point against some time and distance threshold. We want to include points where a taxi has remained stationary for some reasonable amount of time within a small region. When choosing this *reasonable amount of time*, we must consider a typical taxi journey. A taxi first picks up its client and then makes its way to the desired destination. At both the start location and the destination the taxi will remain stationary for some time, this could be whilst the client loads and unloads their luggage and/or while they make a payment. Along the journey, the taxi may stop at a number of points, that are likely related to obstructions of the journey. Such as, heavy traffic, traffic lights, crossroads, pull-outs and other interferences regarding the layout of the road network.

Using the stay point detection algorithm described in the approach we can see that the higher the distance threshold, the more points the algorithm will merge, resulting in a lower amount of stay points produced. The higher the time threshold, the fewer the points detected, since it is unlikely that a taxi will stop for such long periods during a journey. Consequently, a time threshold of 3 minutes should indicate areas of importance on a typical taxi route. This time threshold will ignore small hindrances such as a traffic lights, but will include more significant delays such as heavy congestion. These areas of congestion are of importance to us as they indicate areas of the city that are either popular destinations themselves, or roads that lead to hotspots and attractions within the city. With regard to the distance threshold, we must take into account the potential inaccuracies of a GPS tracer. A distance threshold of 50 metres would appear reasonable, since a taxi travelling further than 50 metres would indeed appear to have moved. These threshold values are entirely heuristic in nature. Due to the fact that each taxi log contains an unequal amount of data at varying time intervals, running the stay point detection on each taxi log, generates a different number of stay points. It is possible that the algorithm will detect no stay points for some taxis due to the fact that the original log is either empty, or only contains a handful of entries. Furthermore, there are logs that only contain identical entries in terms of longitude and latitude, although somewhat negligible, they will be detected and generated as stay points, since it is classed as a type 1 stay point illustrated in Figure 3. The histogram in 7 shows the distribution of stay points generated for each taxi log per day.

The stay point algorithm reduces original data-set of over 10 million points to 1322674 points, which is a drastic reduction. Although upon further inspection of the data-set, it is somewhat expected as we see that in many of the logs, the same points are frequently repeated. As a result, these types of points will be grouped together as stay points, as they are considered type 1 stay points. Running the stay point detection algorithm on the whole data-set takes around 30 minutes ($T_t = 3,\ T_d = 50$).

Although we seem to detect a large number of stay points, we cannot be sure that they are indeed accurate depictions of when the taxi was truly stationary. Firstly, due to the fact that the GPS tracer was taken at varied intervals we cannot be sure that these stop points are entirely accurate. Secondly, within the stay points detected, we don't know the exact movements. Looking at Figure 8, it is possible that the driver spent the majority of the time within a certain area between two points within a trajectory. The driver could have spent most of the time within the dashed red circle, but we don't know, due to the fact that this method of detection does not take into account the sampling rate. Additionally, the dataset provided does not contain the taxi meter data. When a taxi picks up a passenger, the driver turns on the meter and when the customer is dropped off it is stopped. The meter is used to calculate the journeys fare. If the dataset included the times at which this meter was started, stopped and reset we could segment the trajectories into more accurate journeys and identify key geographical areas this way.

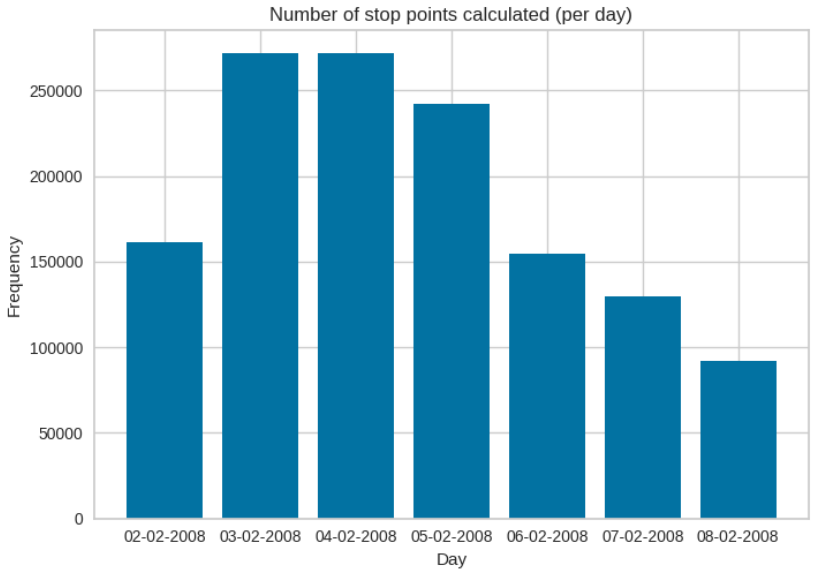


Figure 7: Number of stay points calculated

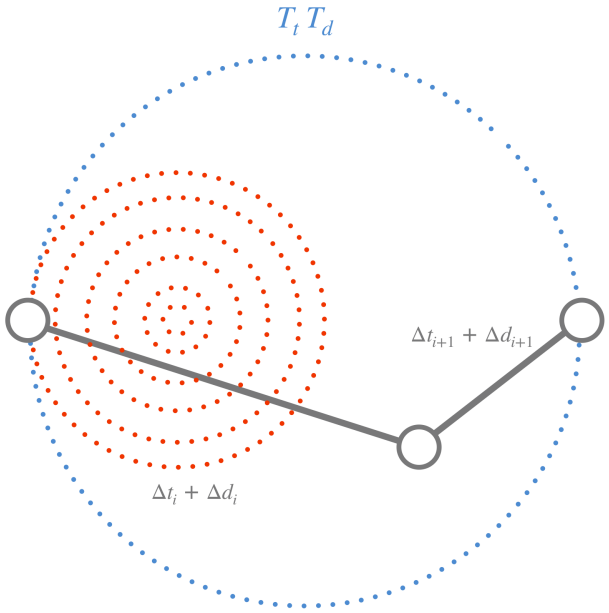


Figure 8: Inaccuracies in stay (stop) point detection

5.2 Clustering

Using the automatic approach declared previously, we can obtain a threshold value for our CFT. We have our stay points separated based on the day of the week so that the clusters produced are more applicable to reality. Consequently we create seven CFTs, one for each day of the week and so need seven different threshold values.

For our K-Means threshold approximation, in order to find the best fitting value for K, we take 100 samples and calculate the elbow value for each random sub-sample we took. On this data-set running more samples did not change the value of K chosen or even have a significant effect on the average radius calculated, therefore we keep it at 100 random samples. The value of K returned did fluctuate slightly depending on the day, however $K = 2$ seemed to be the value that was most commonly returned. If we were to run the approach on another data-set, it would be a good idea to increase the number of sub-samples we take as this may improve accuracy.

Running the sub-sampling method we obtain different thresholds for each day. Note that each time we run this method, it will produce varying thresholds. This is due to the fact that different random sub-samples are taken as well as the nature of a clustering approach. Clustering is NP-hard, thus each time we cluster we find one of the numerous local optimum values. Meaning that, we cannot guarantee that this method will produce the same average radius value each time it is run, even if by chance the same random sample is chosen. Below are the calculated threshold values.

```
Thresholds: [0.11189519451951799, 0.11480419902764678, 0.08365315634878759,
0.11967915777197739, 0.20216968155705575, 1.5380923627103122, 0.23554024364460485]
```

Once the thresholds have been calculated we create the cluster feature tree for each day using the corresponding threshold. Each cluster feature tree created uses an order of 50. This means that the tree produced is shallow, but wide, since 50 cluster feature entries are allowed per node. Choosing a high value like this means that the clusters produced are more of an overview at a high geographical level, rather than a zoomed in look at a low level as described in Figure 6a.

The results we obtain are shown in Figure 10. The circles represent the area covered by a cluster (bus stop) produced by the CFT, thus indicate the maximum distance a client assigned to that bus stop has to walk to get to it. Dark red circles indicate clusters with a comparatively large number of points, thus indicate that a bus stop placed here would be heavily used based on the data it was given. The lighter the colour, the fewer points the cluster contains.

Clearly these stops produced aren't very convenient, but why is this the case? To understand why, we need to look at the details of the clusters produced. Below shows the detailed output of the clusters produced on 04-02-2008, 06-02-2008 and 07-02-2008.

```
Date: 04-02-2008
Clusters:
   centroid_0  centroid_1         n    radius           size
0  117.102412   40.143065    1437.0  0.115382  330325.192309
0  116.395639   39.926882  238972.0  0.115528  331943.823753
```

```
Date: 06-02-2008
Clusters:
   centroid_0  centroid_1         n    radius          size
0  116.423132   39.935819  129532.0  0.513512  6.543422e+06
```

```
Date: 07-02-2008
Clusters:
   centroid_0  centroid_1        n    radius          size
0  116.399153   39.926878  84783.0  0.207086  1.066040e+06
0  117.082311   40.173079   2766.0  0.223930  1.243073e+06
0  116.549023   39.959083   1628.0  0.114425  3.255227e+05
```

We see that only a few clusters are produced. Furthermore on the 6th of February, only a single cluster is produced that engulfs the whole city. Like mentioned above, the algorithm is working at a relatively high geographical scale and is trying to accommodate points that are actually quite far away from each other, as a result it is quite possible that outliers still remain. These are not necessarily outliers that are inconsistent with a trajectory, but logs that contain the same repeated point at different time frames. Hence why our pre-processing step does not account for this. For instance, it is possible that the GPS tracker was left on while the taxi was not operating. Figure 10d and 10e appear to visualise this, as we see another cluster is created far outside the city centre which has a comparatively low amount of points (shown by the cream coloured circle). Additionally the cluster details above also appear to support this argument, shown by the significant majority of points falling within one single cluster.

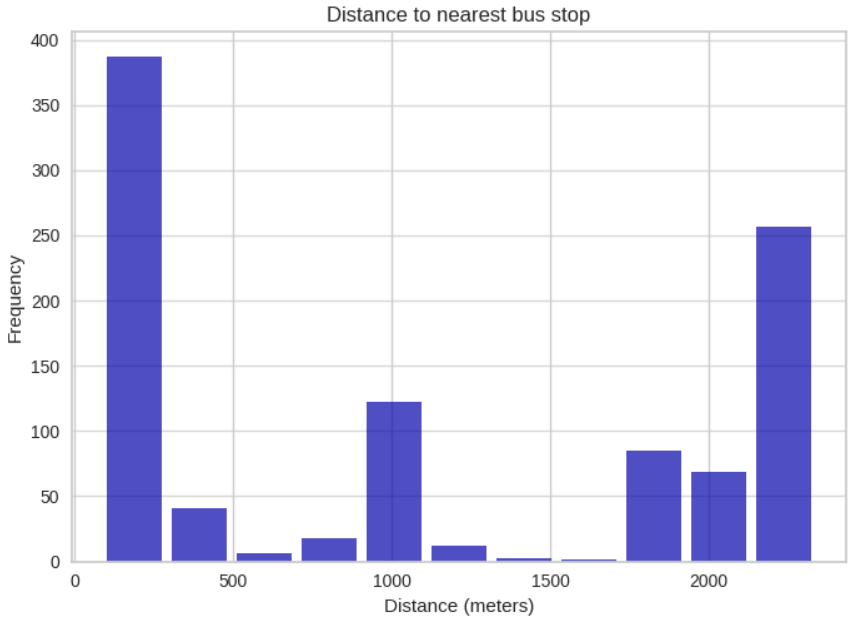
In the approach we illustrated how the threshold and branching factor dictate the final shape of the CFT. We can see from the results here that the tree produced here is far from what we are looking for. Clearly the threshold is way too high since customers are required to walk very far to reach their closest stop - shown by the huge circles. Furthermore, the geographical scale that the CFT is working at is too abstract to lead to useful results. Consequently, we want to encourage cluster splitting, which can be achieved by lowering the threshold and branching factor.

Doing so produces seven cluster feature trees of varying height, which appear to be much more practical. Figure 11 contains each one-week plot of the cluster centroids at the lowest layer in the tree, this layer in the tree contains the most refined clusters with regard to geographical scale. Looking at the figures produced for each day, within them we see that the majority of clusters are produced in the city centre with the exception of the airport (top right). Moreover, we see that on different days not only are different clusters produced but different regions are busier than others. For instance, on a Sunday, the airport seems to be the most visited location, compared to Wednesday where there were very few trips made - shown by the contrast in colours of the clusters produced. Although they visually appear to be more convenient, we should see what this actually means for the client. To do so, we can take a random sample of points and see how far they are from their nearest centroid, this is effectively how far a customer would have to walk to the bus stop. Figure 9 shows that by decreasing the threshold and branching factor we introduce cluster splitting, in turn creating more clusters so that the distance a client has to walk to their nearest stop is drastically reduced.

As shown, the introduction of a smaller branching factor and reduced threshold increases cluster splitting and can be seen in the plots. Additionally this has increased cluster overlap, which is represented by overlapping points in the plots. Again, this makes sense since we have greatly reduced the order of the tree. Furthermore this tells us that it would not be wise to further decrease the order of the cluster feature tree, since it would increase unnecessary cluster splitting, that we are already starting to see in some areas of the plots.

Finally, if we inspect the CFT as it is built, the clusters that are created within the first few hundred points are those that lead to the final clusters. Subsequently, after only a few seconds of running the algorithm and after only a small portion of data-points inserted we start to see solid results which only improve in accuracy the more points we insert. This is a huge advantage over other methods, and shows that using a CFT can quickly lead us to decent results as long as the threshold and branching factor values are chosen correctly.

(a) Using the automatic threshold detection - Date 03-02-2008 (Sunday)



(b) Using the manually input threshold 0.005 - Date 03-02-2008 (Sunday)

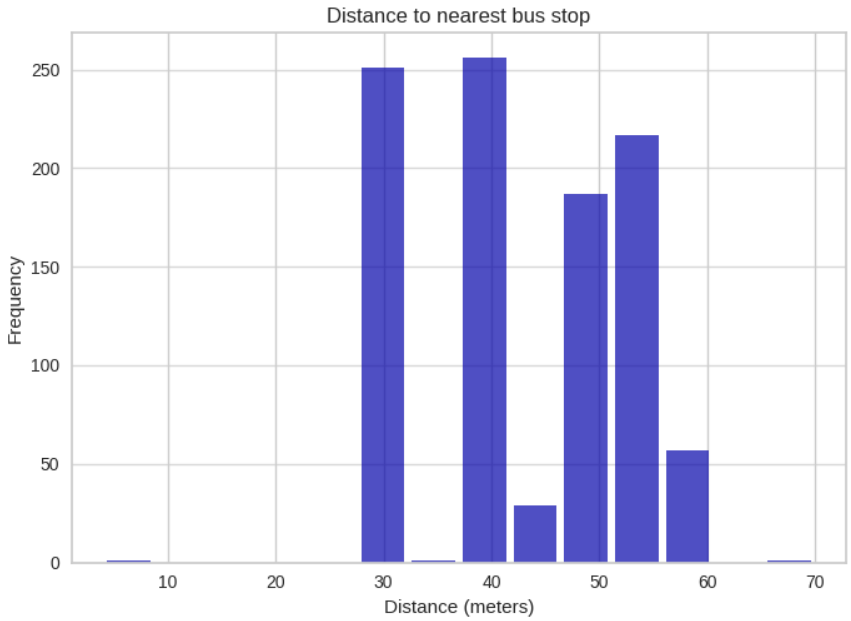


Figure 9: Random sample of 1000 points showing the distance to the nearest stop

6 Conclusion

What we have presented here, is a clustering approach that identifies similarities between trajectories and further uses these similarities to find an accurate placement of bus stops based on key geographical regions within the data-set. We have shown the affects of choosing one clustering technique over another and how that alters the clusters produced. Consequently, it was illustrated and argued why this clustering approach is well suited to not only trajectory data-sets in general, but large trajectory data-sets with millions of points.

From the plots produced, we can see visually how much of an influence the threshold and branching factor have on the clusters created. Unfortunately, this automatic approach itself did not yield lucrative results, but why was this the case? There are potential numerous reasons but the main ones are likely down to a a couple of factors. First, if the sub-sample taken does not follow the same distribution as that of the population, it is not representative of the population and so may lead to inaccurate radius values. For instance, if we knew in advance that the data-set did indeed follow a Gaussian distribution then the random sub-sampling results would likely be valid. It is also possible that there are still outliers in the data-set that were not removed in the pre-processing step. Most importantly though, is that these threshold and branching factors are dependent on the data-set. Looking at the distribution of points in the clusters produced automatically, there was always one cluster that contained significantly more points than the others produced. This leads me to believe that outliers or possibly insignificant points were still present in the data-set when clustering was applied. Although it would be very convenient to have an automatic approach that does not require an analyst, in order to obtain solid results that are indeed useful, input from such an analyst is required here. To briefly summarise, what leads to accurate results, is finding the Goldilocks zone for both the threshold and branching factor values.

Although the approach may not be automatic, that is, one without human interaction, it is still scalable and can be applied in an online fashion. As a result, if implemented as an online clustering approach, one can see the results in real time after only a few hundred data points have been inserted into the tree, which sets it apart from the majority of other clustering techniques used in the context of trajectory data-sets.

Additionally, we have shown that judging the similarity between trajectories is subjective and non-binary. From the GPS trajectories, we cannot say for definite that two trajectories are identical, hence similarity is a spectrum. This therefore makes a clustering approach well suited to this application, as clusters of high similarity are created and grouped together.

7 Future Work

Evaluating what has been achieved in this research project, there are numerous further improvements that can be made. There are certain aspects of the approach that can be refined, such as the CF Tree itself, allowing the us to improve its efficiency. Additionally, we can further develop the project, extending its application into other related fields. Consequently, listed below, are different extensions and adaptions we could make to the project described here.

Sorting clusters using a K-D Tree: When we insert a point into the Cluster Feature Tree we must find the closest cluster to that point, at each layer, until we find the closet entry in a leaf node. This requires us to check all centres on the path against the incoming point at each layer. If we can sort the cluster centres then we can reduce the amount of operations needed.

Extending the problem to an optimisation problem: The problem can now be transformed into an optimisation problem, using the stops produced as input into the optimisation problem. For example, we could minimise the number of buses needed to service the routes defined by the created bus stops.

Improved automatic threshold test: We could try and fit the data to a certain distribution. Depending on whether or not the data follows a certain distribution (i.e. Gaussian) we could use a different approach to obtaining the threshold value. Additionally instead of looking for the average radius, using the minimum radius may actually lead to better results. Furthermore, we could incorporate the distance between cluster centres into the prediction, as this also has an effect on the clusters produced.

Incorporating the sample rate into stay point detection: As mentioned, the current stay point detection algorithm does not take into account the sampling rate of the trajectory set. This means that in between the points recorded, we do not know the actual movements of the vehicle. Using a model that takes into account this sampling rate may improve the accuracy of these points created.

Map matching: Mapping points within a trajectory to a certain road would be useful here. This would give us a better understanding of each vehicles trajectory as we would have more information on the exact route that was taken through the city. When applying clustering here, we could identify key roads which would make it easier to identify key areas located on the identified roads.

Identify similar sequences: After clustering, we can query the tree to find the order in which a taxi visited the clusters. This could be done for each taxi, identifying a sequence of visited clusters for each taxi in the set. Then we could further cluster the sequences themselves, or create a graph representing the flow of traffic more accurately.

There are plenty of avenues to explore here and many different trajectories that future research can take.

References

[1] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," p. 316–324, 2011.

[2] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: Driving directions based on taxi trajectories," p. 99–108, 2010.

[3] J. A. Long and T. A. Nelson, "A review of quantitative methods for movement data," *International Journal of Geographical Information Science*, vol. 27, no. 2, pp. 292–318, 2013.

[4] M. Nanni and D. Pedreschi, "Time-focused clustering of trajectories of moving objects," *J. Intell. Inf. Syst.*, vol. 27, pp. 267–289, 11 2006.

[5] F. Giannotti, M. Nanni, D. Pedreschi, F. Pinelli, C. Renso, S. Rinzivillo, and R. Trasarti, "Unveiling the complexity of human mobility by querying and mining massive trajectory data," *The VLDB Journal*, vol. 20, no. 5, p. 695, 2011.

[6] G. Andrienko, N. Andrienko, S. Rinzivillo, M. Nanni, D. Pedreschi, and F. Giannotti, "Interactive visual clustering of large collections of trajectories," *2009 IEEE Symposium on Visual Analytics Science and Technology*, pp. 3–10, 2009.

[7] P. S. Castro, D. Zhang, C. Chen, S. Li, and G. Pan, "From taxi gps traces to social and community dynamics: A survey," *ACM Comput. Surv.*, vol. 46, Dec. 2013.

[8] D. Ashbrook and T. Starner, "Using gps to learn significant locations and predict movement across multiple users," *Personal Ubiquitous Comput.*, vol. 7, p. 275–286, Oct. 2003.

[9] L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. F. de Macedo, B. Moelans, and A. Vaisman, "A model for enriching trajectories with semantic geographical information," in *Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems*, GIS '07, (New York, NY, USA), Association for Computing Machinery, 2007.

[10] A. T. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares, "A clustering-based approach for discovering interesting places in trajectories," in *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, (New York, NY, USA), p. 863–868, Association for Computing Machinery, 2008.

[11] Y. Zheng, "Trajectory data mining: An overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, May 2015.

[12] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "On clustering validation techniques," *Journal of Intelligent Information Systems*, vol. 17, no. 2, pp. 107–145, 2001.

[13] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," *SIGMOD Rec.*, vol. 25, p. 103–114, June 1996.

[14] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," p. 49–60, 1999.

[15] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," p. 226–231, 1996.

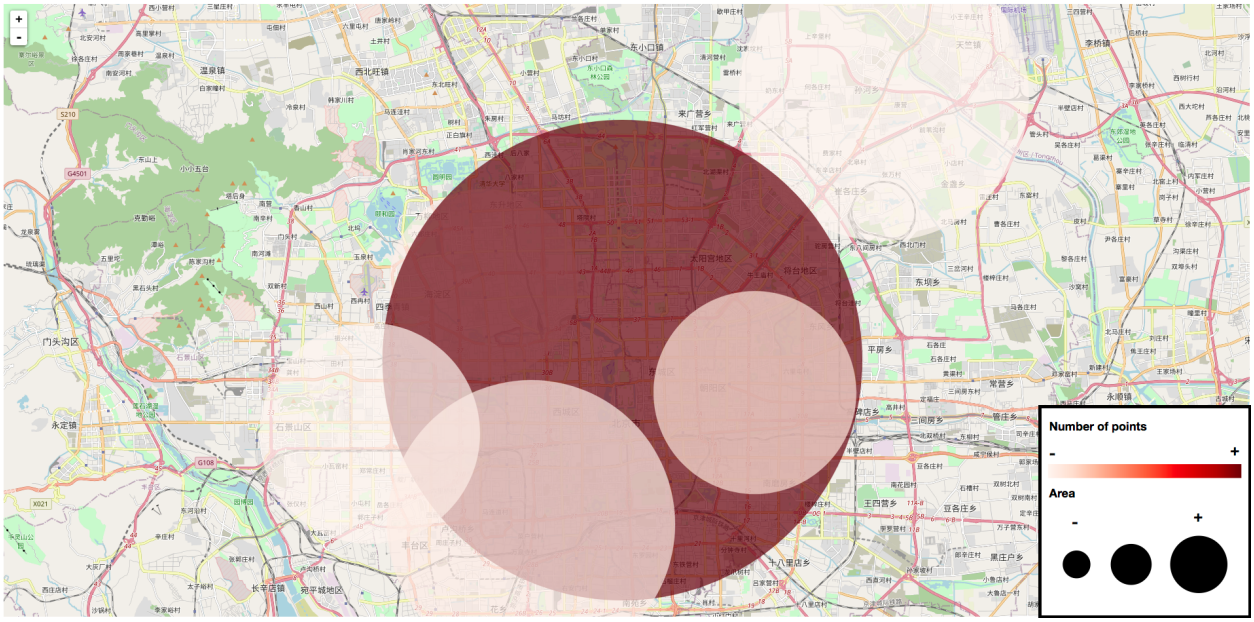
[16] W. R. Tobler, "A computer movie simulating urban growth in the detroit region," *Economic Geography*, vol. 46, pp. 234–240, 1970.

[17] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma, "Mining user similarity based on location history," in *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '08, (New York, NY, USA), Association for Computing Machinery, 2008.

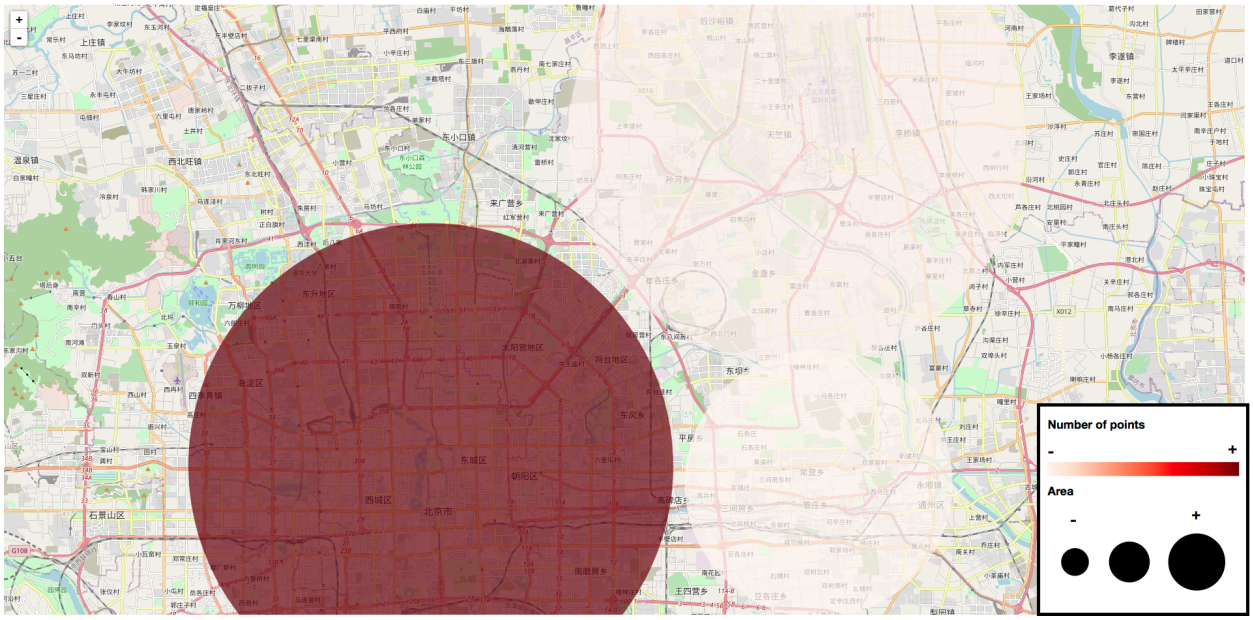
[18] B. Lorbeer, A. Kosareva, B. Deva, D. Softić, P. Ruppel, and A. Küpper, "A-birch: Automatic threshold estimation for the birch clustering algorithm," pp. 169–178, 2017.

8 Appendix

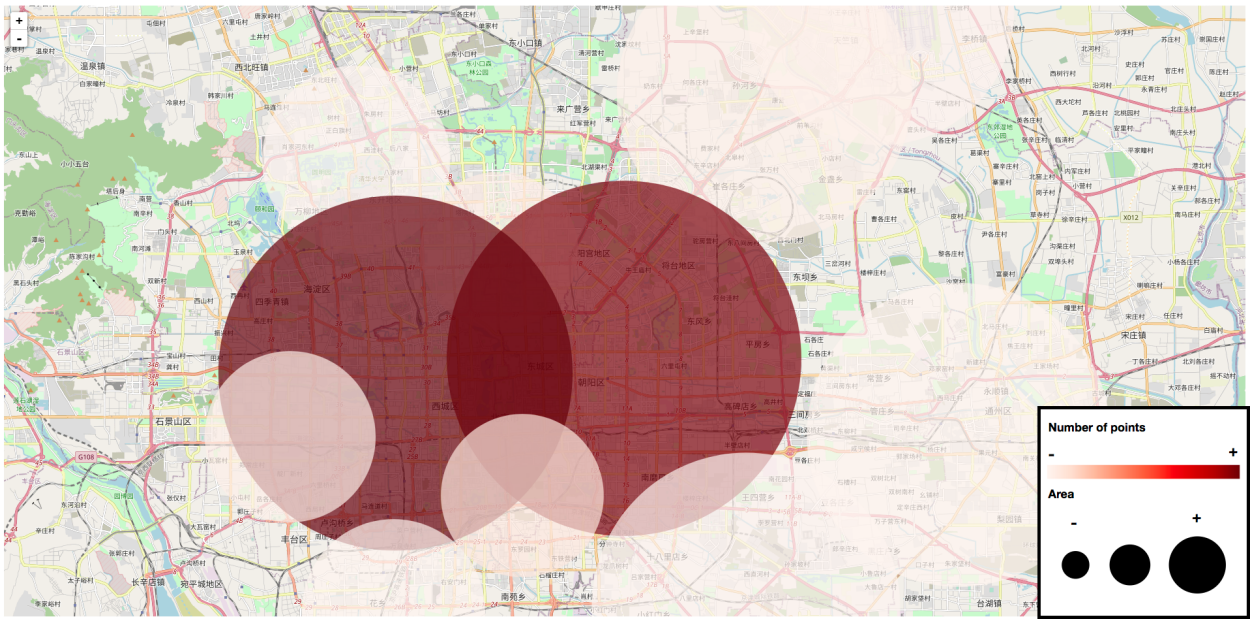
(a) Date 02-02-2008 (Saturday) Order = 50 Threshold = 0.11189519451951799



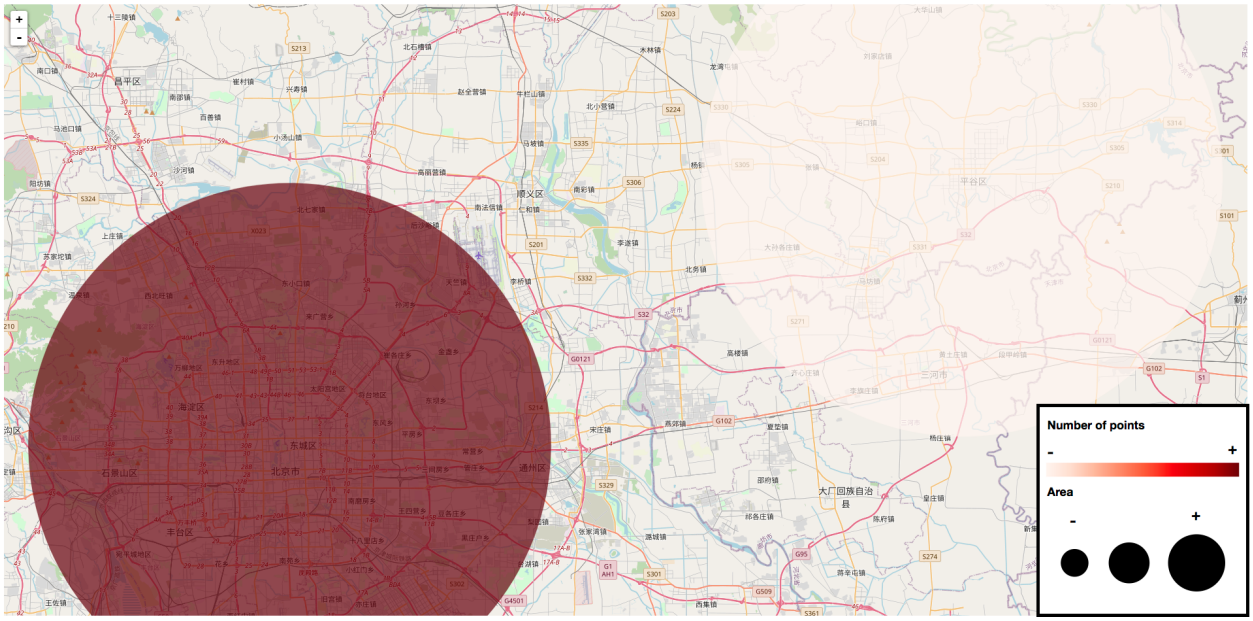
(b) Date 03-02-2008 (Sunday) Order = 50 Threshold = 0.11480419902764678



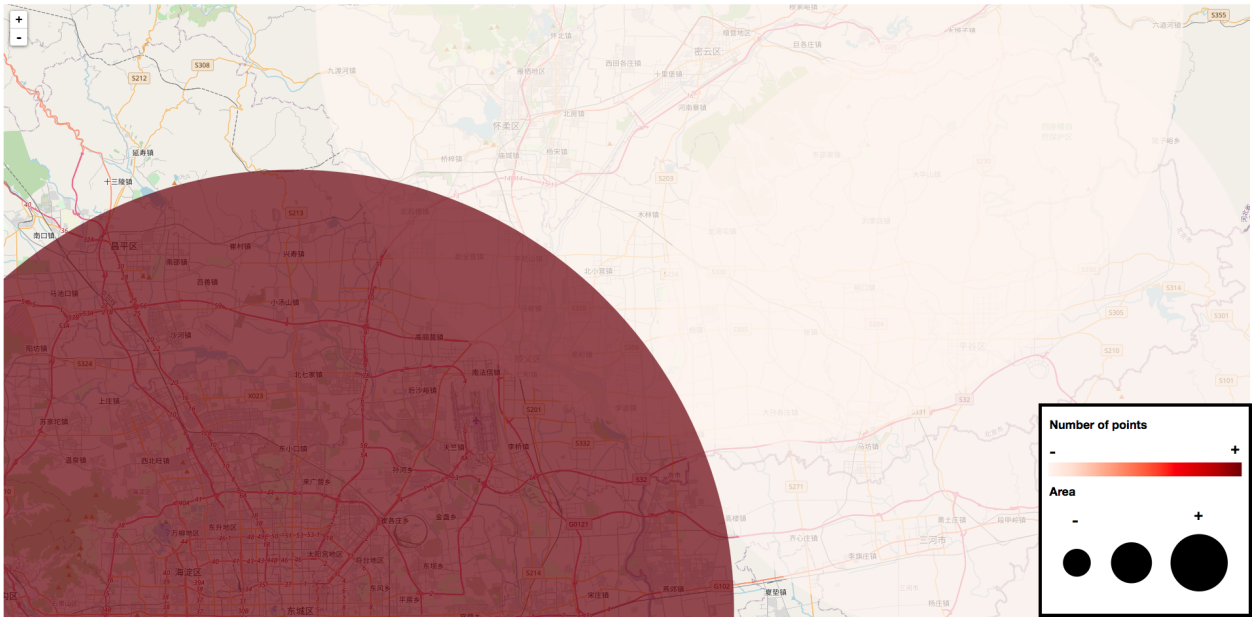
(c) Date 04-02-2008 (Monday) Order = 50 Threshold = 0.08365315634878759



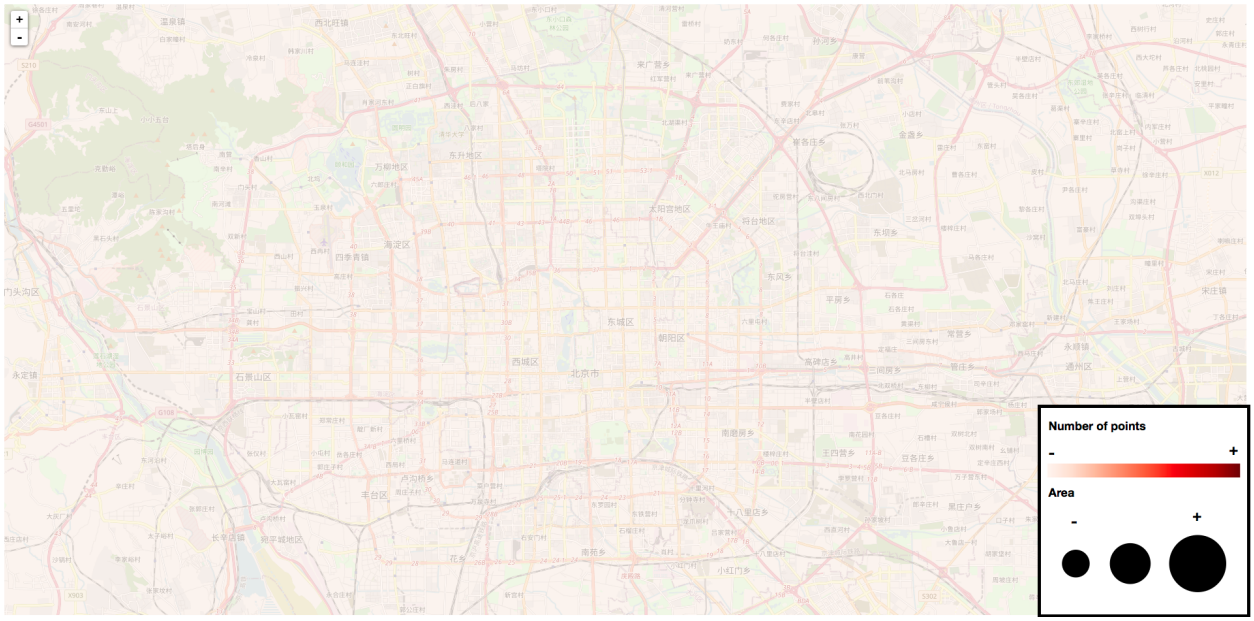
(d) Date 05-02-2008 (Tuesday) Order = 50 Threshold = 0.11967915777197739



(e) Date 06-02-2008 (Wednesday) Order = 50 Threshold = 0.20216968155705575



(f) Date 07-02-2008 (Thursday) Order = 50 Threshold = 1.5380923627103122



(g) Date 08-02-2008 (Friday) Order = 50 Threshold = 0.23554024364460485

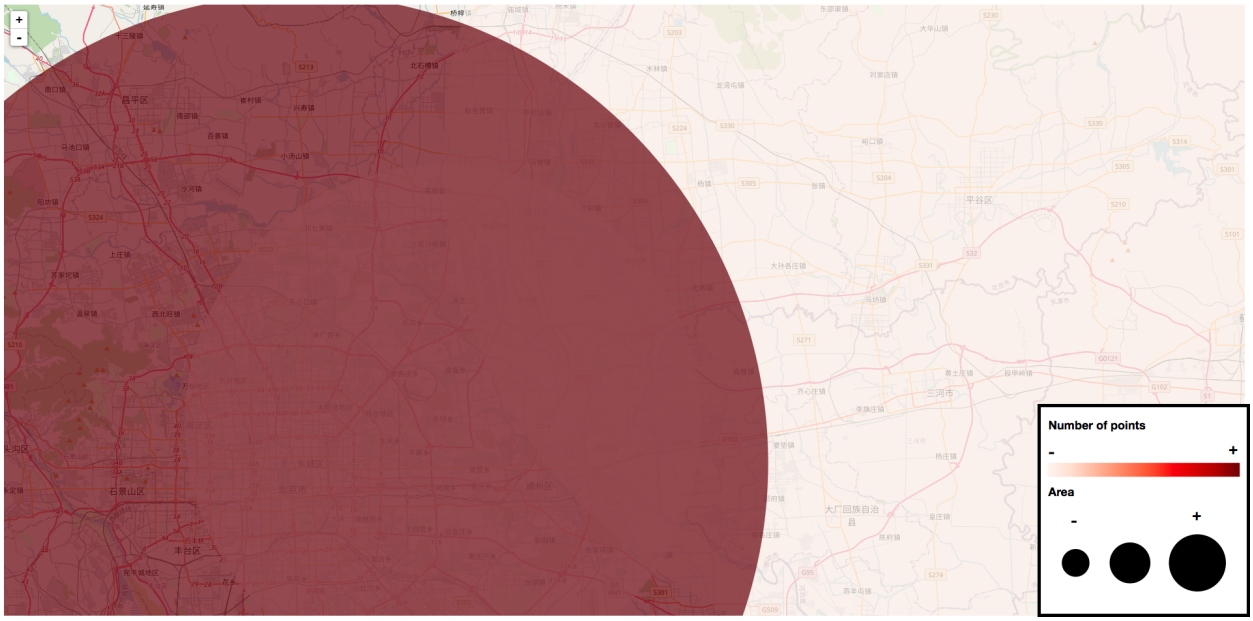
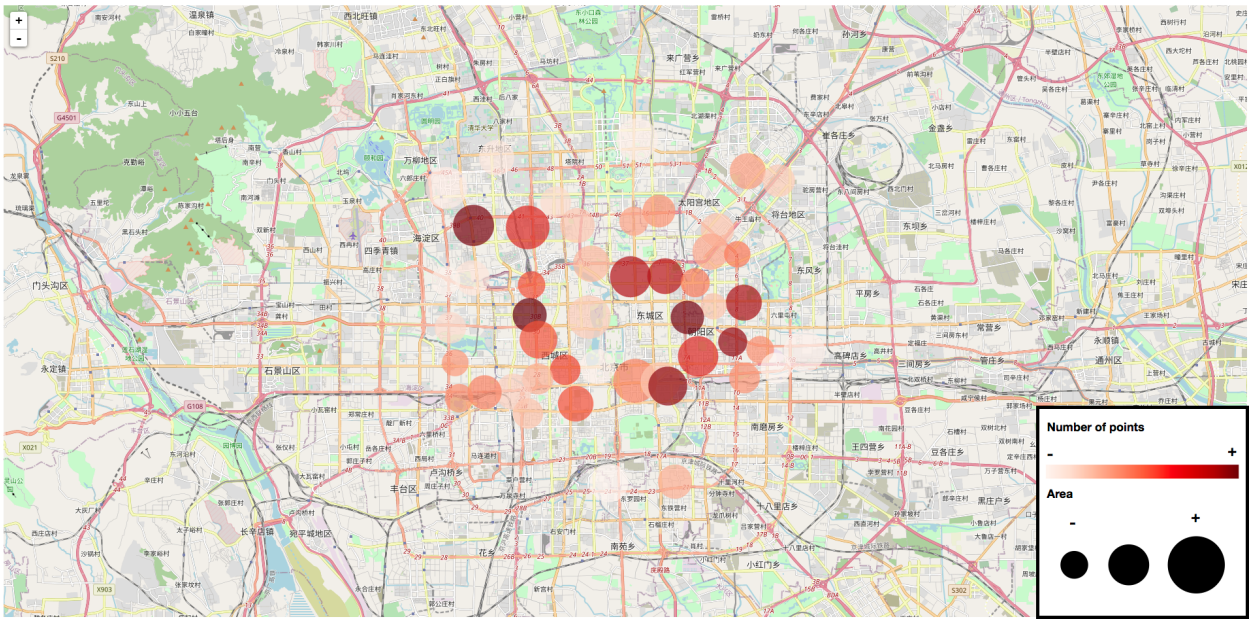
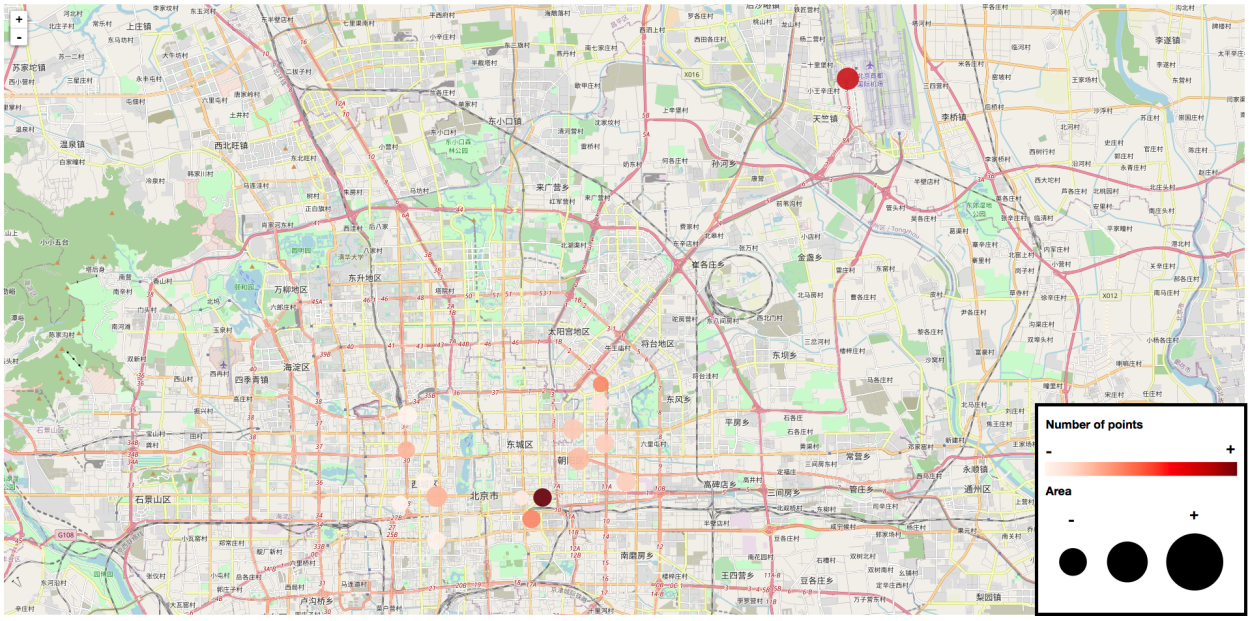


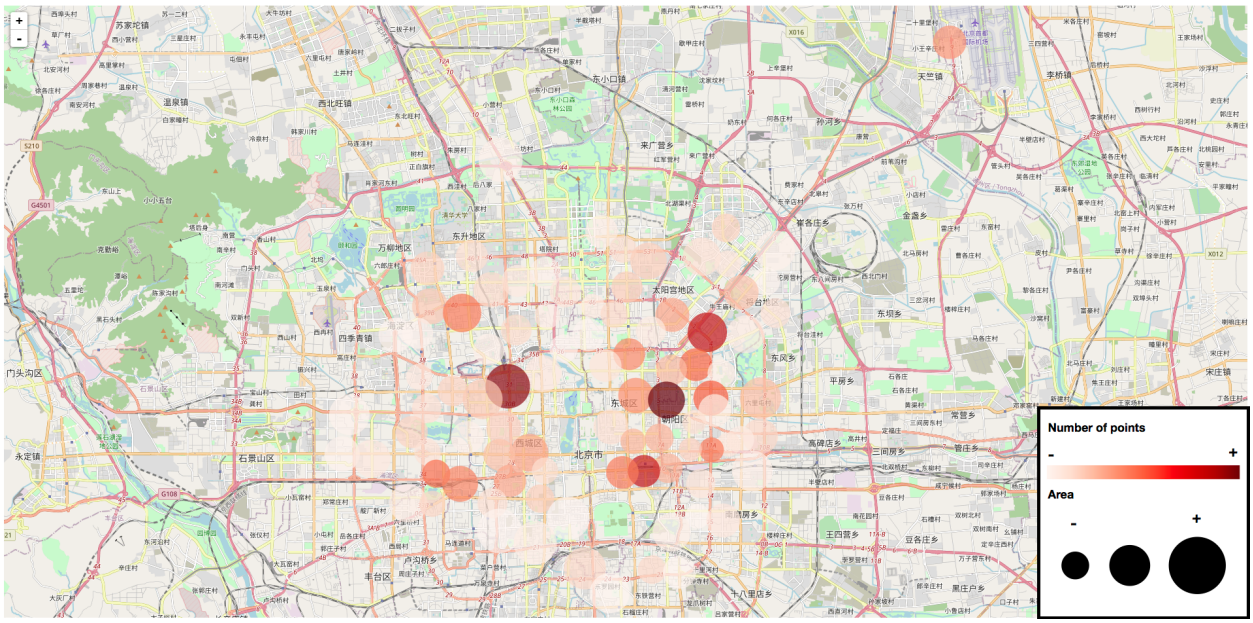
Figure 10: The plots showing the clusters produced per day.



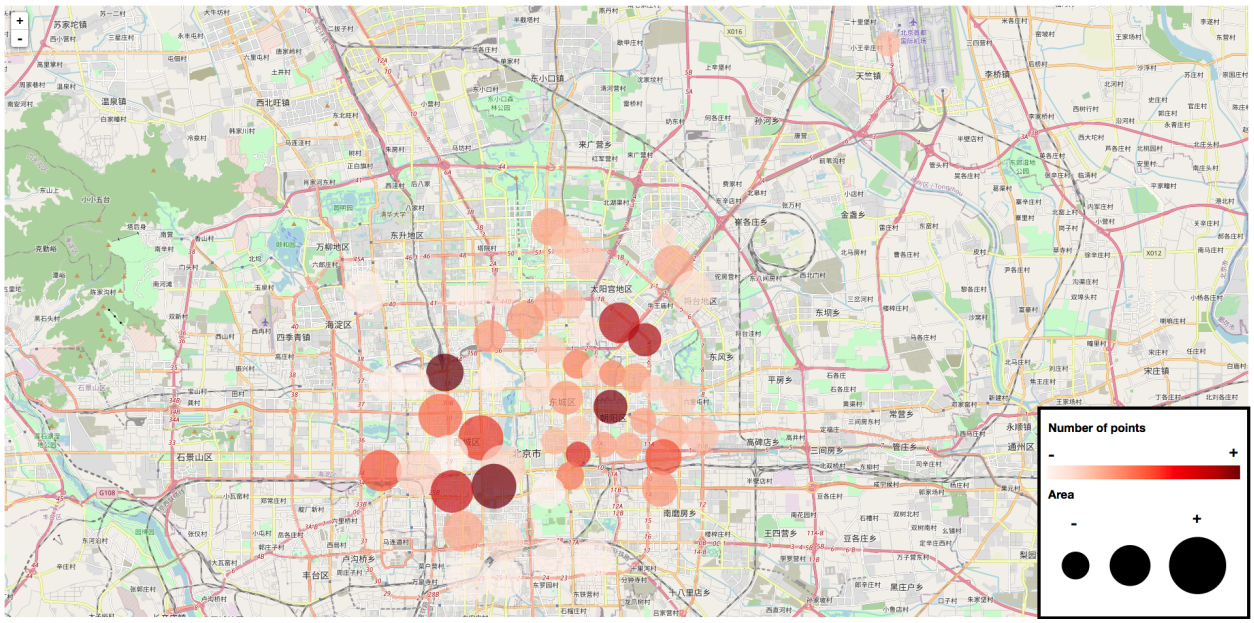
(a) Date 02-02-2008 (Saturday) Order = 10 Threshold = 0.01

(b) Date 03-02-2008 (Sunday) Order = 10 Threshold = 0.01

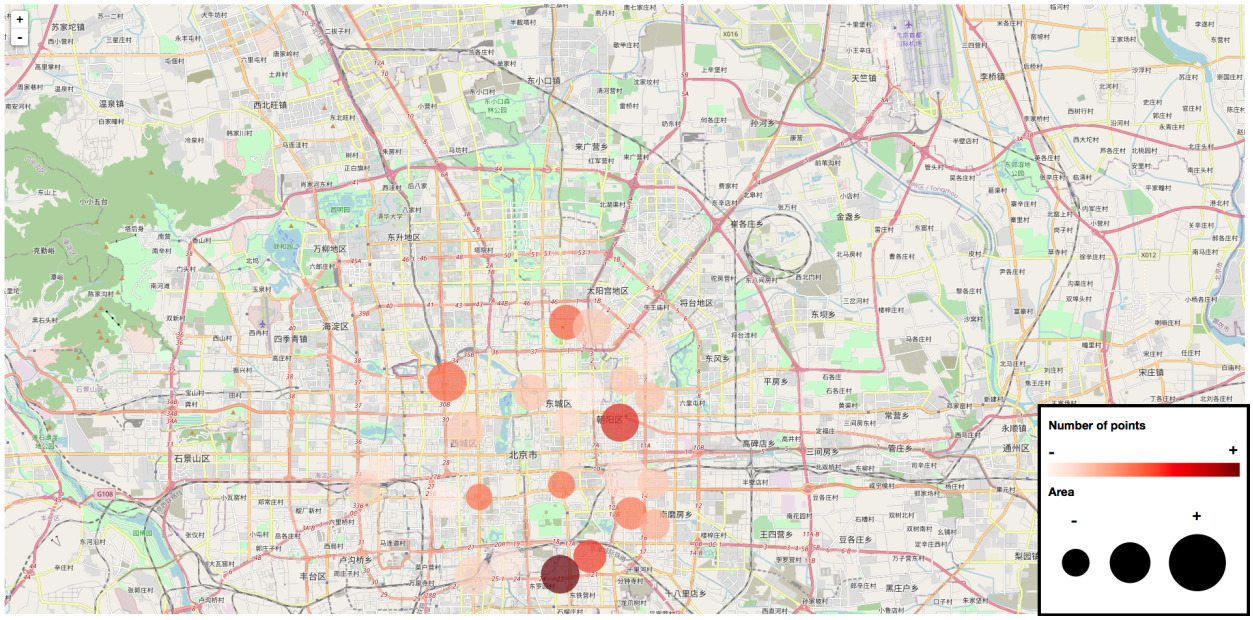




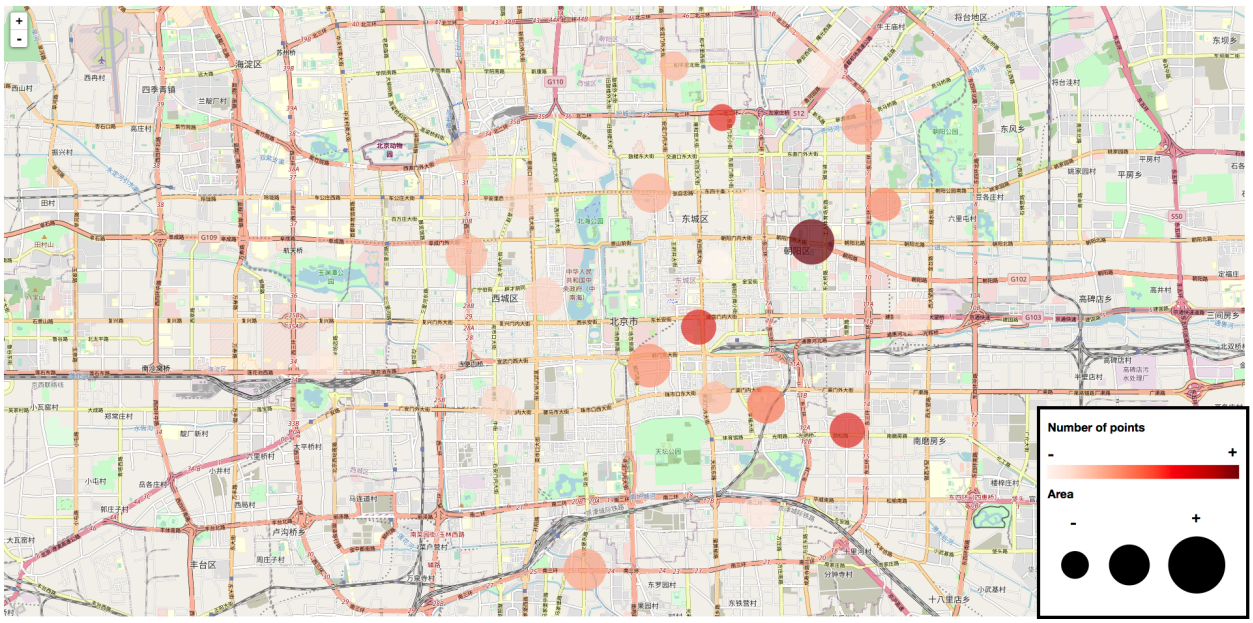
(c) Date 04-02-2008 (Monday) Order = 10 Threshold = 0.005



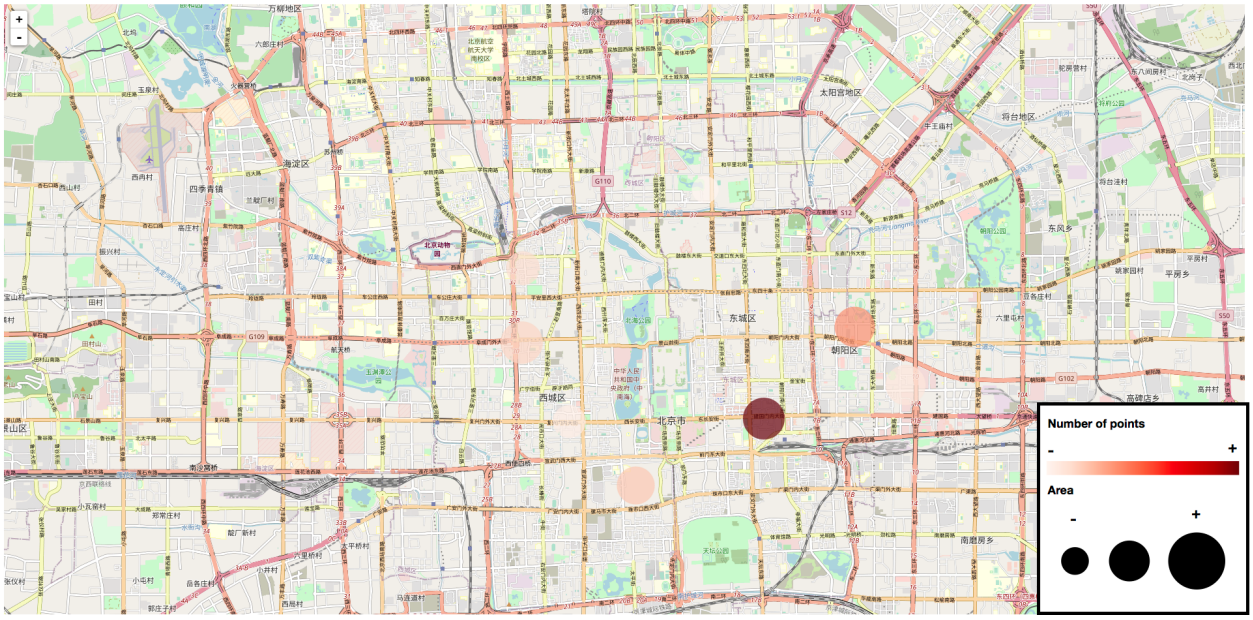
(d) Date 05-02-2008 (Tuesday) Order = 10 Threshold = 0.01



(e) Date 06-02-2008 (Wednesday) Order = 10 Threshold = 0.01



(f) Date 07-02-2008 (Thursday) Order = 10 Threshold = 0.01



(g) Date 08-02-2008 (Friday) Order = 10 Threshold = 0.01

Figure 11: The plots showing the clusters produced per day.