Geospatial data analysis

Julijonas Kikutis

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh
2017
Abstract

The travelling trajectories from GPS-enabled devices can be used to observe the behaviour of people and in turn optimize real-world systems. The applications include transportation network planning, control of autonomous vehicles, ride sharing, routing suggestions, recommendation systems, and various others. This report investigates the identification of travel demand patterns. We formulate and discuss the following problem: how to route vehicles in a city in an automated way to maximize the usefulness of the system with limited resources for its users given timestamped trajectory data from people travelling with GPS trackers?

The proposed approach includes two parts. First, we have done the processing to segment the trajectories and find the modes of transportation of the travel segments. Second, we have identified travel demand patterns and constructed routes based on the trajectories that have the relevant modes of transportation.

The first subproblem was approached using two different techniques: (a) classification of points of trajectories and (b) rule-based segmentation of trajectories and then classification of the resulting segments. We find that the classification of points directly achieves higher accuracy that the segmentation approach.

For the second subproblem we propose an unsupervised route construction algorithm with several variations that constructs routes based on the geometric layout of the main travel patterns. Finally, we evaluate the method comparing the total travel time with baselines of equidistant routes and naive clustering of endpoints in a system with limited resources. Our method constructs routes with smaller total travel time values than the baseline methods.
Acknowledgements

I would like to thank my supervisor Rik Sarkar for his guidance, support, and suggestions.
## Contents

1 Introduction ....................................................... 7  
1.1 Overview ..................................................... 7  
1.2 Approach ...................................................... 8  
1.3 Contributions .................................................. 8  

2 Background ....................................................... 11  
2.1 Detection of transportation mode .............................. 11  
2.2 Route construction ............................................. 12  
2.3 Software selection ............................................. 13  

3 Detection of the transportation mode ............................ 15  
3.1 Problem statement ............................................. 15  
3.2 Dataset ......................................................... 15  
3.3 Processing the dataset ......................................... 16  
3.4 Framework of the experiments ................................ 17  
3.5 Classification of points ....................................... 18  
3.5.1 Inclusion of features from the nearby points .............. 19  
3.5.2 Inclusion of coordinate features ............................ 20  
3.6 Segmentation and classification of segments ................. 22  
3.7 Evaluation .................................................... 24  

4 An algorithm for route construction .............................. 27  
4.1 Problem statement ............................................. 27  
4.2 Outline of the approach ....................................... 28  
4.3 Trip preprocessing ............................................ 29  
4.4 Algorithms .................................................... 30  
4.4.1 Normalization .............................................. 30  
4.4.2 Lines in dual plane and dual polar coordinate space .... 31  
4.4.3 Directed variant of dual plane ............................. 34  
4.4.4 Clustering in dual plane ................................... 35  
4.4.5 K-means clustering and regression ......................... 37  
4.4.6 Hierarchical clustering and filtering ....................... 38  
4.4.7 Route endpoint considerations ............................. 40  
4.5 Objective function ............................................. 41  
4.5.1 Graph search ............................................... 41  
4.5.2 Grid search ................................................. 43
## CONTENTS

4.6 Evaluation ................................................................. 44
   4.6.1 Baselines .......................................................... 44
   4.6.2 Artificial example ............................................... 45
   4.6.3 Results on full dataset ....................................... 48

5 Conclusion and future work 53
   5.1 Conclusion .......................................................... 53
   5.2 Future work ......................................................... 54

Bibliography 55

A Experimental results 57
   A.1 Distance measure constants ..................................... 57
   A.2 Inter-cluster distance measure ................................. 58
   A.3 Inconsistency coefficient ....................................... 58
   A.4 Linear regression method ...................................... 59
Chapter 1

Introduction

1.1 Overview

The mobile devices used nowadays have the ability to track various attributes of the world, including the location of the device. Applications of the collected data include optimal and sustainable transportation network design, traffic planning, recommendation systems, routing suggestions, path-finding for autonomous cars, and various others. The data oriented approaches are likely to satisfy the needs of users of the designed systems better than purely engineered designs.

The network design and transportation planning could benefit from data oriented approaches because of the massive amount of GPS location data available. However, these problems are still largely unsolved due to their NP-hardness and many spatial and temporal variables involved. The proposed solutions usually make use of heuristics and iterative checking of generated solutions. We attempt to approach the network design problem by considering a small subproblem and taking just the GPS trajectories as input.

The geospatial data needs to be processed to relate it to the physical world before using it for the network design application. The GPS data might be coming from users that use different transportation modes. For this reason, some of the data might be irrelevant for planning, for example, air travel data for bus network design. This data can be filtered using machine learning to classify travel segments of trajectories into the suitable transportation modes. Moreover, performing such classification can be useful for other tasks such as traffic analysis and intermodal transport planning.

The resulting processed data can be used for a network design application that identifies patterns for travel demands. The work identifying patterns for travel demands can also be useful in other areas. It has applications not only for network design, but also for planning of ride sharing and autonomous vehicles.

We formulate the research question: where to put k vehicle routes to maximize the usefulness of the transportation system for its users? A solution to this problem that reflects travel patterns could increase the utility of the system for its users, reduce
travelling time, and decrease fuel consumption. The limit on the number of routes reflects the limited resource situation in a real-life transportation system. The utility of the system for its users is quantified by defining an objective function of the total time spent by all users in the system.

1.2 Approach

The first part of this dissertation analyses the detection of the transportation mode from timestamped trajectories. The difficulty of the problem lies in deciding between transportation modes that have similar properties such as bike and run, or bus and car. Moreover, the precise detection of change points, which are points in trajectories where the mode of transportation changes, also raises detection issues and has an impact on the subsequent classification.

We attempt to solve this problem by using two different approaches:

- classification of points of trajectories, and
- rule-based segmentation of trajectories and then classification of the resulting segments, which is an approach found in background literature.

Our results indicate that the direct classification of points achieves higher accuracy than the segmentation approach.

The second part of this dissertation considers the question: where to put $k$ vehicle routes to maximize the usefulness of the transportation system for its users? We suggest an unsupervised algorithm with several variations for the discovery of travel demand patterns and route construction from timestamped trajectories. We consider two different clustering algorithms and also provide a variant to construct directed routes. The directed variant could have an application in, for example, a smart bus network in which buses are rerouted based on demand. The proposed algorithm is both automated and computationally feasible for the dataset. Finally, we evaluate the algorithm comparing it with two baselines, placing routes equidistantly and clustering the source-destination trip endpoints in a naive fashion. Our method achieves lower total time values than the baseline methods.

1.3 Contributions

To summarize, the contributions of my dissertation are:

- Design features for the classification of points that allow to find the transportation mode and perform the preprocessing of the dataset as outlined in Section 3.3.
- Implement two different methods of the detection of the transportation mode used in timestamped trajectories in Sections 3.5 and 3.6.
1.3. Contributions

- Develop a map visualization tool for trajectories and the annotated and predicted transportation modes; used to make figures throughout Chapter 3.

- Evaluate the two methods on a real dataset, comparing them with each other and with prior work in Section 3.7.

- Propose and implement an algorithm discovering travel demand patterns for route construction from timestamped trajectories in Section 4.4.

- Define an objective function for the problem and implement its calculation in Section 4.5.

- Evaluate the algorithm on artificial and real datasets comparing it with two baselines in Section 4.6.
Chapter 2

Background

2.1 Detection of transportation mode

There exists prior work on transportation mode detection from timestamped GPS trajectories.

This task was attempted by segmentation of trajectories and classification of segments by Zheng et al [19]. They segment tracks by detecting change points of transportation mode based on speed, acceleration, and minimum distance thresholds. Then they infer transportation mode of those segments using machine learning: decision tree, Bayesian network, support vector machine, and conditional random field. They find that the decision tree gives the best results. We use their approach as the second method of transportation mode detection in Section 3.6. However, we do not know the minimum distance and point merge thresholds they use for their algorithm. They also utilise a probability-based post-processing that is not implemented in our work. Their post-processing approach includes clustering the transportation mode change points using density-based clustering. The clustered centres are taken as vertices in a graph where the segments with different transportation modes are edges. Then the graph is spatially indexed. They calculate the likelihoods of transportation modes for every unique edge and conditional probabilities between different transportation modes. Then the inferred mode is post-processed using the new probabilities if the related change points are found in the spatial index.

A follow up paper discusses additional engineered features of segments used for classification [18] that we also implemented: heading change rate, stop rate, and velocity change rate. These features convey distinguishing information about used transportation mode. Then they cluster the change points using a density-based clustering. The clustered centres are taken as nodes in a graph and the segments with different transportation modes are edges, graph nodes are spatially indexed. They calculate likelihoods of transportation modes for every unique edge and conditional probabilities between different transportation modes. Then they post-process the inferred mode using new probabilities if change points are found in the spatial index.
2.2 Route construction

In general, the network design and transportation planning problems that include both spatial and temporal dimensions are difficult to approach. The reasons for the difficulty lie in the problem formulation and multi-objectiveness, non-linearity and non-convexity of the objective functions, combinatorial complexity of the solution set, and NP-hardness [10]. In general, due to many variables the algorithms usually involve iterative generation of solution instances and checking whether they satisfy the constraints [4].

There are numerous previous proposed unsupervised algorithms at discovering urban mobility patterns in the literature. We looked into the idea of unsupervised trajectory point clustering as a basis for travel demand discovery and network planning.

An interesting application of such clustering using shared nearest neighbours was done to discover meaningful regions [7]. Then a graph was constructed where nodes are those places and weighted edges are the travels between them. The net flows for every region were calculated to show travel pattern changes during different time periods. There also exists work of using geotagged tweets to find origin-destination pairs and perform a similar net flow analysis [5].

Source and destination taxi trip endpoints were clustered to find traffic movement between city regions in Singapore [8]. The authors claim that k-means clustering does not work for their task due to present of outliers, possibly non-convex clusters, and unknown number of clusters. Instead, they prefer hierarchical and density based-clustering. Due to computational complexity of clustering, they only cluster randomly selected samples from the dataset and then extend the clusters with the nearest points in the full dataset. There is also an unpublished work using Gaussian mixture models to cluster taxi trip endpoints into broad source-destination clusters to find city regions between which there is considerable travelling in Singapore [12]. There is work done to cluster source and destination trip endpoints separately using great circle distance radius-based clustering which produces circular clusters and then finding most common origin-destination cluster pairs [15]. The construction of circular clusters is contrary to the expectation that the desired clusters might be non-convex expressed in the previous papers. Another approach to clustering of origin-destination pairs is to use k-nearest neighbours to find close origin and destination points separately and obtain contiguous trips, which are called flows in the paper [22]. The flows are then clustered using shared nearest neighbours-based distance measure and a hierarchical clustering method.

A different approach was taken by [1] where they proposed a mixture model clustering algorithm that takes travel hours into account and uses it on a public transport network dataset. They left open the question on how to choose the number of clusters which needs to be a set parameter. Another approach is employing latent Dirichlet allocation to model the source-destination trip pairs over time in a bike sharing system [2] where origin-destination trips are taken as words and origin-destination ‘templates’, which are similar to clusters, are taken as hidden topics in the LDA model. Here they were able to choose the number of ‘templates’ based on the decrease of perplexity of the
We are also interested in the distribution of trip distances. In a general analysis of a taxi trip dataset in Lisbon [14] they find that the distance travelled is exponentially distributed. This differs from [6] where they analyse human trajectories obtained from cell phone tower locations and find that travel distances follow a truncated power law distribution. This finding is confirmed by a study in Chicago and Melbourne [13]. In this paper, they find that origin-destination travel demand networks (weighted directed graphs) that they construct are affected by the interaction distribution between geographical locations.

We also investigated approaches to route construction based on trip source-destination endpoints. There is a bus route planning algorithm [17] that models the city map as a graph, constructs route circuits by calculating shortest paths between each origin-destination pair in the trip dataset under demand constraints, removes walk edges, clusters similar circuits, and removes routes that do not have sufficient demand. This algorithm is interesting due to its low complexity, no need for iterative generation of solutions, and that it is a fully automated approach to bus network planning. We would like our route construction algorithm to also have these properties.

2.3 Software selection

We chose the software that allows easy prototyping and has a wide range of pre-built algorithms. The Python libraries used for the processing of data, linear algebra and geometry calculations, and experiments are pandas, NumPy, and SciPy. We used the machine learning classifiers from scikit-learn library and the clustering routines from SciPy, NLTK, and scikit-learn libraries. These libraries allow for data processing such as adding new features and rapid experimentation with different machine learning methods. We used the OpenLayers JavaScript mapping library to develop the trajectory visualization tool that highlights the transportation mode, which is used to generate trajectory visualizations that are shown throughout this dissertation. The visualizations also have an underlying OpenStreetMap base map. The Shapely library was used to provide spatial data primitives, such as points and line strings, that have convenient distance and projection functions. For calculation of objective function we constructed the travel graphs using the NetworkX library and we used its implementation of bidirectional Dijkstra’s algorithm.
Chapter 3

Detection of the transportation mode

3.1 Problem statement

We want to detect the transportation mode of travel segments in timestamped trajectories. The difficulty of the problem lies in deciding between the transportation modes that have similar properties such as bike and run, or bus and car. Moreover, the precise detection of change points, which are points in trajectories where the mode of transportation changes, also raises detection issues and has an impact on the subsequent classification.

We attempt to solve this problem by two different approaches:

- classification of points of trajectories, and
- rule-based segmentation of trajectories and then classification of the resulting segments, which is an approach found in background literature.

3.2 Dataset

The dataset for the experimentation is GeoLife [20, 21]. It was collected in Beijing by Microsoft Research and records trajectories of 178 users in a period from April 2007 to October 2011. The trajectories consist of timestamped points and are grouped by their source user. There are 18,670 trajectories in total. There are 182 users out of which 69 have provided the time intervals with annotated transportation mode and thus their trajectories are useful for the classification task. The number of trajectories of those users is 10,906, however, only 4,477 of them have at least one annotated interval. The transportation mode annotations are: walk, bike, bus, car/taxi, subway, train, airplane, boat, run, and motorcycle. The annotated transportation mode counts of the whole annotated dataset are presented in Figure 3.1. The GPS coordinate sampling interval in the dataset is one to five for 91% of the trajectories except when the signal is lost, for example, when travelling in tunnels.
There are several inconsistencies in this dataset. First, even though the transportation mode annotations are specified in the granularity of seconds, the changes between transportation modes do not seem to be precisely timed. As seen later in Figure 3.5, the user walks to the bus station and boards a bus, while the annotation stays walk and only later when bus is already en route it changes to bus. Second, the beginnings and endings of annotations do not precisely match the endpoints of the trajectories. This results in having to discard parts of trajectories that do not overlap with annotations. Third, there are cases when the annotation changes from two non-walk transportation modes without an intermediate walk mode resulting from users having different annotation styles as noted by Zheng et al [19].

### 3.3 Processing the dataset

The dataset consists of points having geodetic coordinates above ellipsoid: latitude, longitude, and altitude. First, we approximately convert them to Earth Centered Earth Fixed Cartesian Coordinates as seen below where $N$ is the average radius of Earth. This allows us to use the Euclidean distance to measure the distance between points, because the distances between points are small enough as the GPS coordinates are sampled every second.

\[
X = (N + \text{alt}) \cos(\text{lat}) \cos(\text{lon})
\]

\[
Y = (N + \text{alt}) \cos(\text{lat}) \sin(\text{lon})
\]

\[
Z = (N + \text{alt}) \sin(\text{lat})
\]

Afterwards, we find the basic features of the points with the following motivation for each of them:

- **distance $d$** between two consecutive points, used to calculate the subsequent features,
3.4 Framework of the experiments

The framework for the experiments is to split the trajectories having at least one annotated interval into a train/validation set (80% of the trajectories) and a held-out test set (20% of the trajectories). The trajectories from the same user might end up both in train/validation set and test set. Then, we train the classifiers and validate different features using 3-fold cross-validation on the train/validation set. We do the final evaluation in Section 3.7 by applying the trained models on the test set.
3.5 Classification of points

The first approach to transportation mode detection is to classify the points in the trajectories directly. We use the basic features outlined above for the classification. The total number of points with annotated transportation modes in the train-validation set is 4,014,559. The first two points and the last point from every trajectory are removed as shown in Figure 3.3. This is because those points cannot be used for classification as the acceleration values (in case of the first two points) and heading change angles (in case of the first and the last points) are not defined at those points. The total number of points after the removal is 3,885,661.

For initial experimentation, we have selected \( v, a, d, \) and \( \theta \) features as the basic features to use. First, we tried different classifiers. The classification accuracy when using different classification methods and same \( v, a, d, \) \( \theta \) features are shown in Figure 3.4. The experiments with multi-layer perceptron classifier included trying out all combinations of hidden layer sizes (5, 5-2, 10, 10-5, 100, and 100-50), L-BFGS and Adam solvers, and ReLU, logistic, and tanh activation functions. The Adam solver achieved
better accuracy and faster training times than L-BFGS in all cases due to large dataset size. The activation functions did not have a large effect on the resulting accuracies. L-BFGS seems to have a peak of accuracy when training hidden layers of size 10-5, while Adam consistently produces a higher accuracy with larger hidden layer sizes. The needed computation time is limiting the achieved accuracy of the Adam solver. The configuration of the best model found for the task is Adam solver and 100-50 hidden nodes. A map of a selected trajectory with colour-coded predicted modes is shown in Figure 3.5.

Afterwards, we did the experiments with various other combinations of basic features. The random forest classifier was chosen for these experiments due to its short training and prediction time on the full dataset. The classification accuracy when fitting on various combinations of basic features using random forest are shown in Figure 3.6. It suggests that the more advanced features help the classifier.

### 3.5.1 Inclusion of features from the nearby points

It seems reasonable that the inclusion of features from the nearby points in trajectory would increase the classification accuracy. We investigated the following features that provide information about the nearby points:

- the features from points -500, -200, -100, -50, -20, -10, 5, 10, 20, 50, 100, 200, 500 places before and ahead, which are called shifted features, added to the feature set of every point,

- the average, sum, minimum, and maximum values of features in moving windows of sizes 5, 10, 20, 50, 100, 200, 300, 500, 1000, which are called rolling features, around every point with it itself being in the centre added to the feature set of every point. When the point is near the beginning or end of its trajectory,
Chapter 3. Detection of the transportation mode

3.5.2 Inclusion of coordinate features

To improve the accuracy of the classification task we could also use the coordinates as features. The accuracies with and without these features can be seen in Figure 3.9. Note that altitude (or $Z$) features are not trained on because they are not available.
3.5. Classification of points

Figure 3.8: Map of user’s 084 trajectory 20081008045306 with colour-coded modes. The border of the circle represents the annotated mode and the fill represents the prediction of decision tree with features from nearby points. Walk – red, bike – green, bus – blue, car – yellow, subway – pink.

Figure 3.9: The increase of accuracy of the random forest with coordinate features added. Fitting on coordinate features improves accuracy.
Chapter 3. Detection of the transportation mode

The border of the circle represents the annotated mode and the fill represents the random forest prediction trained with coordinate features. Walk – red, bike – green, bus – blue, car – yellow, subway – pink.

throughout the whole dataset and their presence or absence could make the model distinguish between the users which supplied altitude and those that did not, thus overfitting on the training data.

We observe the improvement in the change of point classification in trajectory visualisation in Figure 3.10. This particular example illustrates the confidence of the model near roads, where it predicts the car transportation mode. Also, this shows that the classifier learns to discriminate the transportation modes based on the road network of the specific area in which the dataset was gathered. However, it could also be seen as overfitting to the data as the classification accuracy would get better for common roads and travel patterns in an area. Such model could be successfully used in a data rich setting where users all over the world provide their trajectories.

3.6 Segmentation and classification of segments

The second approach for transportation mode detection is to segment tracks by detecting points where the transportation mode possibly changes and classify the resulting segments between those points as per Zheng et al [19].

The segmentation algorithm has been implemented as follows. First, having the velocity $v$ and acceleration $a$ values calculated at every point in the trajectory as outlined in Section 3.3, we can call point a walk point when both of its two values are lower than the maximum walking velocity and acceleration thresholds $v_T$ and $a_T$. The resulting segments which are sequences of consecutive walk or non-walk points are identified as seen in Figure 3.11. The length of the first segment is at least two points, because the acceleration value is not available for the first two points as previously shown in Figure 3.3. Also, we ignore trajectories that consist of fewer than three points. Second, we calculate the total distance $D$ of segment, which is a sum of all constituent
3.6. Segmentation and classification of segments

distances $d$ between points, for every segment. We join any segment which has its total distance lower than the minimum distance threshold with the previous segment. If the first segment is shorter than the minimum distance threshold, we join it with the following segment.

To train classifiers on the resulting segments we need to decide on what transportation mode labels to assign to them. We have chosen to define the label of a segment as the most frequent label (mathematical mode) of the constituent points of the segment following an intuition that the transportation mode of a segment should be the most ‘overlapping’ with the real transportation mode of the constituent points.

The features of segment $S$ calculated and used for classification are:

- total distance $D$,
- average velocity, calculating for the whole segment $\bar{v} = \frac{d}{t_1 - t_0}$,
- expectation of velocity, which is the mean of velocities of points $E[v] = \frac{1}{|T|} \sum_{p \in T} p.v$,
- variance of velocity $\text{Var}[v]$,
- three highest velocities $v_i$ and accelerations $a_i$.

Three additional features as suggested by Zheng et al [18] are also calculated:

- heading change rate $HCR = \frac{|\{p_i | \theta_i < \theta_T\}|}{D}$, which models the fact that pedestrians or runners can change their direction more flexibly than the users of other transportation modes because they are not as constrained by roads,
- stop rate $SR = \frac{|\{\text{groups of consecutive points } p_i | v_i < v_{stop}\}|}{D}$, which reflects that transportation modes have different stop frequency within the unit distance,
- velocity change rate $VCR = \frac{|\{p_i | \text{rate}_{i} > \text{rate}_T\}|}{D}$, which counts points with significant velocity change within the unit distance.

We trained different classifiers using these features. The cross-validation accuracies can be seen in Figure 3.12. Unlike the point classification, the multi-layer perceptron does not provide better accuracy values in this situation, and the random forest achieves better value. The classifier for this task tends to overpredict the bus label.

We investigated different minimum distance thresholds as shown in Figure 3.14. The accuracy increases with the increasing minimum distance threshold because there are
Chapter 3. Detection of the transportation mode

Support vector classifier with linear kernel
Logistic regression
Decision tree
Random forest
Multi-layer perceptron (Adam solver, 100-50 hidden nodes)

Figure 3.12: Accuracy of different segment classification methods. Random forest produces the best accuracy.

less segments created by the segmentation algorithm. Thus it is easier for the classifier to correctly find the most frequent transportation mode in that segment.

3.7 Evaluation

The accuracies of the two methods of transportation mode detection are not directly comparable due to the fact that the segmentation algorithm defines the segments itself and classifies them instead of points directly. We map the classified segments back

Figure 3.13: Accuracy of the random forest with varying point merge threshold.
3.7. Evaluation

Figure 3.14: Accuracy of the random forest when changing the minimum distance threshold in segmentation.

to points and then calculate the accuracy of correctly classified to total points. Now the accuracies are comparable for both segment and point classification. This is done on the test set, which consists of the same held-out trajectories for both classification methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment classification</td>
<td>0.454</td>
</tr>
<tr>
<td>Point classification</td>
<td>0.796</td>
</tr>
</tbody>
</table>

Table 3.1: Accuracies after prediction on test set.

As seen from Table 3.1, the segment classification accuracy on the test set is considerably lower than the point classification accuracy. The paper [18] which was the original source of both the segmentation algorithm and the basic and advanced features obtained an accuracy of 0.728 without doing additional post-processing. It is considerably higher than our segment classification accuracy and closer to our point classification accuracy.

Our reasoning for the low segment classification accuracy is as follows. The main issue with our implementation of segmentation is the detection of segments, for which the minimum distance and point merge thresholds need to be set, which is a quite arbitrary decision. As discussed, we performed grid search and cross-validation on different threshold values and chose the ones giving the best accuracy. We found that increasing the minimum distance threshold increases the accuracy. This is problematic, because we can increase the accuracy until the whole trajectory is a single segment, which defeats the purpose of segmentation. It seems that an approach where an algorithm could do segmentation and classification in one go, taking original annotated segments as inputs and training parameters for both tasks, would be better suited for this task.

It could also be that the assumption to use the most frequent annotated transportation mode in the set of points in a created segment as the annotation of that segment is not
Chapter 3. Detection of the transportation mode

<table>
<thead>
<tr>
<th>True label</th>
<th>walk</th>
<th>bike</th>
<th>bus</th>
<th>car</th>
<th>subway</th>
<th>train</th>
<th>airplane</th>
<th>boat</th>
</tr>
</thead>
<tbody>
<tr>
<td>walk</td>
<td>0.84</td>
<td>0.04</td>
<td>0.09</td>
<td>0.02</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bike</td>
<td>0.09</td>
<td>0.85</td>
<td>0.05</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bus</td>
<td>0.11</td>
<td>0.03</td>
<td>0.82</td>
<td>0.04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>car</td>
<td>0.04</td>
<td>0.01</td>
<td>0.19</td>
<td>0.73</td>
<td>0.01</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>subway</td>
<td>0.08</td>
<td>0.02</td>
<td>0.05</td>
<td>0.08</td>
<td>0.76</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>train</td>
<td>0.01</td>
<td>0.24</td>
<td>0.1</td>
<td>0.07</td>
<td>0.58</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>airplane</td>
<td>0.14</td>
<td>0.25</td>
<td>0.24</td>
<td>0.05</td>
<td>0.32</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>boat</td>
<td>0.18</td>
<td>0.57</td>
<td>0.24</td>
<td>0</td>
<td>0.06</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Predicted label</th>
<th>walk</th>
<th>bike</th>
<th>bus</th>
<th>car</th>
<th>subway</th>
<th>train</th>
<th>airplane</th>
<th>boat</th>
</tr>
</thead>
<tbody>
<tr>
<td>walk</td>
<td>0.59</td>
<td>0.07</td>
<td>0.23</td>
<td>0.07</td>
<td>0.02</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bike</td>
<td>0.3</td>
<td>0.57</td>
<td>0.11</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bus</td>
<td>0.29</td>
<td>0.04</td>
<td>0.55</td>
<td>0.09</td>
<td>0.01</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>car</td>
<td>0.15</td>
<td>0.02</td>
<td>0.24</td>
<td>0.56</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>subway</td>
<td>0.45</td>
<td>0.03</td>
<td>0.1</td>
<td>0.06</td>
<td>0.32</td>
<td>0.04</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>train</td>
<td>0.23</td>
<td>0.01</td>
<td>0.12</td>
<td>0.06</td>
<td>0.59</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>airplane</td>
<td>0</td>
<td>0</td>
<td>0.19</td>
<td>0</td>
<td>0</td>
<td>0.09</td>
<td>0.72</td>
<td>0</td>
</tr>
<tr>
<td>boat</td>
<td>0.72</td>
<td>0</td>
<td>0.28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3.15: Normalized confusion matrices after prediction on test set using point and segment classification.

The best approach because then the classifier trains on not entirely realistic annotations. For example, if a constructed segment matches to two segments having different transportation modes in the dataset, where they change in the middle of the constructed segment, the transportation mode that gets assigned to the constructed segment only ‘explains’ at most half of the real annotation. However, then the question arises how to decide on a better transportation mode annotation of a segment when you have the annotations of the constituent points of that segment.

The confusion matrices of both classifiers on the test set are presented in Figure 3.15. The confusion classification confusion matrix indicates a realistic misdetection of transportation modes such as classifying car or train as bus. The accuracies on the diagonal are considerably lower in the segment classification confusion matrix between the walk, bike, bus, car, subway transportation modes that the point classification is able to predict better. The misprediction of airplane and boat modes can be attributed to the small amount of training trajectories available having these annotations.
Chapter 4

An algorithm for route construction

4.1 Problem statement

After obtaining the transportation modes, the rest of this work investigates the route construction problem. The formulation of this problem is: how to place $k$ routes to maximize the usefulness of the transportation system for its users, given a dataset of source-destination pairs representing the demand for trips? We define the term trip as a travel of an user between the source endpoint $A = (x_0, y_0)$ and the destination endpoint $B = (x_1, y_1)$. Thus a trip is a directed segment $(A, B)$. We define the term route as a constructed route between its source terminal point and its destination terminal point. The input to our problem is a set of trips and the output is a set of routes of size $k$. An example of this is pictured in Figure 4.1.

The simple approach to this problem might be placing trip endpoints in a 4-dimensional space of $(x_0, y_0, x_1, y_1)$ coordinate values, clustering them, and expecting to get constructed routes. However, this is not simply a trip endpoint clustering problem. The approach fails since the only clusters that would be produced in such way are when the endpoints of a few trips are close together, the source and destinations endpoints respectively. We would be unable to detect partially overlapping trips and slight differences in the direction of trips as shown in Figure 4.2 when constructing the routes.

![Figure 4.1: Example of inputs and outputs, trips are solid lines and constructed routes are dashed lines.](image)

27
Chapter 4. An algorithm for route construction

Figure 4.2: Naive clustering might be able to detect a cluster of trip segments (a) with close respective endpoints to form a route, however, it will not detect (b) when they are further away, (c) overlap only partially or (d) have opposite directions.

4.2 Outline of the approach

The route construction problem investigated here is approached geometrically by choosing routes that are the most similar to the underlying trip endpoints of users. The approach is unsupervised. Two versions of the algorithm are presented: one constructing undirected routes and another preserving the direction and constructing directed routes. We use the total time users spend travelling as the objective function to quantify the usefulness of the constructed routes.

There are a few simplifying elements in the proposed algorithm that make the problem more approachable and computationally feasible. All of these issues could be addressed later on top of the proposed algorithm. First, we are analysing routes as geometric objects without mapping them to the underlying network of roads, which can be done as a next processing step. Second, our approach does not establish the locations of stops on the routes, which could be determined in an additional clustering-based step. Third, we do not establish the timetabling, for which other already researched solutions could be used. Finally, we assume that commuters prefer to travel in mostly straight line trajectories in a city. The rationale for this assumption is an empirical observation of travel patterns that people tend to travel in straight lines that closely match their “desire lines” of travel [16]. The authors conclude that “people tend to be oriented by the shortest paths” for transportation modes where people have “decision control”. We also expect that vehicles in an optimal system would follow the desired paths of their users. This assumption could be relaxed by a post-processing step joining the endpoints of constructed routes for which such joining would improve the total time.

The preprocessing we do consists of filtering trips keeping only those that lie in Beijing and projecting their endpoints to a 2-dimensional plane. Then, the route construction algorithm consists of the following steps:

1. normalizing the trip endpoints to lie around the mean between -1 and 1,
2. converting trip segments to lines in the vector equation form in primal plane,
3. performing duality transform, mapping the set of lines to the set of points [3],
4. doing k-means or hierarchical clustering of the resultant points in dual plane with the number of clusters equal to the desired number of routes $k$,
5. transforming the cluster centroids from dual plane to lines in the vector equation
form in primal plane, and
6. performing optimization on trip source-destination endpoints in every cluster in
primal plane to detect outliers.

Some variations on the algorithm are also presented. First, we provide an alternative
conversion to and from the dual plane that preserves the direction of trips. Thus, we can
make directed routes. Second, instead of k-means clustering, we also do hierarchical
clustering without setting \( k \) clusters and as a final step choosing \( k \) lines representing
routes that are being used by the highest number of trips.

We quantify the usefulness of routes using total time all users spend travelling. We
present two methods, graph search and grid search, that calculate time of each trip
given routes. Finally, we evaluate time values of our routes comparing it with two
baselines, placing routes equidistantly and naively clustering the endpoints in a 4-
dimensional space.

4.3 Trip preprocessing

We filter out the trips that are not within the bounding box of the city to be able to use
the remaining trips in our algorithm. Also, we need to be able to calculate distances
between points in the dataset in distance units. For this reason, we project coordinates
to a 2-dimensional plane.

We use the bounding box of Beijing to filter trips that have endpoints that are not
contained within the boundary of the city. After filtering, there are 4,008 remaining
trips. A sample of those trips are represented as line segments between endpoints in
Figure 4.3. We observe that there exist multiple trips having similar endpoints that
came from same users because one user usually has a similar repeating commuting
pattern.

![Figure 4.3: 500 random trips from the dataset in Beijing represented as line segments
between their endpoints illustrating the similarity of trips.](image-url)
Chapter 4. An algorithm for route construction

Figure 4.4: The distortion of distances \( A'C \) and \( B'C \) after a stereographic projection of points \( A \) and \( B \) on a sphere onto a plane \( P \) (cross-section shown).

Because the route construction algorithm has to be able to calculate large distances between points, we need to project the trips from spherical latitudes and longitudes into a local projection on a 2-dimensional plane. A local projection distorts the surface around the defined local area the least so the distances calculated between points in that area are close to real. In general, there are no equidistant projections that preserve distances between arbitrary points. Equidistant projections preserve distances only between one or two chosen points and all other points on the map.

We use the stereographic projection with the latitude of origin and central meridian set to the centre of Beijing to ensure that the Beijing area is distorted the least. As seen in Figure 4.4, the stereographic projection projects a point \( A \) on a sphere to a 2-dimensional plane \( P \) at the intersection of the plane and string going from the north pole \( Q \) to the point \( A \). This results in a higher distortion of distances the more distant from point \( C \) the endpoints are on the sphere. In Figure 4.3 the trips are shown in a stereographic projection.

4.4 Algorithms

In this section we explain the steps to construct \( k \) routes. First, we normalize the trip endpoint coordinates. Second, we transform the trip endpoints to vector line equations in primal plane. Then, we represent the lines as points in dual plane. Third, we use hierarchical clustering of cluster centroids. Finally, we do a reverse transformation and normalization.

4.4.1 Normalization

We normalize the coordinate \([x \ y]^{\top}\) values for both dimensions before the transformation to polar coordinate space using:

\[
\begin{bmatrix}
\frac{x - \min(x)}{\max(x) - \min(x)} & \frac{y - \min(y)}{\max(y) - \min(y)}
\end{bmatrix}^{\top}
\]

This improves the experimental results for artificial route input data. It has a lesser effect on the real projected dataset. This is because the local stereographic projection
already centres the coordinates around the centre of Beijing. However, the normalization is still useful, because later we can reason about the range of coordinate values which will be from $-1$ to $1$. Moreover, we expect that stretching will help the clustering algorithm in situations like in Figure 4.5. Here, if $b$ is longer compared to $a$, clustering algorithm might cluster both sets of lines together. Stretching can prevent this and make $a$ and $b$ approximately the same. Also, if $b$ is shorter compared to $a$ and $d$ is shorter than $b$, stretching can help to enlarge $d$ for the clustering algorithm.

### 4.4.2 Lines in dual plane and dual polar coordinate space

The main idea behind our approach on how to place $k$ routes to maximize the usefulness of the system for its users is cluster formation. We need to find user trips having similar position in space and direction between endpoints. By doing this, we would find a small amount of directions that capture the travel patterns of most users. For this task, we want to express the trips in values that quantify their position and direction. Moreover, we want trips having similar position and direction to have close values. Forming clusters in a 4-dimensional space of their source-destination endpoints $(x_0, y_0, x_1, y_1)$ does not work because clusters would only be formed from trips having similar respective source and destination endpoints as previously shown in Figure 4.2. Thus, in this section we work with trip segments as lines because they represent the direction of a trip. In a later stage, we use trip source-destination endpoints to find terminal stops of constructed routes.

#### 4.4.2.1 Point-line duality

A non-vertical line $\ell$ in primal plane can be described by a line equation $y = mx + c$. This way the line has two parameters: the slope $m$ and the intersection with x-axis $c$. Given two points $(x_0, y_0)$ and $(x_1, y_1)$ on the line, the parameters are:

$$m = \frac{y_1 - y_0}{x_1 - x_0} \quad \text{and} \quad c = y_1 - mx_0$$

In dual plane, it is represented by a point $\ell^* = (m, c)$ as seen in Figure 4.6. In computational geometry, the one-to-one conversion process is called duality transform and...
Chapter 4. An algorithm for route construction

Figure 4.6: Duality transform of two lines.

Figure 4.7: The differences of slopes do not reflect that the lines have equal changes in their directions.

the resultant point is the dual of the line [3]. Our line clustering problem can be approached in the dual plane.

4.4.2.2 Dual polar coordinate space

However, the parametrization of a line \((m, c)\) is not suitable for clustering, because two lines having equal difference in their directions have varying difference in their slopes when they are rotated in space as seen in Figure 4.7. Then the clusters will not represent lines having same directions relative to each other equivalently throughout the whole range of \(\theta\).

Instead, we express the line using its polar line equation \(\ell : r = x\cos \theta + y\sin \theta\) in primal plane. The corresponding dual point is \(\ell^* = (r, \theta)\). As seen in Figure 4.8, geometrically \(r\) is the shortest distance from the origin to the line. In other words, it is the distance of the segment OP going through the origin and perpendicular to the line \(\ell\) where P is the intersection point on \(\ell\). \(\theta\) is the counter-clockwise angle between the x-axis and OP.

This transformation is called the Hough transform in the computer vision research. The Hough transform is used to detect lines and other shapes in an image. For every detected data point in an image, multiple lines are plotted through it. The lines are
4.4. Algorithms

![Figure 4.8](image1)

**Figure 4.8:** Different parametrization of a line for the dual plane in geometric terms.

![Figure 4.9](image2)

**Figure 4.9:** Lines having similar slopes and positions in space have close values of $r$ and $\theta$. Note that difference in $\theta$ is calculated as $\min(|\theta_0 - \theta_1|, 2\pi - |\theta_0 - \theta_1|)$.

Transformed into $r$ and $\theta$ values and the dual points are plotted. In the dual plane, a set of lines going through a single point appear as a sine curve. The intersection point of multiple curves in the dual plane represents a line going through multiple points in the primal plane of the original image.

The properties of the dual polar coordinate space are attractive for the clustering task. Polar coordinates provide a unique parametrization of a line unlike position and direction vectors. This is because a position vector can represent any possible point on the line and a non-normalized direction vector can be arbitrarily scaled. Also, unlike the slope parameter of a line equation, the polar coordinate angle $\theta$ is an angle, so it reflects the relative difference of direction between two lines the same regardless of the rotation of lines. Another useful property of lines expressed in the dual polar coordinate space is that lines having similar slopes and positions in space have close values of $r$ and $\theta$ as seen in Figure 4.9.

We calculate the polar coordinates of a line from vector line equation $\mathbf{r} = \mathbf{p} + \lambda \mathbf{d}$ as opposed to from line equation to be able to express vertical lines. The parameters of a vector line equation, a position vector $\mathbf{p} = [p_x \ p_y]^\top$ that is perpendicular to the line and a direction vector $\mathbf{d} = [d_x \ d_y]^\top$, can be found the following way from two points $\mathbf{a}$ and $\mathbf{b}$ on a line:

$$
\mathbf{d} = \mathbf{b} - \mathbf{a} \quad \lambda = \frac{\mathbf{a} \cdot \mathbf{d}}{d_x^2 + d_y^2} \quad \mathbf{p} = \mathbf{a} - \lambda \mathbf{d}
$$

Then we can find $r$ and $\theta$:

$$
r = |\mathbf{p}| \quad \text{and} \quad \theta' = \begin{cases} 
\arctan2(d_y, d_x) - \frac{\pi}{2} & \text{if } |\mathbf{p}| = 0 \\
\arctan2(p_y, p_x) & \text{otherwise}
\end{cases}
$$
Chapter 4. An algorithm for route construction

Figure 4.10: Parallel lines equidistant from the origin, intersections between them and the perpendicular line going through origin end up in opposite quadrants.

The first case in $\theta'$ formula handles the situation when the line is crossing the origin and the position vector carries no information about the direction of the line. The second case finds the angle from the position vector. Then we make $\theta$ to have range $[0, 2\pi)$:

$$
\theta = \begin{cases} 
\theta' + 2\pi & \text{if } \theta' < 0 \\
\theta' & \text{otherwise}
\end{cases}
$$

The range of $r$ values is $[0, +\infty)$. The range of $\theta$ values is $[0, 2\pi)$. This is chosen for simplicity as opposed to $[-\pi, \pi)$ and to remain consistent with the definition of polar coordinates in the Hough transform.

Note that angles $\theta_0$ and $\theta_1$ have distant values in Figure 4.9 although the lines $\ell_0$ and $\ell_1$ have similar directions. However, as seen in Section 4.4.4, when we are comparing two $\theta$ values, we use the minimum of $\Delta \theta$ and $2\pi - \Delta \theta$ which means that $\theta_0$ and $\theta_1$ are close in that space. The $(r, \theta)$ space is cylindrical as $\theta$ dimension wraps around at $\theta = 2\pi$.

We also have considered the possibility to define $\theta \in [0, \pi)$ to make $\theta$ the same for parallel lines in opposing quadrants. To preserve this information, we would need to define $r \in (-\infty, +\infty)$, otherwise the lines in Figure 4.10 would be considered the same in the dual plane. However, this change would make the lines pictured in Figure 4.9 far apart in polar space as $r_0$ would be positive and $r_1$ would be negative.

The conversion of a line back from the dual plane is the following:

$$
p = \begin{bmatrix} r \cos \theta \\ r \sin \theta \end{bmatrix} \quad \text{and} \quad d = \begin{bmatrix} -\sin \theta \\ \cos \theta \end{bmatrix}
$$

### 4.4.3 Directed variant of dual plane

So far we have worked with the trip segments ignoring their direction from source to destination endpoint. Instead, we can consider trips as directed segments. This would allow us to plan directed routes which can be useful for directed network where
vehicles are reallocated for different routes depending on changing user travel patterns throughout the day, for example, during morning and evening rush hours.

We redefine the angle $\theta$ to convey the direction of a line based on which side of the line the coordinate origin point lies. We could define it as the angle between the directed x-axis and the directed line. However, to remain consistent with the polar coordinate angle definition in the previous section, we subtract $\frac{\pi}{2}$ to get the angle between the directed x-axis and the segment from origin perpendicular to the line. So:

$$\theta'_\text{dir} = \arctan2(d_y,d_x) - \frac{\pi}{2}$$

$$\theta_{\text{dir}} = \begin{cases} 
\theta'_\text{dir} + 2\pi & \text{if } \theta' < 0 \\
\theta'_\text{dir} & \text{otherwise}
\end{cases}$$

Now we have the problem that parallel lines having same direction are indistinguishable in the dual plane. In Figure 4.10, the two lines would both have the same distance $r_0 = r_1$ and the same angle $\theta_0$. For this reason, we make $r$ negative if the origin lies to the right of the line. We detect the side on which the coordinate origin point lies with respect to the line using the cross product of two points on the line:

$$r_{\text{dir}} = \begin{cases} 
|p| & \text{if } p \times (p + d) > 0 \\
-|p| & \text{otherwise}
\end{cases}$$

To convert a line back from the dual plane:

$$p = \begin{bmatrix} r_{\text{dir}} \cos \theta_{\text{dir}} \\
r_{\text{dir}} \sin \theta_{\text{dir}} \end{bmatrix}$$

and

$$d = \begin{bmatrix} -\sin \theta_{\text{dir}} \\
\cos \theta_{\text{dir}} \\
\sin \theta_{\text{dir}} \\
-\cos \theta_{\text{dir}} \end{bmatrix}$$

4.4.4 Clustering in dual plane

Because the points in the dual plane represent lines with similar direction and position in the primal plane and such lines have close $(r, \theta)$ values, we are able to use clustering methods on these values to find the direction and position of route lines underlying the highest amount of trips. The clustering works to find reasonable routes for a set of trip segments, because the formed clusters will capture the general heading direction in space of all constituent trips in that cluster as desired as shown in Figure 4.1.

To do clustering we need to define a distance metric. The Euclidean distance cannot be used because the angle $\theta$ wraps around from $2\pi$ to 0. Our dual plane is in fact a cylinder. Locally our plane is still Euclidean, however, globally it is not. We define the distance metric between two points $u = [r_0 \theta_0]^\top$ and $v = [r_1 \theta_1]^\top$ as follows:

$$d_{uv} = \sqrt{\beta(r_0 - r_1)^2 + \alpha \min(|\theta_0 - \theta_1|, 2\pi - |\theta_0 - \theta_1|)^2}$$

On $\theta$ axis, it finds the closest distance between two points: either the one between 0 and $2\pi$, or the one that wraps around at $2\pi$. Also, we set two constants $\alpha$ and $\beta$, 

that are the weights for the two dimensions. They are needed for comparing the two dimensions because they are not in the same units and ranges.

The intuition for setting $\alpha$ and $\beta$ is as follows. First, we set $\beta = 1$ for simplicity and vary only one parameter, $\alpha$. Then, we notice that the largest difference between $\theta$ values can be when the lines are parallel and symmetric with respect to the origin as seen in Figure 4.11. Then $\Delta \theta = \pi$ and it cannot get any larger, because the distance measure would calculate $\Delta \theta = 2\pi - \Delta \theta'$ instead. Now $\Delta r = 0$ because lines are equidistant from the origin. We want $\Delta \theta$ to reflect the distance between the two lines. It should affect the distance approximately the same amount as $\Delta r$ would have done in case the lines would have been both placed at positive or negative $x$. Then $\Delta r$ would have been the distance between the two lines, $\Delta r' = 2r$. Maximum $r$ value can approximately be $\Delta r = \max_i(|x_i|, |y_i|)$ where $x_i$ and $y_i$ are the coordinates of the line endpoints. Because we did normalization, the coordinate values are in $[-1, 1]$ range, so $\Delta r' = 2$. Then, to make $\Delta \theta$ affect distance the same amount as $\Delta r$ would have, we equate the two distance components when they are in those different situations:

$$(\Delta r')^2 = \alpha(\Delta \theta)^2 \Rightarrow \alpha = \left(\frac{\Delta r'}{\Delta \theta}\right)^2 = \left(\frac{2}{\pi}\right)^2 = \frac{4}{\pi^2}$$

Our experiments calculating the objective function confirm this intuition as seen in Section A.1. There we tried using using both normalized and non-normalized dataset and changing $\alpha$ accordingly.

Using this distance metric, we ran two algorithms based on two clustering methods:

- **k-means clustering** where cluster count is the wanted number of routes $k$, followed by a robust linear regression on every cluster’s trips to detect outliers in each cluster and recalculate the position and direction of routes that are unaffected by them, and

- **hierarchical clustering** which finds the number of clusters itself based on cluster merge distances, followed by only taking the wanted number of routes $k$ out of them having the largest amount of trips in every cluster.

The k-means clustering requires the number of clusters to be specified beforehand. The hierarchical clustering can also produce a set number of clusters, however, it can also...
decide on the number of clusters itself based on cluster merge distances. When using a set number of clusters in k-means we have to remove the outliers that affect the routes, because the hard constraint of cluster number forces k-means to form clusters that include outliers in order to assign all points to clusters. When we let the hierarchical clustering to find the number of clusters itself, we expect that the outliers will form their own clusters, which we will discard afterwards due to a small number of trips that constitute these clusters. For the sake of completeness, we also try a variant of hierarchical clustering set to form strictly $k$ clusters, in the same way we do with k-means clustering, and then detect outliers.

4.4.5 K-means clustering and regression

K-means clustering places $k$ cluster centres and iteratively reassigns points to clusters, minimizing the sum of distances from each point to its cluster’s centroid. Once finished, it produces $k$ flat clusters. We use an implementation of K-means which allows using a custom distance metric and we change $k$, which is the main parameter to our problem.

The k-means clustering does not guarantee to form exactly as many clusters as specified when using a custom distance metric. For example, when $k = 20$, it forms only 19 clusters and leaves one cluster without any data points. This occurs because k-means algorithm with an arbitrary distance function no longer has convergence guarantees as it has with the Euclidean distance.

We take the mean of $r$ values of all trips in a cluster to find the $\bar{r}$ value of the cluster’s centroid. To find $\bar{\theta}$ value we use the formula for mean of angles, so that, for example, the mean of $\frac{3\pi}{2}$ and $\frac{\pi}{2}$ would be correctly equal to zero:

$$\bar{\theta} = \arctan2\left(\frac{\sum_{i=1}^{n} \sin \theta_i}{\sum_{i=1}^{n} \cos \theta_i}\right)$$

After clustering, we could consider the cluster centroids $(\bar{r}, \bar{\theta})$ as the constructed routes in the dual plane and transform those to lines in the primal plane. However, we notice that k-means algorithm was forced to create $k$ clusters and assign every point to exactly one cluster, so every cluster could have outlier trips that, when removed, would make the constructed route more optimal for all other users as seen in Figure 4.12.

We use a robust regression method on the endpoints of trips in the same cluster in primal plane for every single formed cluster to do outlier detection and fit the line that will represent a route. The robust regression methods differ from the ordinary linear regression because they are insensitive to outliers and so their predicted line better fits the data points that are not outliers. For our application, this suggests that after applying robust regression the constructed route will be more optimal for users having trips with most similar position and direction to other trips in the same cluster. We have tried a few different regression methods. The objective function values when using them and without any regression applied at all are presented in Section A.4. The best performing robust method is the Random sample consensus (RANSAC). It
works by repeatedly selecting a random subset of data, fitting a linear regression to it, applying the fit to the whole dataset, and calling the points fitting well a consensus set. This is repeated until we obtain a large enough consensus set.

The resulting values of the objective function when using k-means clustering followed by regression and without it are presented and evaluated in Section 4.6.

4.4.6 Hierarchical clustering and filtering

The hierarchical agglomerative clustering works the following way: each data point starts as its own cluster and clusters are iteratively merged based on the inter-cluster distance measure between them. In turn, the inter-cluster distance measure uses distance metric between points that we defined in Section 4.4.4. There are four commonly used inter-cluster distance measures that allow using a non-Euclidean distance metric between points [9]. These are single, complete, average, and weighted. The single method uses the distance between the nearest points in two clusters as distance between the two clusters. The complete method uses the distance between the farthest points in two clusters. The average method takes the average distance between all pairs of points in the two clusters. The weighted method takes the average of distances between the the first constituent cluster of the first cluster and the second cluster, and the second constituent cluster of the first cluster and the second cluster.

Our intuition for the results of different distance measures is the following. The single method forms clusters as chains of points because it merges the closest clusters. This is not a desirable property for our task, because $r$ and $\theta$ of our lines will be spread out and thus the lines will not have the same overall position and direction in space. The complete method forms more compact clusters but due to considering farthest points it might assign a point to a cluster which might in fact be closer to another cluster’s points. The clusters it produces are close to each other. The average method is a balance between the two previous methods and the weighted method also considers previous cluster merges when computing the average distance. We present the experimental results with different distance measures in Section A.2.

The next step is to choose the amount of clusters from the constructed cluster hierarchy.
Figure 4.13: A sample dendogram showing cluster merges and their heights.

Figure 4.14: A dendogram of hierarchical clustering of $\left( r, \theta \right)$ values in dual plane, 8 top levels of merges displayed.
A dendogram of hierarchical clustering displays a sequence of clustering assignments, where y-axis is the value of the distance measure at which a decision to merge two clusters has been done. While we could just specify a set number of clusters to retrieve, this is not desirable, because the amount of clusters in data is not known beforehand. We want choose such an amount of routes, in which all constituent trips would be different in position and direction approximately by the same amount across all routes. Afterwards, we are able to discard those routes that do not have enough underlying trips. For this reason, we instead can use two other cluster number selection criteria: maximum cophenetic distance between points in a formed cluster, or inconsistency coefficient. The cophenetic distance is the height at which two clusters are merged in a dendogram as seen in Figure 4.13. Limiting cophenetic distance makes all clusters contain points that have approximately same maximum distance from each other. The inconsistency coefficient quantifies the height of each merge in a dendogram compared to the average height of all other merges in the same cluster tree below it. It is calculated as the height of a cluster merge divided by the average of heights of cluster merges below it divided by the standard deviation of all those heights. If a merge has a higher inconsistency coefficient than others, it indicates a boundary between clustered points in the dataset. The dendogram of hierarchical clustering of \((r, \theta)\) values in dual plane can be seen in Figure 4.14. We observe large increase of cluster heights when more distant clusters are merged and expect the inconsistency coefficient to detect where this change is the most visible.

The cophenetic distance for our task is measured by the distance formula above in \(\sqrt{m^2 + \text{rad}^2}\) units. We could set a maximum threshold of this value manually by deciding on reasonable maximum permissible differences of \(r\) and \(\theta\). However, such setting would be arbitrary. Instead, we use a set inconsistency coefficient, which is not changed throughout our experiments. Having a predefined setting is advantageous as the algorithm could work with different datasets given they have the same pre-processing and normalization applied. Experimental results changing the inconsistency coefficient are presented in Section A.3.

The next step is the route filtering. We need to choose \(k\) routes out of all constructed routes. We do this by choosing \(k\) routes that have the highest number of underlying constituent trips, discarding other routes. The users that would have used the discarded routes will be routed through the remaining routes. It might also be the case that it will be faster for them to walk from the source to destination endpoints. This is consistent with the expectations that some users that have very unusual trips might not benefit from the transportation system. The routes are constructed from the clustered trips using the formula for mean of angles defined in Section 4.4.5. Then the \((\bar{r}, \bar{\theta})\) points in dual plane are transformed to lines in the primal plane.

### 4.4.7 Route endpoint considerations

In this work, we do not consider the stop placement problem. For the evaluation of the constructed routes, we model users as being able to board and disembark vehicles at any coordinate on the route. For that, we have to decide only on the endpoint
coordinates of every route.

For every single cluster, we project the endpoints of all trips in that cluster to the route line in primal plane. We calculate the distance from the position vector to the projected point along the direction vector of the line. The distance tells us where the point is on the route relative to other points. We take two points having minimum and maximum distances as the endpoints of the route.

### 4.5 Objective function

Our objective function is the total time people spend commuting when there are $k$ routes given that everyone embarks vehicles and walks to them in the shortest time duration in their trips:

$$
\text{Total time} = \sum_{i=0}^{T} \sum_{j=0}^{N_i} \left( d_w^{(i,j)} + c d_b^{(i,j)} \right) + d_w^{(i,N+1)}
$$

where

$T$ is the number of trips in the dataset,

$N_i$ is the number of vehicle changes for a trip $i$,

$c = v_w/v_b$, walk velocity over vehicle velocity,

$d_w^{(i,j)}$ and $d_b^{(i,j)}$ are distances walking to $j$th vehicle and travelling by $j$th vehicle during $i$th trip of a user, respectively.

The total time is calculated in relative units of time taken to walk one metre. To set the vehicle velocity, we take a bus system as a model for our evaluation. A transportation study in Beijing cites that the average bus velocity of a bus is 9.2 km/h or 2.56 m/s [11]. We set the constant $c = 1/2.56 = 0.39$.

The total time all users spend commuting is calculated in two ways: graph search and grid search. The first method constructs either directed or undirected graph of the map where nodes are transportation mode change points and edge weight represents the time spent travelling when taking that edge. Then we use bidirectional Dijkstra’s algorithm to find the total time spent in the system for every trip. The second method models the map as a rectangular grid where every cell transition has a time value and represents the routes as contiguous cells that have lower transition values. Then we use A* search to find the total time for every trip.

#### 4.5.1 Graph search

We calculate the the total time all users spend in the network by constructing a weighted graph where nodes are transportation mode change points, edges are segments travelled by bus or on foot, and edge weight represents time and searching it for the shortest path.
Chapter 4. An algorithm for route construction

This graph approximately represents the network. It is either directed or undirected graph depending on the variant of the route construction we are using.

We construct the graph and search it the following way. First, every route is added as an edge with an appropriate weight representing the travel time taking that route between terminal stops.

Then, for every two route combinations, shortest distance between them is determined. The general cases of relationships between two routes are: routes can intersect, be parallel, the endpoint of one route can be closest to a point on another, and endpoints of two routes can be closest to each other. These cases are shown in Figure 4.15. Here route 1 intersects route 2, the north endpoint of route 2 is closest to a point on route 3, and the west endpoint of route 1 is closest to the west endpoint of route 3.

For every new node that needs to be created on a route to represent the aforementioned situations, we calculate its lambda value. The lambda value of a point on a segment is the distance along the segment to the point from one of its endpoints. It tells at which distance the node is on the route relative to other nodes. For every route we keep a list of nodes on it sorted by their lambda values. For every new node on a route, an index where it will be inserted into the sorted list is used to find and remove an already existing edge between two nodes that will neighbour the new node in the list. Then, two appropriate new bus edges are inserted, connecting the route. We insert a walk edge between the two change points on the two routes with weight representing the walk time. For the directed version, we take care to preserve the correct direction of route edges when adding and removing them and add only those directed walk edges between two routes that can be possibly traversed given the directions of routes they connect.

Afterwards, for every trip of a user, we add nodes representing the endpoints of the trip. We project their coordinates into every route line and add nodes for the projected points and edges between the neighbouring route nodes and the newly created route nodes same way as outlined in the preceding paragraph. Then we add edges between the newly created route nodes and the endpoint nodes. An example graph we end up for the undirected version that has three routes is shown in Figure 4.15. Finally, we use the bidirectional Dijkstra's algorithm between the source and destination nodes of a trip to find the shortest path and time user needs to travel using the network.
4.5. Objective function

4.5.2 Grid search

We calculate the total time all users spend in the network by modelling the map as a rectangular grid where every cell transition has a time value and representing the routes as contiguous cells that have lower transition values. Then we use A* search to find the total time for every trip.

We construct the grid and search it the following way. First, we multiply every route and trip coordinate value by a constant and round it to the nearest integer to discretize them. Second, we process every route using the Bresenham’s line algorithm to find the cells of the grid that should constitute that route. Two discretized routes are shown in an example in Figure 4.16. The coordinates of grid cells are saved for every route. Then, for every trip we run the A* search between its source and destination coordinates. The A* heuristic we use is the hypothetical time spent travelling by a bus that travels from the coordinate in question to the destination using grid cells, which makes it the Manhattan distance. We chose this heuristic for the A* search as it is admissible so it never overestimates the time to reach the goal cell.

\[ t_{\text{heur}} = c(|x_0 - x_1| + |y_0 - y_1|) \]

Then, to get the distance from one cell to another we check whether the transition between two cells is part of any route, and if it is, we calculate the time needed for travelling one cell by bus, otherwise on foot. For the directed version of the algorithm, we only allow to take a bus transition if the direction of the search transition and bus transition matches.

Both search methods have their advantages and disadvantages. We might not always find the shortest route using the graph search if the needed edge for an optimal travel of a user connecting two routes is missing. This can happen because we only construct one smallest weight edge between every two routes to make graph representation tractable. The grid search loses the accuracy of time values due to discretisation of the map onto a grid. The size of a cell in the grid is proportional to the loss of accuracy. Also, the size of a cell in the grid is inversely related to the time needed for the A* search to complete. The graph search assumes that users only change vehicles between stops where the routes are closest, while the grid search assumes that users are travelling in a discrete grid in directions along its axes. Because the two approaches make different assumptions about the network, we might expect that if both methods produce similar shortest times across all user trips, the time values are correct.
Chapter 4. An algorithm for route construction

4.6 Evaluation

We evaluate by comparing the results of our methods with two baselines. These are naively clustering all trip source-destination endpoints and placing routes equidistantly in a grid. Also, throughout this evaluation we compare the total time values obtained both with graph search and grid search because the two methods make different simplifying assumptions about the network. First, we evaluate our route construction methods on an artificial example that has realistic trips patterns representative of a city. Then, we evaluate on the trip endpoints in Beijing from the GeoLife dataset.

4.6.1 Baselines

The first baseline is naive clustering of trip endpoints \((x_0, y_0, x_1, y_1)\). It is inspired by the clustering approaches described in background reading, where trip endpoint clustering in a 4-dimensional space is expected to uncover travel patterns. It is computed as follows. First, we normalize the endpoints to \([-1, 1]\) in the same fashion as done for our construction method in Section 4.4.1. Then, k-means clustering is used with the Euclidean distance metric and \(k\) set to the number of routes. This baseline is also able to construct directed routes for the evaluation of the directed versions of our methods. We expect this baseline to perform better than the baseline of equidistant routes as it takes into account the travel patterns in a city, however, it will be unable to cluster partially overlapping trips or slight differences in direction as described in Section 4.1. To give an example, the resulting 10 routes after clustering GeoLife trips in Beijing this way are presented in Figure 4.17. Some of them are quite short due to taking averages of trip endpoints when finding endpoints of a route, which intuitively indicates that they will not be able to satisfy most user trips.

Our second baseline constructs equidistant horizontal and vertical routes, \(k\) in total, as seen in Figure 4.18. This is done by summing the vertical and horizontal differences \(\Delta x\) and \(\Delta y\) of the bounding box coordinates. The routes are distributed equidistantly throughout the bounding box area from each other, \(\frac{\Delta x}{\Delta x + \Delta y}k\) of them vertically, and \(\frac{\Delta y}{\Delta x + \Delta y}k\) horizontally. We expect this baseline to perform significantly worse, as it does not take any travel patterns in a city into account.
4.6. Evaluation

Figure 4.18: The baseline of routes placed equidistantly inside the bounding box of the city, \( k = 10 \).

Figure 4.19: The trips of our artificial example. We place trips between residential locations and places of interest in a city, coloured blue and green.

4.6.2 Artificial example

We have constructed an artificial example that resembles the travel patterns in a city to demonstrate our route construction method. The trip segments are shown in Figure 4.19. This dataset was constructed by choosing residential locations and places of interest that users are commuting to and placing trips between these. In this set, there are trips in which users commute between the regions, intersecting trips, close partially overlapping trips, and also outlier trips that do not follow the main travel patterns.

There are 100 trips in this dataset and we construct 5 undirected routes for this artificial example. The length of a side of a cell in the grid search is set to 100 metres.

The box plots of total time spent in the system by using different algorithms are shown in Figure 4.20. The first two pairs of box plots are the baselines: the naively clustered endpoints and 5 equidistant routes. Third is the hierarchical clustering that decides the number of clusters itself using the inconsistency coefficient set to 1.1 found experimentally in Section A.3. It forms 29 clusters and then the clusters are filtered to be only the 5 most used routes by the number of points in a cluster. Fourth is the hierarchical clustering that cuts off the dendogram when it forms 5 clusters and has the robust regression applied to the trips in a cluster. The last two pairs of box plots
Chapter 4. An algorithm for route construction

Figure 4.20: Box plots of total time for the artificial example. Whiskers extend to 1.5·IQR, outliers shown. Our methods produce lower distributions of total time than the baselines.

<table>
<thead>
<tr>
<th>Method</th>
<th>Graph search, s</th>
<th>Grid search, s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive baseline</td>
<td>124,762</td>
<td>139,657</td>
</tr>
<tr>
<td>Equidistant baseline</td>
<td>123,180</td>
<td>125,326</td>
</tr>
<tr>
<td>Hierarchical inconsist.</td>
<td>97,596</td>
<td>104,204</td>
</tr>
<tr>
<td>Hierarchical regression</td>
<td>104,750</td>
<td>108,178</td>
</tr>
<tr>
<td>K-means centroids</td>
<td>105,302</td>
<td>111,135</td>
</tr>
<tr>
<td>K-means regression</td>
<td>110,431</td>
<td>115,630</td>
</tr>
</tbody>
</table>

Table 4.1: Total time using different route construction methods for the artificial example.
4.6. Evaluation

Figure 4.21: Empirical cumulative distribution functions of total time values on the artificial dataset. Hierarchical method performs better as its CDF curve rises quicker than the baselines.

show the k-means algorithm with $k$ set to 5. First one uses the cluster centroid points as routes in dual plane and second has the robust regression applied to the trips in a cluster. The total time spent in the system corresponding to same experiments is shown in Table 4.1.

We observe that all of our evaluated methods perform better than the baselines. This could indicate that our methods are able to construct $k$ routes that maximize the usefulness of the system for users under our objective function. However, the trips in the dataset might have the properties we expected when designing our algorithm and this dataset might be representative of real network patterns only to some extent.

The outliers in the box plot correspond to users having trips that are extremely long given the constructed routes, which is expected due to limited resources, i.e. number of routes. They indicate users most likely having to walk from the source to destination. The difference between the grid and graph search box plots for hierarchical clustering can be explained by our graph search method being inflexible and punishing the outliers. This is because it cannot find the actual shortest path through the transportation system because of missing needed transfer edges connecting routes.

It can be seen that the inter-quartile and the upper whisker ends at different heights across our methods. This indicates different trade-offs. Hierarchical clustering methods form routes that are more convenient for most of the user trips and not useful at all for others as seen from the inter-quartile range being placed lower. The k-means methods provide routes that are more useful for all users but make the total time higher as seen from the height of the inter-quartile range. This is because k-means forms exactly $k$ routes, and hierarchical clustering with inconsistency coefficient forms more routes that are filtered to $k$ most used.

The corresponding empirical cumulative distributions are presented in Figure 4.21. The hierarchical clustering CDF curve rises faster than the baseline curves. At most of
Chapter 4. An algorithm for route construction

4.6.3 Results on full dataset

Now we present in detail the experimental results for the construction of 10 undirected routes using trips in Beijing from the GeoLife dataset. The trips in the dataset can be seen in Figure 4.3.

The length of a side of a cell in the grid search is set to 500 metres for performance reasons of A* search. Although the width and height of the bounding box of Beijing is about 48,000 metres in both directions, the coarse grid does not affect results because we see that the total time resulting from graph and grid search is similar.

The box plots of total time spent in the system by using different algorithms are shown in Figure 4.22. The box plots show the same route construction methods as for the artificial example: naively clustered baseline, baseline of equidistant routes, hierarchical clustering with number of clusters determined from inconsistency coefficient, hierar-
### 4.6. Evaluation

<table>
<thead>
<tr>
<th>Method</th>
<th>Graph search, s</th>
<th>Grid search, s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive baseline</td>
<td>13,772,898</td>
<td>14,875,370</td>
</tr>
<tr>
<td>Equidistant baseline</td>
<td>18,635,327</td>
<td>19,560,587</td>
</tr>
<tr>
<td>Hierarchical inconsist.</td>
<td><strong>12,081,320</strong></td>
<td><strong>12,873,109</strong></td>
</tr>
<tr>
<td>Hierarchical regression</td>
<td>13,788,964</td>
<td>13,520,065</td>
</tr>
<tr>
<td>K-means centroids</td>
<td>14,409,271</td>
<td>15,072,500</td>
</tr>
<tr>
<td>K-means regression</td>
<td>14,805,457</td>
<td>15,825,761</td>
</tr>
</tbody>
</table>

Table 4.2: Total time spent travelling using different 10 route construction methods for the Beijing dataset.

Hierarchical clustering with 10 clusters and regression applied on trips in clusters, k-means with centroids taken as routes, and k-means with regression applied. In this case, the hierarchical clustering with inconsistency coefficient set to 1.1 forms 1,040 clusters.

The total time spent in the system corresponding to same experiments is shown in Table 4.2. The hierarchical clustering with route filtering performs the best according to both the total time and the position of the inter-quartile range in the box plot. The k-means clustering-based methods do not outperform the naive clustering baseline. However, the naive clustering has more users that have a longer travel time than the k-means methods as seen from the outliers in the box plot.

It must be noted that, as explained in Section 4.4.5, the k-means clustering does not always form exactly k clusters when using a custom distance metric. When set to form 20 clusters it forms only 19 clusters that have points in them and we cannot calculate the mean of \((r, \theta)\) for the last cluster without having any points. The lack of routes might be the reason for the worse performance of the k-means method compared to others when increasing the number of routes to construct as seen in Figure 4.26.

Another problem with k-means clustering might be that it includes outliers into clusters when forming k clusters. Although we do regression on trip endpoints in every cluster to remove the effect of outliers from the constructed route, the outliers might have already affected the assignments of trips to clusters. This is in contrast with hierarchical clustering, which forms a large number of clusters, where outliers can have their own clusters and thus do not affect the construction of main clusters representing the most used routes.

As seen from the box plot, the total time distributions are similar among all methods and the naive baseline. The corresponding empirical cumulative distributions are presented in Figure 4.23. With our method, more elements are in the left bins, so the CDF curve rises slightly quicker and is to the left of the suboptimal baseline case. The reason for that might be that there are only 4,005 trips in Beijing in the dataset and these have been obtained from 70 different users. Because the trips of the same user can be similar when the user has the same commute pattern, there can be a few easy to detect routes that all evaluated methods including the baseline are able to detect that satisfy the demand of most of the users.
Chapter 4. An algorithm for route construction

Figure 4.23: Empirical cumulative distribution functions of total time values resulting from the hierarchical method and two baselines on the Beijing dataset. Hierarchical method performs slightly better as its CDF curve rises quicker than the baseline.

Figure 4.24: The histogram of total time values resulting from hierarchical clustering of the Beijing dataset, bin width is 600 seconds (5 minutes). A fitted exponential distribution also shown. The spikes at particular times are visible.
The histogram of time values for the hierarchical clustering method is shown in Figure 4.24. It reveals two spikes that are even more pronounced in the distributions resulting from other clustering methods. These spikes might correspond to two different kinds of trips that have different lengths that are inherent to the dataset. These could be, for example, commutes inside the city centre and from the outskirts of the city. As explained above, due to similarity of trips in our dataset, trips that have similar endpoints might be affecting the distribution and creating the spikes. This is also seen in the distribution of geometric distances between trip endpoints in Figure 4.25. In this histogram, the spikes are shifted right with respect to the first bin compared to the total time histogram. It suggests that those spikes are inherent to the dataset. Also, the constructed routes improve the travel time for users, otherwise the histograms would look the same, i.e. no shift would be observed.

On the histogram we also plotted a curve of an exponential distribution fitted to the total time values. Our reasoning for the exponential distribution is the following. We expect the transportation system would allow a large amount of users to have short trips, and the rest of users would have increasingly longer trips. Then the expectation of the distribution is average trip duration and variance is the spread of the durations. An exponential distribution would also be consistent with the exponential distribution of distances found in a taxi trip dataset [14]. Under such distribution, for our dataset the average trip duration using routes from hierarchical clustering is 53.5 minutes, and the clustering baseline gives 61.9 minutes. A power law distribution would also seem like a reasonable hypothesis due to travel times being a man-made phenomena and existence of power law in travel distance distributions [6, 13]. However, due to the noisiness of the time values, i.e. the spikes, we were unable to fit a reasonable power law curve.
Chapter 4. An algorithm for route construction

The change of total time resulting from hierarchical method when increasing the parameter of the problem $k$ is shown in Figure 4.26. The difference between our method and the baseline decreases when we increase the number of clusters. The naive baseline is able to construct routes similar to our method when $k$ is 100. This is approving our prediction that the trips in the dataset are similar enough and do not exhibit enough of spatial arrangements discussed in Section 4.1, for example, partial overlapping.

From evaluation on the Beijing trips dataset we conclude that it is not a representative dataset of travel patterns in a city because it contains trips that come from same users, thus have similar repeating source-destination pairs, and is not diverse enough for our route construction evaluation. Moreover, the k-means implementation we use has issues forming exactly $k$ clusters due to using custom distance metric. Still, we observe a slight improvement over baselines.
Chapter 5

Conclusion and future work

5.1 Conclusion

In this work we have approached the problem of discovery of travel demand patterns and network planning based on that data. We investigated how to route vehicles such as buses in a city in an automated way to maximize the usefulness of the transportation system with limited resources for its users given timestamped trajectory data from people travelling with GPS trackers. To do that, we formulated a precise research question: where to put \( k \) routes to maximize the usefulness of the transportation system for its users?

We detected the transportation modes using classification of points of trajectories and rule-based segmentation of trajectories and then classification of the resulting segments. For the latter, we followed a simplified version of an existing work on this task. For a full reproduction of their results, a complete reimplementation of their work would be needed. We managed to achieve 79.6% accuracy on a held-out test set from the GeoLife dataset by a different method, classifying points using random forest and adding features from nearby points in a rolling window.

The background literature related to clustering of the source-destination pairs consists of mainly finding the travel demand patterns for the purpose of visualisation and statistics. We proposed another approach based on computational geometry to construct routes that correspond to travel demand patterns. We have shown that constructing lines from source-destination endpoints, using point-line duality, and clustering lines in dual plane is a viable approach for the route construction task with limited resources.

We evaluated our approach on artificial source-destination endpoints and those found in Beijing in the GeoLife dataset. We defined an objective function, the total time all users spend commuting and compared resulting values with naive baselines. We tried several variations of our method: hierarchical and k-means clustering, and forming routes using cluster centroids or using robust linear regression on clustered points. Our method constructs routes with smaller total travel time values than the baseline methods.
In our work, we discovered the limitations of the dataset: it might be not representative of the real travel demands in a city. Also, we discovered some issues with the implementation, that the k-means clustering might be not the appropriate clustering for our task due to the custom distance measure we need.

We conclude that our approach of discovering travel demand patterns does work and could have applications for network design, ride sharing, and autonomous vehicles.

5.2 Future work

There are multiple extensions to our method of route construction that could be investigated as part of MInf Project Part 2.

A broad goal is the design non-straight routes and comparing the results with the current work. These could be merges of route segments that have terminal stops near each other and have sufficient number of users transferring between them. Another approach for construction of non-straight routes is the design of curved routes, which would rely on a transformation of the coordinate space.

There are improvements to the current method that are possible. First, we could check that there are no segments along the routes without any demand, i.e. without projected trips. This is because as lines stretch indefinitely we have to be sure not to make routes where the vehicle is travelling empty. In case there are segments without projected trips, we should split the route and repeat the step choosing \( k \) lines with the highest demand. Second, we can try assigning small trip segments to routes that are have different direction but are still located along those routes. We would evaluate how adding such segments while performing clustering will change the total time. For this, a change of the distance function might be needed.

An important task would be to use another dataset to evaluate our methods considered here to make a comparison and to check whether our prediction that the GeoLife dataset is not diverse enough is correct. A taxi trip source-destination datasets are available and we could process them to use for evaluation. The taxi trip endpoints may exhibit properties that are more representative of real travel patterns that are less pronounced in the GeoLife dataset due to problems discussed in Section 4.6.

To turn the algorithm into a real life application of network design and transportation planning, we need to extend it to define how precisely vehicles will travel in the city. To map the constructed routes into the city’s network of roads, we could explore Dijkstra’s algorithm or one of its variants for large networks. To specify stops on routes, we might take a clustering approach based on the endpoints of trip segments projected onto routes.

Slightly less related, another concern for transport planning is timetabling, which could also be explored. To create timetables, timetabling solutions that involve repeated generation of timetable instances and simulations could be applied and evaluated.
Bibliography


Appendix A

Experimental results

This appendix presents experimental results that verify our choice of parameters in the route construction method.

A.1 Distance measure constants

Experiment settings: 1000 random trips, stereographic projection, hierarchical clustering, maximum 30 clusters, average linkage, walk velocity 1.5 m/s, bus velocity 12 m/s, undirected version. We expect to get lowest total time with $\alpha \approx 0.4$ in the normalized and $\alpha \approx 2 \cdot 21610 \approx 43220$ in the non-normalized version.

Figure A.1: Effect of changing $\alpha$ on total time spent when using the normalized dataset.
Appendix A. Experimental results

Figure A.2: Effect of changing $\alpha$ on total time spent when using the non-normalized dataset.

### A.2 Inter-cluster distance measure

Experiment settings: all 4,005 trips, stereographic projection, min. max. normalization, hierarchical clustering with 1.1 inconsistency coefficient, distance measure constants $\alpha = 1$ and $\beta = 0.8$, walk velocity 1 m/s, bus velocity 2.56 m/s, undirected version.

<table>
<thead>
<tr>
<th>Distance measure</th>
<th>Graph search, s</th>
<th>Grid search, s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>11,857,935</td>
<td>12,661,326</td>
</tr>
<tr>
<td>Complete</td>
<td>12,961,017</td>
<td>13,787,239</td>
</tr>
<tr>
<td>Average</td>
<td>11,954,455</td>
<td>12,873,109</td>
</tr>
<tr>
<td>Weighted</td>
<td>12,606,540</td>
<td>13,529,370</td>
</tr>
</tbody>
</table>

Table A.1: Effect of different inter-cluster distance measures for hierarchical clustering on total time spent.

### A.3 Inconsistency coefficient

Experiment settings: all 4,005 trips, stereographic projection, min. max. normalization, hierarchical clustering, average inter-cluster distance measure, distance measure constants $\alpha = 1$ and $\beta = 0.8$, undirected version.
A.4 Linear regression method

Experiment settings: 1000 random trips, stereographic projection, min. max. normalization, k-means clustering, distance measure constants $\alpha = 1$ and $\beta = 0.8$, 150 clusters, walk velocity 1 m/s, bus velocity 2.56 m/s, undirected version. Total time is calculated for 100 random trips in the dataset using graph search.

<table>
<thead>
<tr>
<th>Method</th>
<th>Total time, s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster centres, no regression</td>
<td>229,905</td>
</tr>
<tr>
<td><strong>Ordinary regression methods</strong></td>
<td></td>
</tr>
<tr>
<td>Linear regression</td>
<td>232,452</td>
</tr>
<tr>
<td>Ridge regression $\alpha = 1$</td>
<td>232,451</td>
</tr>
<tr>
<td>Ridge regression $\alpha = 100$</td>
<td>232,452</td>
</tr>
<tr>
<td>Lasso $\alpha = 1$</td>
<td>232,452</td>
</tr>
<tr>
<td>Lasso $\alpha = 100$</td>
<td>232,453</td>
</tr>
<tr>
<td><strong>Robust regression methods</strong></td>
<td></td>
</tr>
<tr>
<td>RANSAC</td>
<td>226,896</td>
</tr>
<tr>
<td>Theil-Sen estimator</td>
<td>229,181</td>
</tr>
<tr>
<td>Huber regression $\alpha = 100$, $\varepsilon = 50$</td>
<td>233,100</td>
</tr>
</tbody>
</table>

Table A.2: Effect of different regression methods after k-means clustering on total time spent.

Figure A.3: Effect of changing the inconsistency coefficient on the number of clusters formed by the hierarchical clustering.