1.1 Motivations for Mining Taxi GPS Trajectories

Taxi drivers or any experienced car drivers, more often than not, possess some *implicit* knowledge or intuitions about which route from a source u to a destination v is the best in terms of travel time at a particular moment. Such knowledge or intuitions stem from everyday experiences. For example, a taxi driver may observe that there are always traffic jams from 6 p.m. to 7 p.m. on a particular street and hence avoid travelling on that street during that period of time whenever possible. But observations of this kind, albeit valuable, are too subtle to be captured by any general algorithms and oftentimes, even drivers themselves may not be fully aware of that.

However, mining their GPS trajectories can be an excellent way of revealing such knowledge. In a metropolis such as Beijing or New York, taxi drivers are required by regulations to install GPS devices on their cars and to send time-stamped GPS information to a central reporting agency periodically, for management and security reasons. The GPS information typically includes latitudes, longitudes, instantaneous speeds and heading directions. Therefore, such GPS data is readily available and little effort is needed to collect it. By means of mapping to a real road network all GPS data points pertaining to a particular taxi during a specific period of time, a GPS trajectory can be obtained to represent the driver's intelligence. Definition 3 formally defines taxi trajectories.

Definition 3 (*Taxi Trajectory*)**.** A taxi trajectory $T = (p_1, p_2, \ldots, p_n)$ is a set of chronologically ordered GPS data points pertaining to one taxi. The time span between p_1 and p_n is defined as the *duration* of the trajectory.

1.2 Practical Limitations

Some practical limitations are worth mentioning.

Arbitrary sources and destinations

In a typical map-query use case, a user is able to select an arbitrary source to start with and an arbitrary destination to go to. But this may not be possible for the approach proposed in this project, since the taxi GPS trajectories do not necessarily cover every part of a city's road network. It is likely that there are no trajectories passing through one particular source and one particular destination.

Low sampling rate

Taxis report their locations to the central reporting agency at a relatively low frequency to conserve energy. For the data set used in this project, the expected sampling interval is one minute. But oftentimes, the GPS device may not be working properly or may be occasionally shut down due to various reasons, which causes the actual sampling interval to fluctuate.

Even if the sampling interval *is* strictly kept at one minute, for a taxi moving at a typical speed of $60km/h$, it means the distance between two consecutive sample points is $1km$. Such a large distance increases the uncertainty about the *exact* trajectory that the taxi has moved along. Figure 1-1 [15] demonstrates a problem caused by low sampling rate and long inter-sample-point distance.

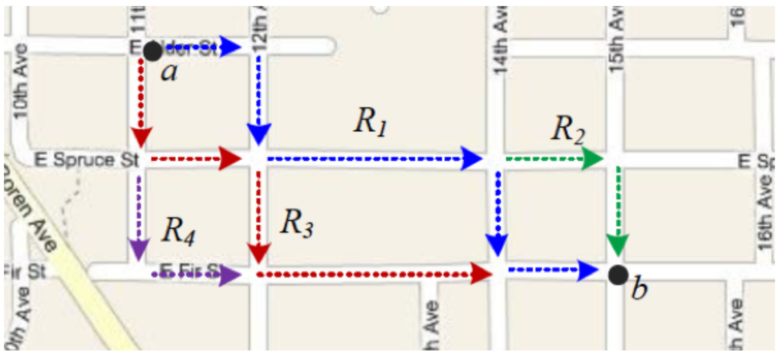


Figure 1-1: An example of low-sampling-rate problem

offer relatively stable and fast reverse geocoding services. However, due to the "China GPS shift problem" [13] where longitudes and latitudes encoded in WGS-84 format are required by regulations to be shifted by some variable amounts when displayed on a street map, Google Maps is not able to display a GPS data point correctly because it only supports WGS-84 formats. Figure 2-1 illustrates the effect of such shift, where the correct location is displayed on the **right**.

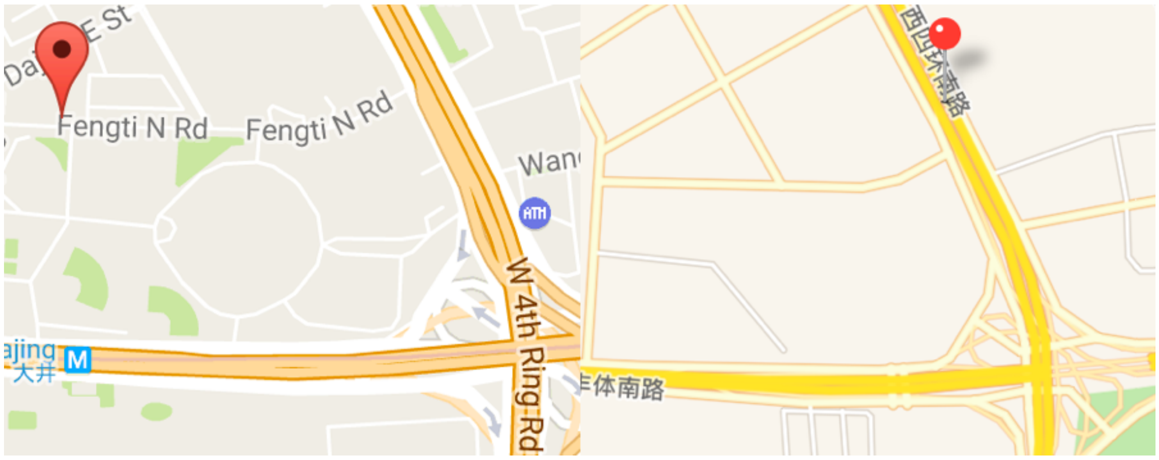


Figure 2-1: An example of China GPS shift problem

Baidu Maps, on the other hand, has been using their own coordinate system, BD-09, which is an improved version of the Chinese official coordinate system, GCJ-02. Baidu Maps does provide a set of APIs to convert WGS-84 coordinates into BD-09 ones. To reverse-geocode GPS data points, longitudes and latitudes must first be converted into BD-09 format. Then another set of APIs from Baidu Maps can be used for reverse geocoding. To store the converted longitudes and latitudes as well as the street names obtained from reverse geocoding, four new fields were added to the original data set as shown in Table 2.2.

In order to use Baidu Maps APIs for coordinate conversion, the following system architecture was set up as shown in Figure 2-2. The Apache HTTP server hides the MySQL database and sends HTTP POST requests containing the original coordinates

Field	**Explanation**
DataUnitID	Nominal primary key for each record
UTC	UNIX_EPOCH in human readable format
BD09_LONG	Longitude encoded in BD-09 format
BD09_LAT	Latitude encoded in BD-09 format
STREET	Street name

Table 2.2: A summary of additional fields

to Baidu Maps Web API to get the converted coordinates. Then it updates the database through PHP *mysqli* utility.

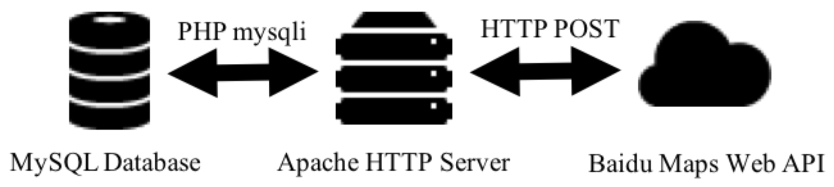


Figure 2-2: System architecture for coordinate conversion

After longitudes and latitudes were converted from WGS-84 format to BD-09 format, Baidu Maps Web API was used to reverse geocode all GPS data points. However, the system architecture was slightly changed, to accommodate the change in technology used. For reverse geocoding, AJAX[4] was used to communicate with the Baidu Maps Web API for speed and unlimited number of requests per day. Therefore, one additional layer was added to the existing system architecture as shown in Figure 2-3.

Executed in a web browser environment, AJAX sent HTTP POST requests to the Apache HTTP server to fetch the converted coordinates in BD-09 format which were

[4]Asynchronous JavaScript and XML

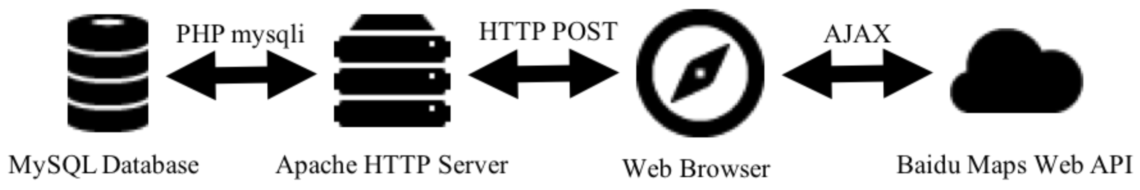


Figure 2-3: System architecture for reverse geocoding

subsequently sent to the Baidu Maps Web API server *asynchronously* via HTTP GET requests. Once the server responded with the name of the street, AJAX updated the database by sending another HTTP POST request to the Apache HTTP server. The asynchronous nature of this architecture, however, caused a few problems which are addressed in Section 2.3.

2.3 Outlier Detection

2.3.1 Motivations for Outlier Detection

The Baidu Maps Web API for reverse geocoding is stable and fast, but does not produce no errors. Sometimes, a GPS data point may be mapped to a main road but actually it is on the side road, which is one of the limitations mentioned in Section 1.2, or it is actually mapped to a street that Baidu Maps does not recognise. In neither case will Baidu Maps produce a correct reverse geocoding. Moreover, the reverse geocoding process is asynchronous, which means that it is being performed in the background in parallel with the main application thread. Therefore, it is inevitable that some street names may get lost when the records are being updated or a record is updated with a wrong street name. Figure 2-4 shows a drastic example.

In this example, Baidu Maps believes that all data points plotted belong to a particular street. But when plotted on a 2-D plane, these data points almost represent

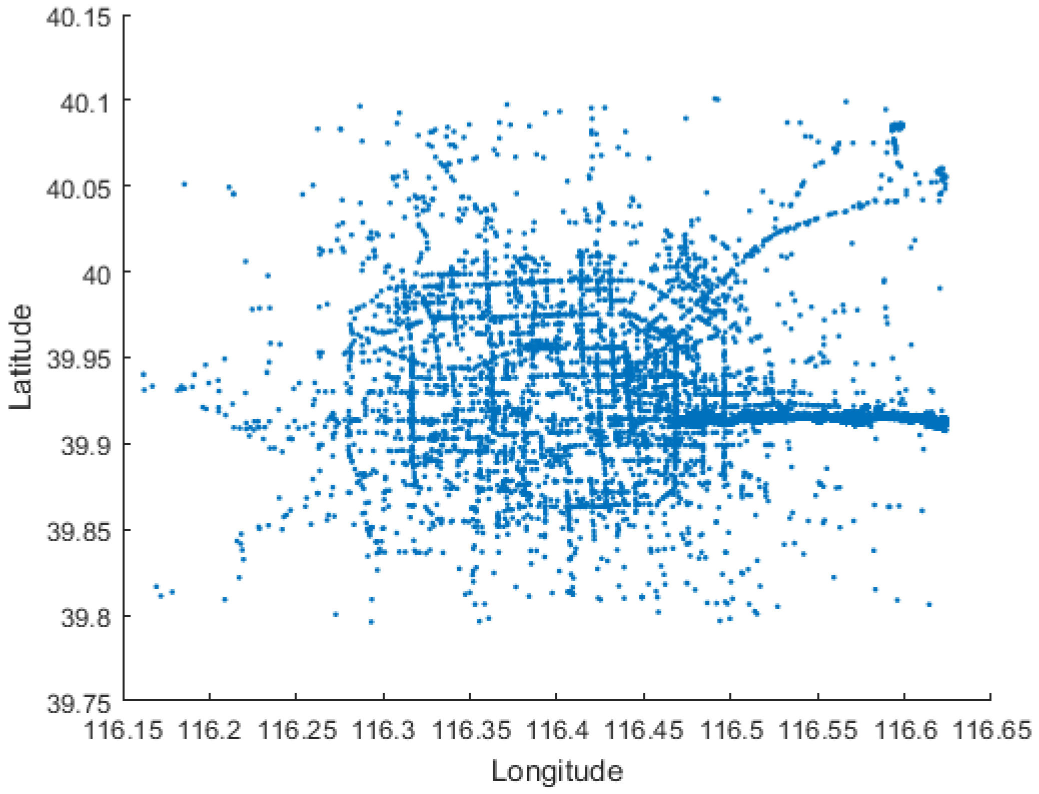


Figure 2-4: An example of outliers

the *entire* road network in Beijing. The actual, correct street is represented in the figure as the *thickest* blue line on the right half of the figure with a longitude ranging from 116.45°E to 116.65°E. Erroneous records like those not on the thickest line are known as *outliers* and must be properly identified and removed. This project proposes a novel outlier detection approach based on unsupervised learning whose principle behind is based on Theorem 1.

Definition 4 (*Reasonable reverse geocoder*)**.** A reasonable reverse geocoder always gives its best matching from a GPS data point to a street whenever possible and has an accuracy more than 50%.

Theorem 1 (*Majority Clustering Theorem*)**.** If a *reasonable reverse geocoder* is used to reverse-geocode a set of GPS data points which are mapped to a particular street *in reality*, then, when plotted on a 2-D plane, majority (more than 50%) of the points must be clustered together to form a rough shape that is similar to the shape of the street that they are supposed to be mapped to.

Proof. Proof by contradiction. Assume, for the purpose of contradiction, majority (more than 50%) of the data points that are *indeed* located on the same street are scattered arbitrarily on a 2-D plane after being reverse-geocoded by a reasonable reverse geocoder. In particular, when plotted on a 2-D plane, majority of them *do not* form any similar shape to that of the street they are supposed to be mapped to. Then, the majority must have been erroneously mapped to some other streets because no single street covers the whole city's road network. Thus, the reasonable reverse geocoder has only achieved an accuracy that is less than 50%, which contradicts Definition 4 of a reasonable reverse geocoder. □

2.3.2 Outlier Identification

Apparently, Baidu Maps provides a reasonable reverse geocoder because it is of industrial-grade quality and has an accuracy larger than 50%. Therefore, if a set of points belong to a particular street, after reverse-geocoded by Baidu Maps, majority of them should be clustered to assume a rough shape of that street according to Theorem 1. Based on that, an unsupervised learning technique — clustering can be used to separate the correctly mapped data points from the outliers.

Many clustering techniques are available [10]. Since each record can be represented graphically as a point on a 2-D plane with longitude as the x axis and latitude as the y axis, a **self-organising feature map** (SOFM) [6] seems to be an appropriate technique to use.

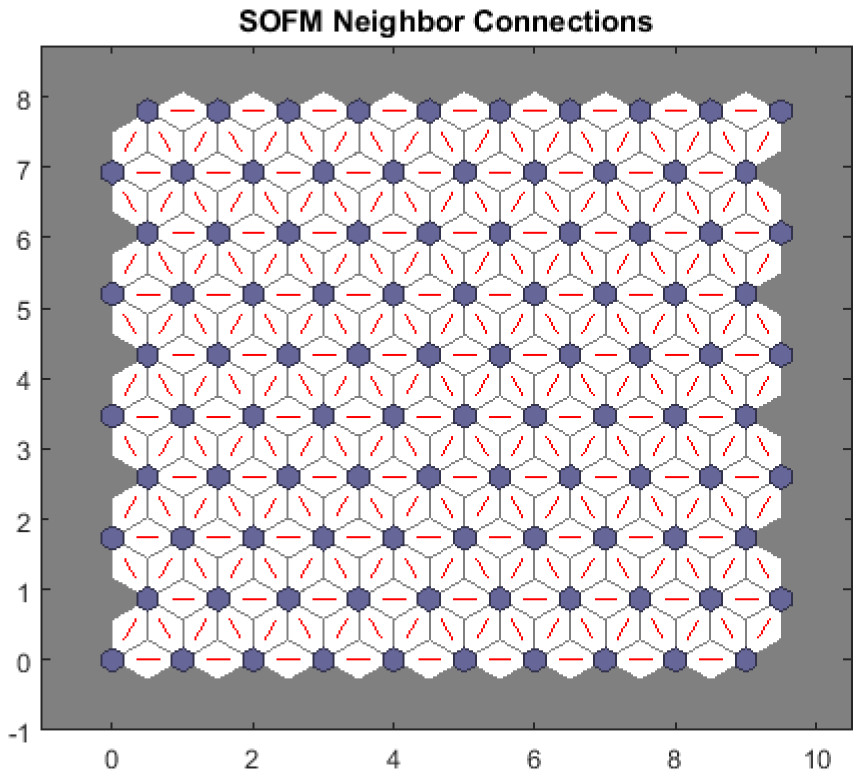


Figure 2-5: An example of SOFM

A self-organising feature map is a form of artificial neural network. It consists of a pre-defined number of interconnected neurons distributed over a 2-D plane as shown in Figure 2-5. Prior to training, the neurons are randomly scattered among the data points and gradually move to the *centroids* of the data clusters they represent as they learn the *features* of the training data. Upon termination of the training, all data points near to a particular neuron, in terms of Euclidean distance[5], are assigned to the cluster that neuron represents. Figure 2-6 shows the results after the clustering is completed.

It is clear from the figure that while some neurons represent the clusters of outliers,

[5]Other distance measures are also possible.

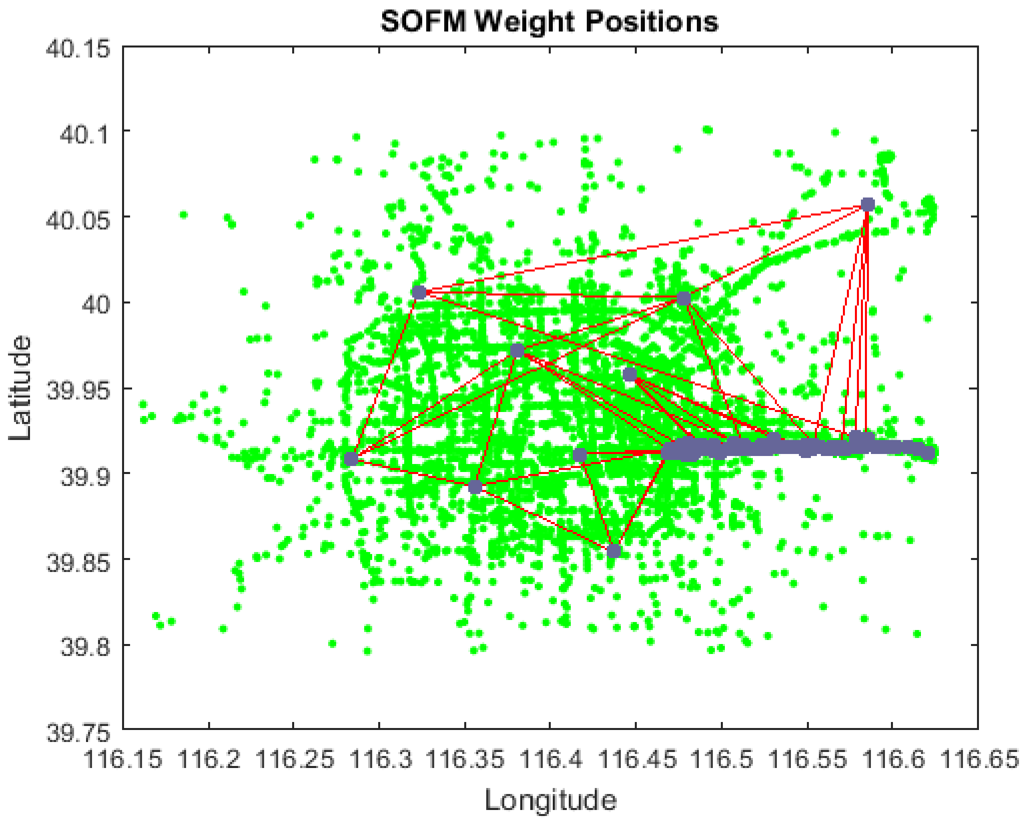


Figure 2-6: A plot of neuron positions after training

majority of the neurons are clustered to *cover* the correct street they should represent. A 10×10 SOFM was used in this project, so there were at most 100 neurons or equivalently, 100 clusters. Each cluster had a different size. To ensure a thorough removal of the outliers, **only the top 50% largest clusters were considered as clusters of correct data points which are called “legal clusters”. All other clusters were deemed as clusters of outliers**.

2.3.3 Outlier Removal

Once the legal clusters were identified, a distance threshold was set to remove outliers so that **whenever the minimum distance between a data point and all cen-**

troids of the legal clusters was above the threshold, that data point would be considered as an outlier and removed. The python-like pseudocode in Listing 2.1 describes this idea with more clarity. Source code is listed in Appendix A.2.

Listing 2.1: Pseudocode for outlier removal

```
Input: a set of GPS records, a set of centroids and a distance threshold
    $d_{max}$
Output: a set of GPS records with all outliers removed

for recd in records:
    min_dist = $\infty$
    for ctrd in centroids:
        min_dist = min(min_dist, get_distance(recd, ctrd))
    if min_dist > $d_{max}$:
        records.remove(recd)
```

However, the *distance* between a data point and a centroid is not as straightforward as Euclidean distance. A centroid, to some extent, can be imagined as a *real* point on the Earth's surface. To calculate the distance between a data point and a centroid is to calculate the **spherical distance** between them, which is given by the Haversine Formula in Theorem 2.

Theorem 2 (*Haversine Formula*)**.** Given two points $P(\lambda_1, \varphi_1)$ and $Q(\lambda_2, \varphi_2)$ on the surface of a sphere, where λ and φ represent longitude and latitude in radians, their spherical distance (the distance along a great circle of the sphere) is given by [9]

$$d = 2R \arcsin \sqrt{\sin^2 \frac{\varphi_2 - \varphi_1}{2} + \cos\varphi_1 \cos\varphi_2 \sin^2 \frac{\lambda_2 - \lambda_1}{2}} \tag{2.1}$$

where R is the radius of the sphere.

Since the Earth is not a perfect sphere but a spheroid, R varies with latitude.

Theorem 3 suggests how to calculate the Earth radius at any latitude.

Theorem 3 (*Radius at any Latitude*)**.** Given a latitude φ in radians, a polar radius R_p and an equatorial radius R_e, the spheroid's radius at that latitude is given by [12]

$$R_\varphi = \sqrt{\frac{(R_e^2 \cos\varphi)^2 + (R_p^2 \sin\varphi)^2}{(R_e \cos\varphi)^2 + (R_p \sin\varphi)^2}} \tag{2.2}$$

It is known that $R_e = 6,378,137$ metres and $R_p = 6,356,752$ metres on the Earth [14] and that Beijing's latitude is about 39°N. Therefore, the distance between a data point and a centroid can be calculated. For this project, two distance thresholds were selected: 30 metres and 50 metres. The thresholds were set in a way that it ensured there was sufficient data for subsequent machine learning tasks while the estimates of travel time were as least affected as possible by outliers. Should the thresholds be set to a too small value, the remaining data could not have been sufficient; on the other hand, however, if the thresholds were set to a too big value, the accuracy of the final results would have been subject to outliers.

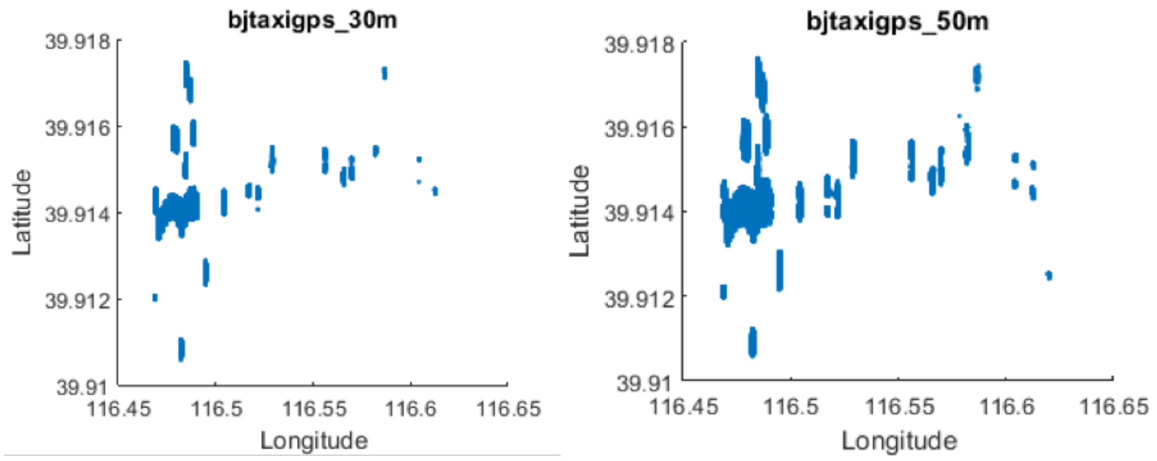


Figure 2-7: A plot of data points after outlier removal

After the outliers were removed, two data sets remained. They are hereinafter referred to as **bjtaxigps_30m**, where outliers were filtered by a distance threshold of 30 metres and **bjtaxigps_50m**, where outliers were filtered by a distance threshold of 50 metres, respectively. bjtaxigps_30m contains approximately 51 million records while bjtaxigps_50m has 59 million. All algorithms and procedures described hereinafter are applicable to both data sets. Figure 2-7 gives a plot of the GPS data points after outliers were removed from both data sets. Clearly, the longitude and latitude now have a much smaller *range* and the data points roughly form a shape similar to that of the street they are supposed to be mapped to.

drivers frequently enough is it recognised as a landmark. In this project, the notion of frequency is given by Definition 7.

Definition 7 (*Frequency of a Road Segment*)**.** The frequency of a road segment is the sum of *unique* occurrences of that road segment in each trip.

Table 3.2 illustrates how the frequency of a road segment is calculated. In Trip 4552265, Street A occurs twice but **its frequency increases only by one**. Similarly, Street C occur three times and Street A occur twice in Trip 1, but **their occurrences only add one to their frequencies**. Therefore, even though Street B *appears* less times than Street A or Street C, **all streets have the same frequency of 2**, based on this set of records.

The algorithm for counting frequency of a road segment is rather straightforward as in Listing 3.2. Source code is listed in Appendix A.4.

Listing 3.2: Pseudocode for counting road frequency

```
Input: a set of trips
Output: a set of frequencies for each street

frequencies = dict{}
for trip in trips:
    streets = unique(trip.streets) // get unique streets
    for street in streets:
        if street in frequencies:
            ++frequencies[street]
        else:
            frequencies[street] = 1
```

After the frequency of each road segment is determined, a parameter k is set to select the top k frequent road segments as *landmarks*. For this project, k was chosen to be 500 to ensure the road segments are significant enough to be landmarks. When plotted on a map, these 500 landmarks cover most of the main streets in Beijing as shown in Figure 3-1.

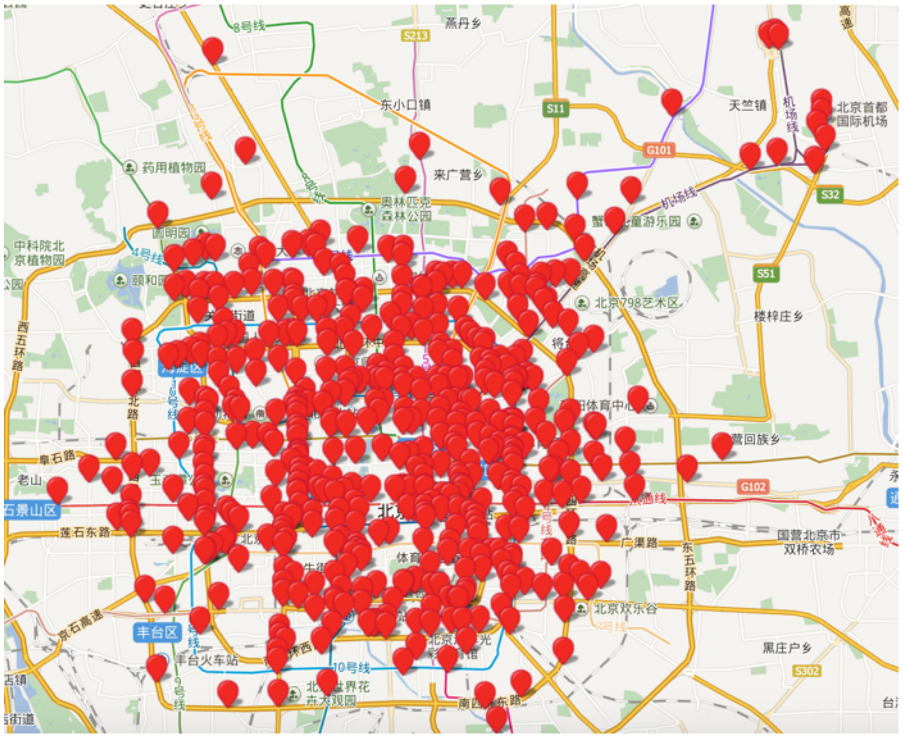


Figure 3-1: A plot of landmarks when $k = 500$

3.3 Landmark Graph

Definition 8 (*Landmark Graph*)**.** A landmark graph is a weighted, directed graph $G = (V, E)$ where V is the set of landmarks defined by parameter k and E is the set of taxi trajectories $T = (p_1, p_2, \ldots, p_n)$ that satisfy the following conditions:

1. p_1 and p_n must be landmarks;

2. $p_2, p_3, \ldots, p_{n-1}$ must **not** be landmarks and;

3. The duration of the trajectory must **not** exceed a threshold t_{max}.

4.1 Travel Time Distribution

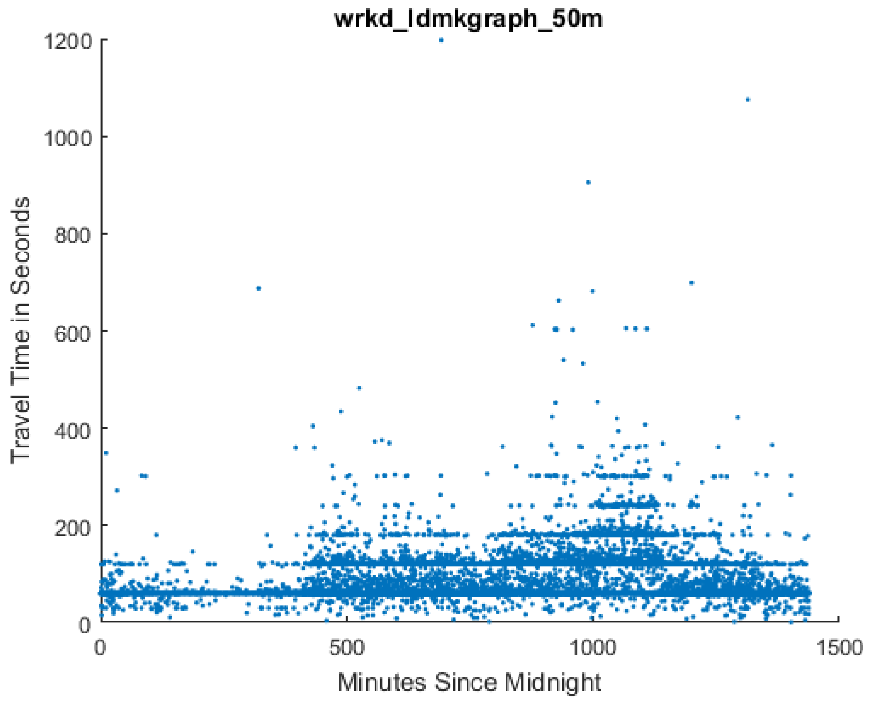


Figure 4-1: An example of travel time patterns

Figure 4-1 illustrates a scatter plot of the travel time of a particular landmark graph edge during the course of a weekday. It can be observed that the travel time does not seem to be a single-valued function with respect to time of the day, as one may expect; rather, the scatter points tend to gather around some values and form some *clusters*. For instance, when it is 500 minutes since midnight, namely 8:20 a.m., the travel time seems to have three main clusters which are represented by three horizontal lines formed by the scatter points. When it is 1,000 minutes since midnight, namely 4:40 p.m., there are about five such lines. This pattern is attributable to three possible reasons:

1. Drivers may actually choose different routes to travel between the two land-

marks, which cannot be captured by the landmark graph since it only knows a driver has traversed from one landmark to the other but not the exact route. Different routes have different traffic conditions and speed limitations, therefore the travel time varies;

2. Drivers have different driving skills, preferences and behaviours. Some drivers just drive faster than others, even if the road conditions are similar and;

3. The GPS devices on taxis reported locations *periodically*, therefore, durations like 60 seconds or 120 seconds are very commonly seen in the landmark graphs. Even if the *actual* travel time is 53 seconds, it is still recorded as 60 seconds. This corresponds to the low-sampling-rate problem mentioned in Section 1.2.

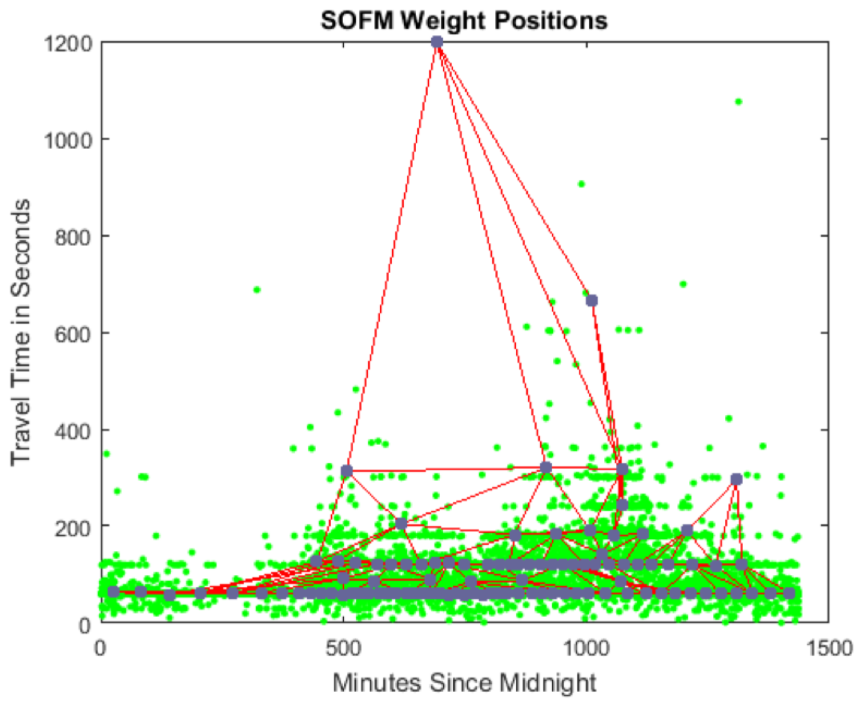


Figure 4-2: A plot of neurons' final positions

Therefore, it is not possible to fit the scatter points with a single-valued function.

Rather, the clustering technique should first be employed to identify the travel time clusters. Like in Section 2.3.2, a **self-organising feature map** is used to cluster the scatter points. Figure 4-2 shows the final positions of the neurons at the end of the training.

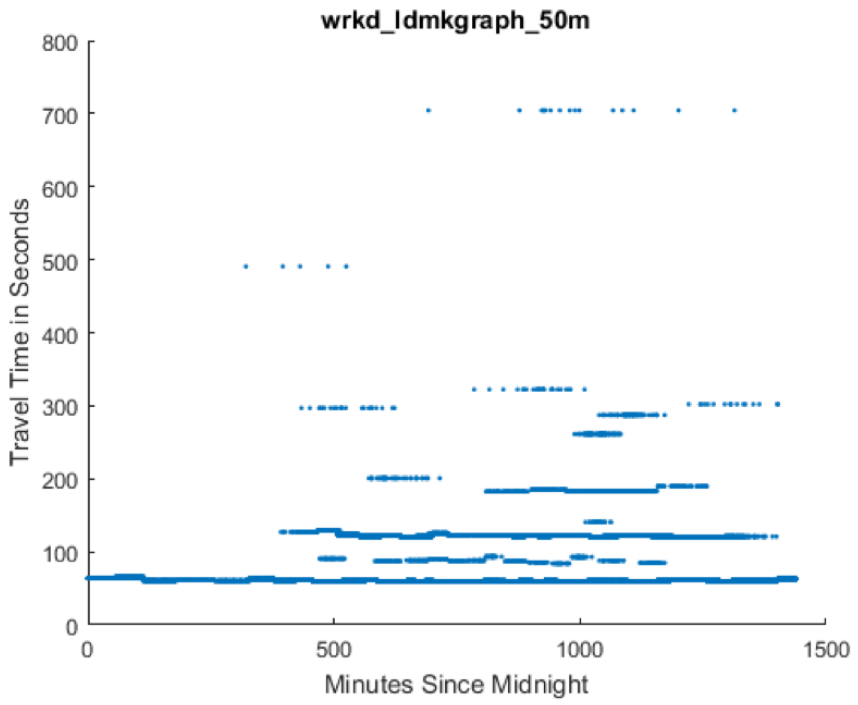


Figure 4-3: An example of representing data points with centroids

One merit of SOFM clustering is *feature extraction*. In this case, after the training is completed, the SOFM has learned some features about the travel time at different times of the day and expressed its understanding by moving its neurons to the centroids of the clusters. Now, **the data points can be represented by their respective centroids**. Figure 4-3 shows the effect of replacing each data point's value with their centroids'. The clusters are clearly shown by the horizontal lines formed by those centroids.

Representing data points with cluster centroids or features learned provides the benefit of *generalisation*. The data set used in this project, albeit large in size, is nevertheless a small *sample* of the *population* of taxi trajectories over time. In other words, these trajectory records are only the *observed* ones but there are potentially infinite number of trajectories that cannot be all observed. The only information known about them is that **they must fit into one of the clusters** after undergoing the same procedures described in the previous chapters. The use of centroids instead of real data points also takes into consideration the unobserved trajectories and reflects the real underlying patterns. In fact, generalisation is an advantage that all neural network-based methods share.

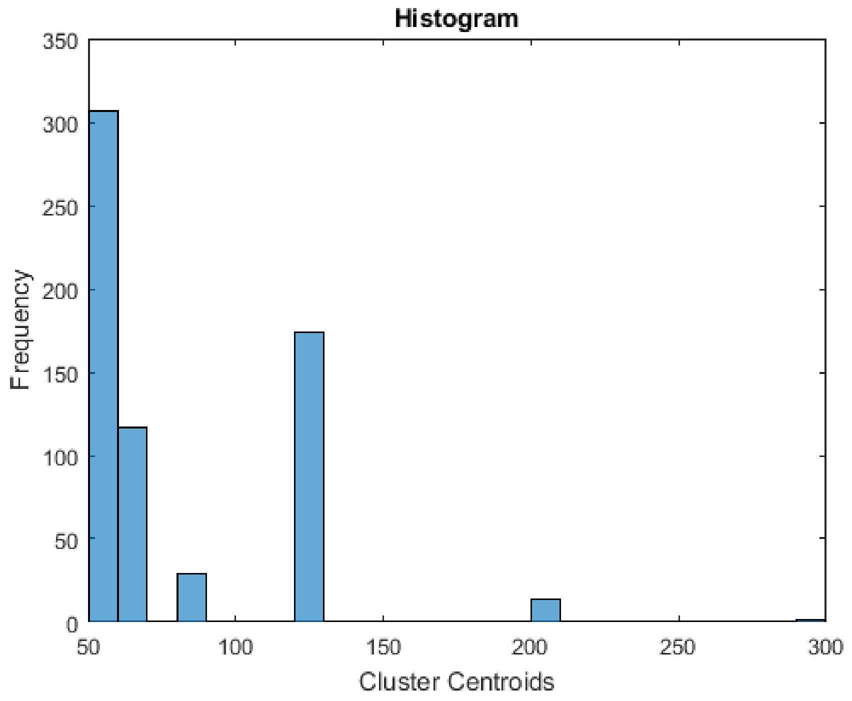


Figure 4-4: An example of distribution of clusters

Now that the clusters are identified, Figure 4-4 shows a histogram of clusters within the 10:00 a.m. to 10:30 a.m. interval. It can be observed that within a particular time interval, there are many clusters of travel time and each cluster has different sizes. To describe the travel time pattern within a particular time interval, the histogram is converted into a cumulative probability distribution of clusters which is then fitted with Weibull Distribution described in Theorem 4.

Theorem 4 (*Weibull Distribution*)**.** A Weibull Distribution is described by two strictly positive parameters, the scale parameter α and the shape parameter β, and has a **probability distribution function** of [5]

$$f(x|\alpha,\beta) = \frac{\beta}{\alpha} \cdot (\frac{x}{\alpha})^{\beta-1} \cdot e^{-(x/\alpha)^{\beta}}. \tag{4.1}$$

Correspondingly, its **cumulative distribution function** is given by [4]

$$F(x|\alpha,\beta) = \int_0^x f(t|\alpha,\beta)dt = 1 - e^{-(x/\alpha)^{\beta}}. \tag{4.2}$$

Figure 4-5 shows an example of fitting a cumulative probability distribution of clusters with Weibull Distribution. The red line represents the cumulative probability distribution of clusters and the blue line indicates the best Weibull Distribution fit estimated by maximum likelihood.

In the referencing paper [15], it uses Inverse Gaussian Distribution to fit the centroid distribution. However, as some experiments showed, Inverse Gaussian Distribution does not always work, especially when the centroids actually have a uniform distribution, namely, when only one cluster appears in the time interval. In that case, Weibull distribution will have the shape parameter $\beta = \infty$ but it still can give the correct result.

The cumulative distribution function of Weibull Distribution is good at calculating the probability whereby the travel time t is less than a particular value. For instance, the probability of the travel time for this landmark graph edge being less than or equal to 100 seconds is approximately $P(t \leq 100) = 0.7$ according to the Weibull

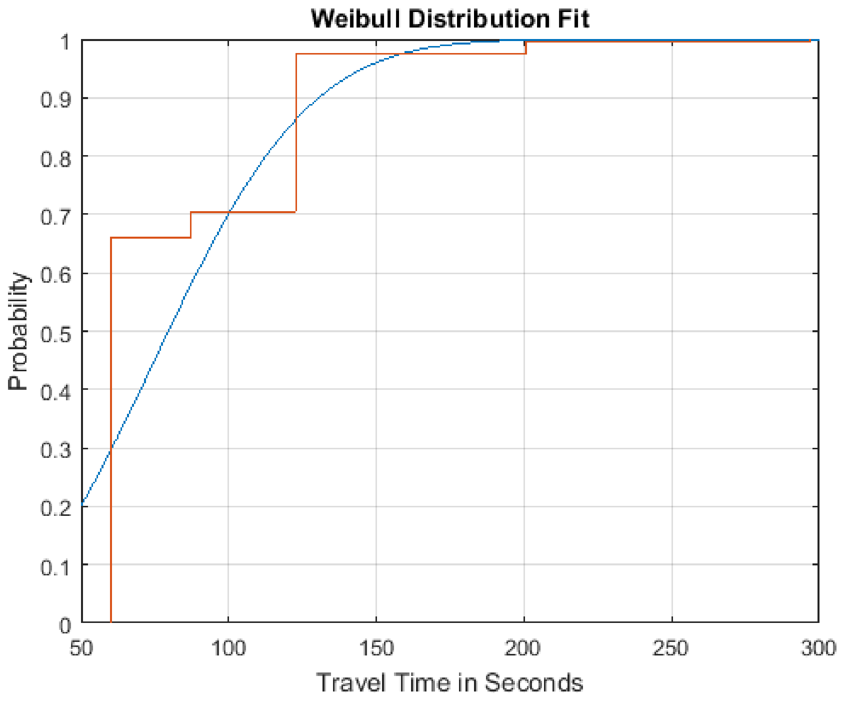


Figure 4-5: An example of fitting data with Weibull Distribution

Distribution curve in Figure 4-5. But to estimate the travel time of a landmark graph edge is in fact the *reverse*: given a known probability p, find the corresponding travel time t, which is exactly the inverse Weibull cumulative distribution function does.

Theorem 5 (*Inverse Weibull Cumulative Distribution Function*)**.** The inverse Weibull cumulative distribution function is defined as

$$F^{-1}(x|\alpha, \beta) = G(p) = \alpha(-ln(1-p))^{1/\beta} \tag{4.3}$$

where p is the given probability.

In practice, the probability p corresponds to some subjective measures of a driver's driving skill. It represents how fast a driver drives as compared to other drivers. If a driver has a p value of 0.2, it means that this driver usually drives faster than 80% $(1 - 0.2)$ drivers. This concept is formally defined in Definition 11.

Definition 11 (*Optimism Index*)**.** An optimism index p indicates how optimistic a driver feels about his or her driving skills [15]. A driver with an optimism index of p_0% usually drives faster than $(1 - p_0)$% drivers under similar road conditions.

The optimism index of a driver can be set by the driver in advance, although it may be subject to the illusory superiority problem[1]. It can also be set to the mean value by default or learned from the driver's trajectory history.

With optimism index defined, it is easy to estimate the travel time using Theorem 5. However, it is worth mentioning that for a cumulative probability distribution function $F(x)$, the probability of a single value is zero, namely, $F(x_0) = 0$; only a range of values can have positive probabilities. Therefore, the travel time calculated from optimism index should really be a *range* of travel time. For instance, if the optimism index is $p = 0.7$, the correct result should be $t \leq 100$ based on Figure 4-5, which means that for a driver with an optimism index of 0.7, his or her travel time is *at most* 100 seconds. No exact value can be determined. However, in the context of this project, only the *worst-case* travel time is considered; therefore, the driver is said to have an expected travel time of 100 seconds.

4.2 Travel Time Evaluation

After the travel time distributions for each significant edge were determined, an evaluation was conducted to ascertain the accuracy of the travel time estimates. In this project, the estimates made by Theorem 5 were compared against real-time estimates made by Baidu Maps at various times of the day.

[1]In a 1986 experiment, *80%* participants rated their driving skills *above-average* [8].

Some complications do arise, however, because the two vertices of each significant edge are actually two *landmarks*. Definition 5 states that a landmark is essentially a road segment with its length ignored. But in estimating the travel time between two landmarks, the length of the landmarks cannot be ignored and has profound effects on how the travel time should be calculated.

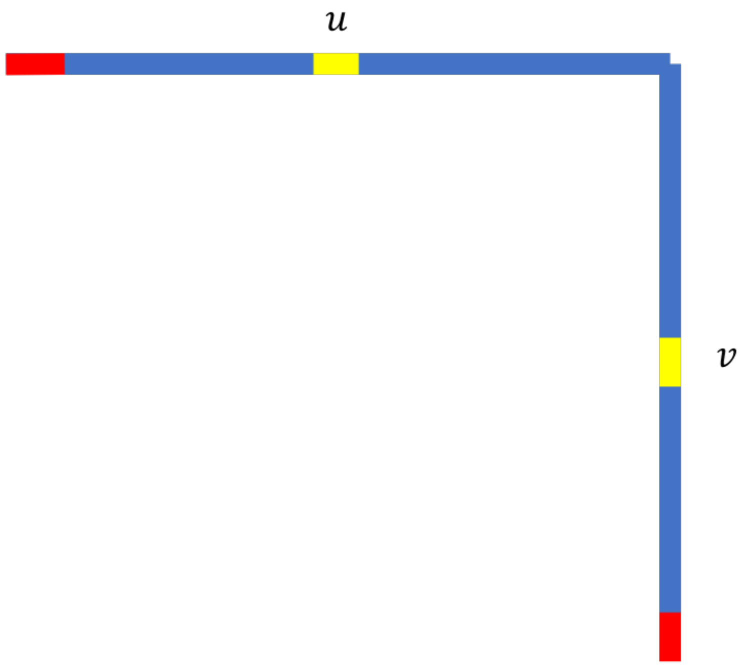
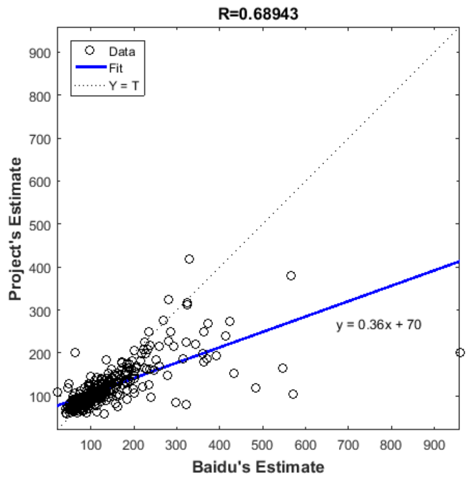


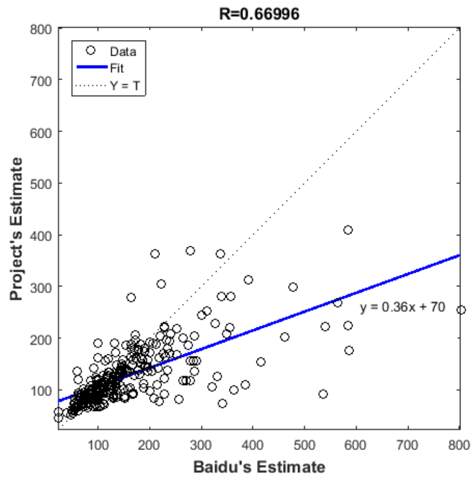
Figure 4-6: An example of different landmark travel time

Imagine there are two landmarks u and v, respectively, and each has a length of L as shown in Figure 4-6. Then, there are potentially infinite number of ways of 'travelling from u to v', because the starting point could be any points on u and the ending point could be any points on v. If a driver starts at the red point on u and stops at the red point on v, the total distance is $2L$ and the total time is T. But had the driver stopped at the yellow mid-point on v, the total distance would be $1.5L$ and the total time would be $3T/4$, assuming the speed is constant.

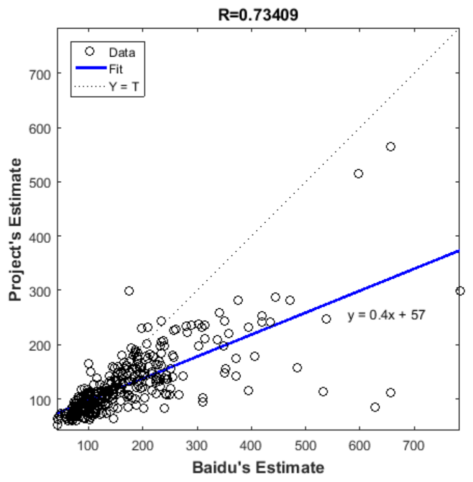
Therefore, it is difficult to give a *definite* estimate of the travel time between two



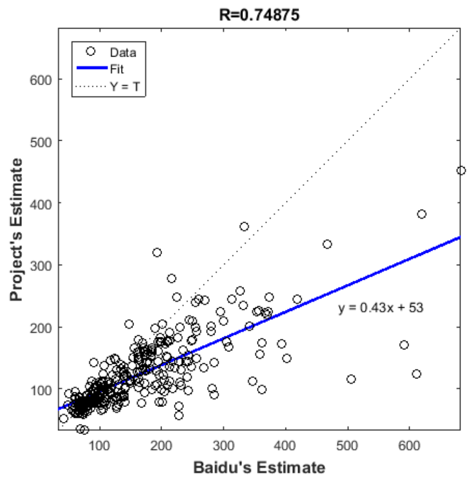
(a) wrkd_ldmkgraph_50m



(b) wrkd_ldmkgraph_30m



(c) holi_ldmkgraph_50m



(d) holi_ldmkgraph_30m

Figure 4-7: A plot of regressions

THIS PAGE INTENTIONALLY LEFT BLANK