Geospatial data analysis. Route construction for shared travel

Julijonas Kikutis

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2018

Abstract

The goal of this work is the discovery of travel demand patterns and route construction from timestamped trajectories. This problem is important for transportation network design, traffic planning, recommendation systems, routing suggestions, and various others. The more concrete question is "Where to put k vehicle routes to maximise the usefulness of the transportation system for its users." We say the transportation system satisfies a user if they are not required to walk a large distance. This results in another question "If each user is willing to walk a maximum distance d, where to put k vehicle routes?"

The proposed algorithms construct routes on a road network graph. To evaluate we use three real-life taxi trajectory datasets from San Francisco, Rome, and New York, and the associated network graphs. We calculate the total time users spend in the system and the ratio of satisfied users. We compare our measured results with two baselines: naive endpoint clustering in Euclidean space and placing equidistant routes.

First, we show that the problem can be formulated as submodular optimisation. We design a method of route construction based on the submodular maximisation which greedily maximises the number of satisfied users. It produces reasonable results as evaluated using the total time users spend in the system and the ratio of satisfied users. However, it is not performant when the number of input trips is large.

Then, we devise an angular dual space which is based on a geometric construction into which we transform our input trip segments and then apply clustering. This method is generally faster than the submodular optimisation.

Acknowledgements

I thank my supervisor Rik Sarkar and his student Maria Astefanoaei for support, ideas, and suggestions.

Contents

Chapter 1

Introduction

There are many geospatial datasets tracking the movement of people across various transportation modes available on the internet and the amount of available data is increasing. The transportation network design and route planning could benefit from data-oriented approaches. We can use the data to understand transportation demand patterns and subsequently help optimal and sustainable network design and route planning. Moreover, it is applicable to recommendation systems, routing suggestions, path-finding for autonomous cars, and other problems.

The motivation for this work stems from the fact that vehicle route planning is either solved using heuristics and iterative checking of solutions or designed manually. We investigate the subproblem of constructing vehicle routes taking solely the historical trip trajectories as input. Examples of such unprocessed trajectories from three different cities are shown in Figure 1.1. The challenge here is to find the routes that would satisfy the most users.

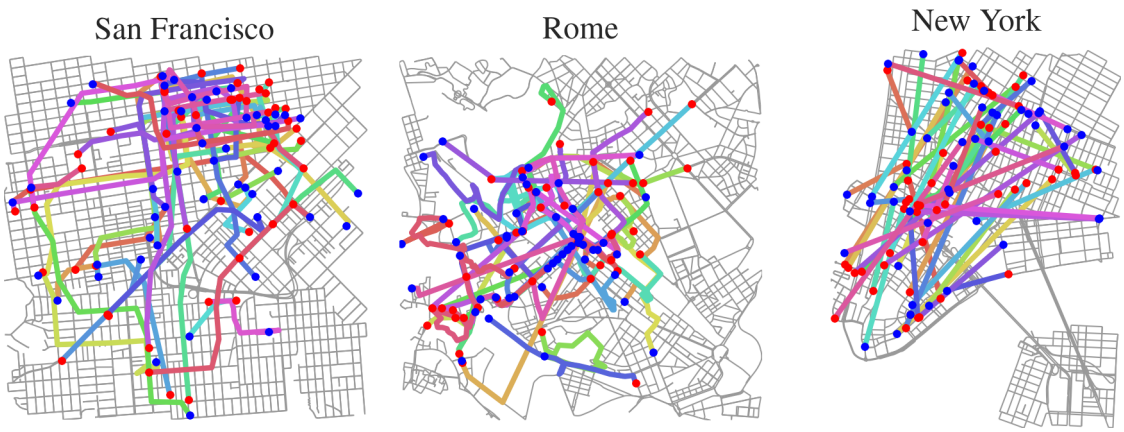


Figure 1.1: 50 random trajectories from San Francisco, Rome, and trip endpoints from New York.

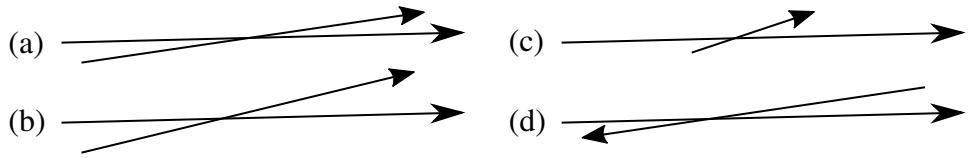


Figure 1.2: Naive clustering might be able to detect a cluster of trip segments (a) with close respective endpoints to form a route, however, it will not detect (b) when they are further away, (c) overlap only partially, or (d) have opposite directions.

1.1 Motivation

The clustering-based approaches might look as an attractive solution as they operate on whole datasets and provide a unique solution even when a clear clustering is not visible to a human eye as in Figure 1.1.

We can decide to initially approach route construction as a clustering task because the trajectories consist of 2-dimensional points. A naive method is to naively cluster passenger trip endpoints (x_0, y_0, x_1, y_1) in a 4-dimensional plane using the Euclidean distance as a distance metric. However, it fails to detect partially overlapping trips and slight differences in the direction of trip segments. Those cases are visualised in Figure 1.2. This is because the Euclidean distance assigns high values just between those 4-dimensional points which are close to each other in all four dimensions. But exactly the situations shown in the figure are important for the route construction problem because those passengers can travel using the same vehicle. The naive clustering only detects the situation when the respective endpoints of a few trips are close together.

The motivation for our work was also verified experimentally. The cumulative distribution of total time needed to travel in San Francisco with set bus and walking velocities can be seen in Figure 1.3. The 4-dimensional hierarchical clustering using the Euclidean distance metric constructs routes that result in a visible improvement in the total time over plainly walking from the source to destination endpoints. The question then arises how large a potential improvement could be in this travel setting using a different algorithm that accounts for the outlined issues.

1.2 Goals

We would like to discover the travel demand patterns and then construct routes from timestamped trajectories. The goal of this work is to investigate the question "Where to put k vehicle routes to maximise the usefulness of the transportation system for its users." Knowing the answer to this question would help transportation planning on various scales, from buses in cities to trains across countries. We can also set a constraint that a transportation system is deemed useful if passengers are not required to walk a large distance. This results in another question "If each user is willing to walk a maximum distance d, where to put k vehicle routes?"

It is desirable that the algorithms developed would not have many hyper-parameters

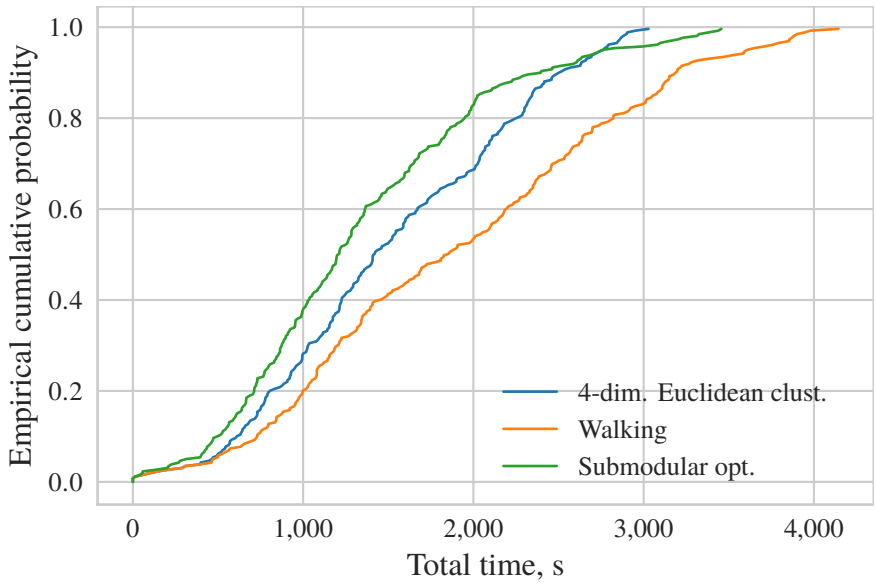


Figure 1.3: The cumulative distribution of the total time of trips using the bus system with routes produced using different methods. The naive Euclidean clustering in the 4-dimensional space is compared to a walking baseline without routes, routes constructed using the submodular optimisation, and routes from the clustering-based method (introduced later). Dataset: 259 trips 2 km around the centre of San Francisco, forming 7 routes.

that need to be set, would seamlessly work on multiple datasets of various sizes of extents and trajectory counts, and would have a reasonable time complexity.

1.3 Approach

In this work, we are proposing two methods for the route construction from timestamped trajectories. We are constructing the routes in a road network graph setting, as opposed to different methods applicable to the purely geometric case analysed last year.

The first method is based on the submodular optimisation which maximises the number of satisfied users. A user is considered satisfied when the walking distance of a user does not exceed a set distance d. We introduce the method in Chapter 4.

The second method is based on expressing the trips in the angular dual space and using k-medoids clustering to produce routes. This approach does not directly optimise any evaluation metric, although it assigns every trip to one route out of a set number of k routes based on their similarity in the dual space. It is described in Chapter 5.

The submodular optimisation method outperforms the clustering-based method with respect to the total time spent travelling using the routes, this can be seen in Figure 1.3. However, the running time of k-medoids clustering in the angular dual space has a substantially better running time when the trajectory count is increasing as seen in Figure 1.4 due to more optimal computational complexity.

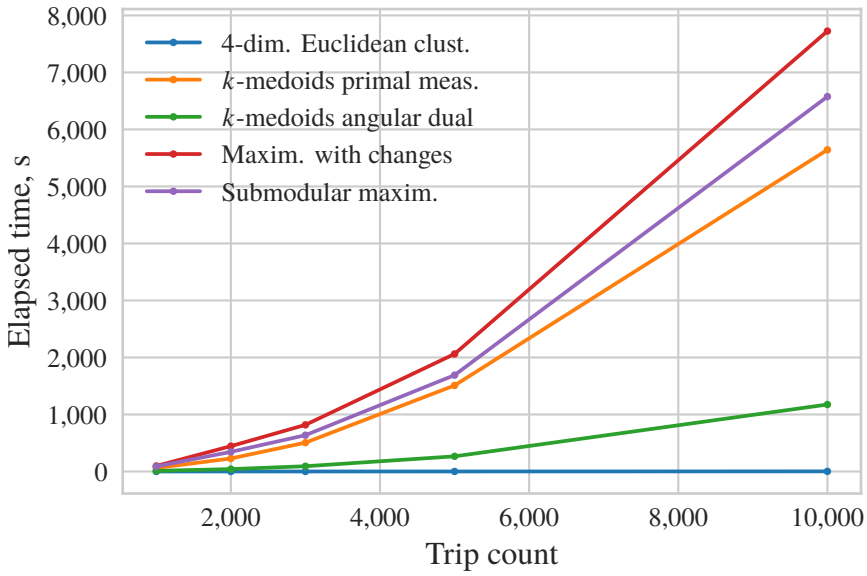


Figure 1.4: Elapsed system-wide time during clustering and submodular methods applied on different dataset sizes. San Francisco, radius 2 km, 6 routes, max. walk length 200 m.

1.4 Software selection

We have done the programming work for this report using Python 3.6, processed data and done geometry calculations using NumPy, SciPy, and pandas, and created figures using matplotlib. The hierarchical and k-means clustering algorithms are taken from SciPy and scikit-learn. The k-medoids algorithm is taken from Pycluster, available in Biopython, see de Hoon et al. [2003]. We use Shapely geospatial data processing primitives, such as points and line strings, and geospatial operations, such as finding a distance and checking whether one geometry contains another, geopandas to store collections of primitives, pyproj to project data. The city maps are converted into graphs from the OpenStreetMap data using OSMnx and stored in NetworkX digraphs. Raster city base maps are downloaded using the Cartopy library.

1.5 Contributions

The contributions of this dissertation are:

- Propose a dual space transformation that has desirable properties for clustering in Section 5.3.
- Propose, implement, and analyse a route construction algorithm that uses the angular dual space transformation in Chapter 5.
- Propose, implement, and analyse route construction algorithms based on submodular optimisation in Chapter 4.

- Preprocess the datasets described in Chapter 3 and construct their graph representations as detailed in Section 3.2.
- Implement two baseline route construction methods on a graph described in Section 6.1.
- Define and implement two evaluation measures for the problem based on graphs in Section 6.2.
- Evaluate the algorithms and baselines on the three datasets in Section 6.3.

1.6 Outline

The outline for the rest of the report is:

- In Chapter 2 we present background literature on trajectory and line segment clustering and our previous work.
- In Chapter 3 we introduce the data types we are working with, the road network graph construction, our dataset processing, and comment on the datasets that we use.
- In Chapter 4 we present the submodular maximisation, formulate our problem as the submodular maximisation, and analyse the resulting algorithm.
- In Chapter 5 we investigate clustering as a method of route construction. We present clustering of trip segments in primal space, propose a dual transform of trip segments based on a geometric construction and the subsequent clustering in dual space.
- In Chapter 6 we present the baselines, two evaluation measures, measure our proposed methods, and comment on the results.
- In Chapter 7 we overview our goals, approach, and results, and provide ideas for further research.

Chapter 2

Background

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, and combinatorial complexity of the solution set [Owais et al., 2014]. In general, due to many variables the algorithms usually involve iterative generation of solution instances and checking whether they satisfy the constraints [Fan and Mumford, 2010].

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

2.1 Endpoint clustering

An interesting application of such clustering using shared nearest neighbours was done to discover meaningful regions by Guo et al. [2012]. Then a graph was constructed where the nodes are those places and weighted edges are the commutes between them. The net flows for every region were calculated to show the travel pattern changes during different time periods. There also exists work using geotagged tweets to find origin-destination pairs and perform a similar net flow analysis [Gao et al., 2014].

Source and destination taxi trip endpoints were clustered to find traffic movement between city regions in Singapore [Kumar et al., 2016]. The authors claim that the k-means clustering does not work for their task due to present of outliers, possibly non-convex clusters, and an unknown number of clusters. Instead, they prefer hierarchical and density based-clustering. Due to the 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 travel in Singapore by Rus et al. [2011]. There is work done to cluster the source and destination

trip endpoints separately using the great circle distance radius-based clustering which produces circular clusters and then finding the most common origin-destination cluster pairs [Verma and Cheah, 2016]. 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 by Zhu and Guo [2014]. The flows are then clustered using shared nearest neighbours-based distance measure and hierarchical clustering.

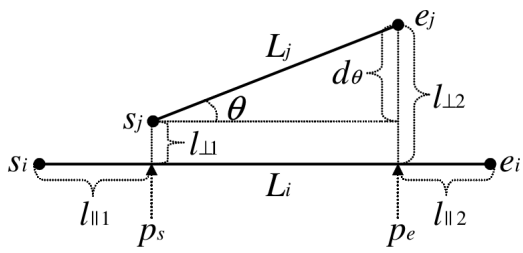
A different approach was taken by Briand et al. [2016] where they proposed a mixture model clustering algorithm that takes travel hours into account and uses it on a public transport network dataset. They left open the question on how to choose the number of clusters which needs to be a set parameter. Another approach is employing latent Dirichlet allocation to model the source-destination trip pairs over time in a bike sharing system [Come et al., 2014] 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. They were able to choose the number of 'templates' based on the decreasing perplexity of the model.

We also investigated approaches to route construction based on the trip source-destination endpoints. There is a bus route planning algorithm by Xian et al. [2013] 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 algorithms to also have these properties.

2.2 Trajectory clustering

The paper by Lee et al. [2007] proposes a trajectory clustering method in which trajectories are partitioned into line segments and those line segments are then clustered. Their motivation of sub-trajectory clustering is for the analysis of movement in a certain region, for example, of hurricane trajectories.

The partitioning is done using a method similar in its aim to existing curve simplification algorithms, which is finding the smallest subset of trajectory points such that the difference between the original trajectory and trajectory that connects points in a subset would be as small as possible. They use the minimum description length (MDL) principle. The hypothesis length $\mathrm{L}(H)$ is formulated as the sum of lengths of the simplified segments. The encoded data length $\mathrm{L}(D|H)$ is formulated as the sum of distances between the trajectory segments and simplified segments with respect to a distance metric $d_\perp + d_\theta$ defined in Figure 2.1. Their algorithm is approximating the minimum of $\mathrm{MDL} = \mathrm{L}(H) + \mathrm{L}(D|H)$ by greedily not adding points to a simplified trajectory until MDL of a trajectory with the next point added is smaller than MDL



$$d_\perp = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}} \tag{2.1}$$

$$d_\parallel = \min(l_{\parallel 1}, l_{\parallel 2}) \tag{2.2}$$

$$d_\theta = |L_j| \sin\theta \tag{2.3}$$

Figure 2.1: Components of distance functions between two segments L_i and L_j by Lee et al. [2007].

without the next point.

As seen in Figure 2.1, the authors define the *perpendicular distance* $d_\perp$ that reflects the distance between two segments, *parallel distance* $d_\parallel$ that measures the extra length of one segment compared to another, and *angle distance* d_θ measuring difference in directions between segments. The distance metric used in clustering then is the sum $d_\perp + d_\parallel + d_\theta$. In Section 5.1 we will explain how our developed segment similarity measure in primal space relates to the aforementioned distance measures.

Then the authors perform partitioned segment clustering using DBSCAN density-based clustering algorithm in which they change its definitions from clustering points to clustering trajectory segments. They put an additional constraint in the algorithm so that it would not form clusters solely out of segments from a single trajectory. This way, they are able to find similar sub-trajectories between segments.

The last part of their paper presents a method to find representative trajectory from a single cluster and a heuristic to set DBSCAN parameters. The evaluation is based on calculating the sum of squared distances between segments inside each cluster and changing DBSCAN parameters.

Unlike the described paper, we do not have the need to detect common sub-trajectories. The inputs for our clustering approach are the source and destination endpoints and trajectories themselves are flexible. This is because we are investigating what are the best k routes to create if each passenger is willing to walk the maximum distance d. Thus we are constructing trajectories freely between the endpoints.

2.3 Previous work carried out

The report "Geospatial data analysis" submitted last year was structured in two parts. Its topic was devising ways to use geospatial data for network design and transportation planning.

As the input data, we had the GPS data coming from users that use different transportation modes. In the first part of the report, we used machine learning algorithms to learn and classify parts of trajectories with transportation modes. We devised variables that were associated with coordinates in trajectories and parts of trajectories that were used as input to classifiers. The main issue was the precise detection of change points, which are points in trajectories where the transportation mode changes. In this part, we compared two approaches: the classification of trajectory points and rule-based segmentation of trajectories and then classification of the resulting segments. Our results indicated that the direct classification of points achieves higher accuracy than the segmentation approach.

In the second part of the report, we investigated a problem formulated as how to place k vehicle routes to maximise the usefulness of the transportation system for its users, given a dataset of source-destination pairs representing the demand for trips. We quantified the utility of the system for its users by defining an objective function of the total time spent by all users in the system.

We suggested an unsupervised algorithm with several variations for the discovery of travel demand patterns and route construction. This is done by normalising the coordinates, transforming trip segments into dual polar coordinate space, clustering using an appropriate distance metric, and post-processing the clusters representing routes. There were several alternative clustering and post-processing methods and both directed and undirected versions of the algorithm. Finally, we evaluated the algorithm comparing it with two baselines in a geometric setting. The objective function was calculated using bidirectional Dijkstra's algorithm in a graph and A* star search in a discretised grid. Our method achieved lower objective function values than the baseline methods.

Chapter 3

Datasets and preprocessing

In this chapter we define the data types we are working with and overview the datasets that are used for evaluation, their advantages and disadvantages. The historical trajectory datasets described here are the taxi trip endpoints in New York City, taxi trajectories in Rome, and taxi trajectories in San Francisco. We have also obtained and use the road networks of the aforementioned cities.

3.1 Definitions

In this and the subsequent chapters we use the following definitions.

A **trajectory** is a sequence of spherical or projected coordinate pairs. A **trip** is a trajectory between the source (pickup) coordinates $A = (x_0, y_0)$ and destination (dropoff) coordinates $B = (x_1, y_1)$ and it represents a single travel of a passenger in a taxi. A **trip segment** is a directed line segment (A, B) drawn between the source and destination coordinates.

The terminology of geospatial libraries for these primitives is as follows. A trajectory is called a *LineString*. A set of trajectories is called a *MultiLineString* and a set of points a *MultiPoint*. In our implementations, sets of primitives are implemented using the aforementioned types, because that allows using geospatial queries, such as "are points within a set of polygons", on multiple primitives at once.

A **road network** is a directed graph where edges represent roads and nodes represent crossroads. A **path** in a road network is a sequence of edges which connect a sequence of vertices. An **edge** has an associated weight representing the length of a section of a road between two intersections on the map under the local projection. We call a path a **route** when it is an algorithmically constructed path, potentially satisfying the needs of multiple users.

3.2 Road network graphs

The route construction will be performed in a map setting, thus we have to obtain a graph representation of the map.

The road networks of the cities mentioned in following sections are taken from the OpenStreetMap project. The maps are downloaded and turned into a graph data structure using the OSMnx library [Boeing, 2017]. We will perform experiments with paths situated 2 km radius around the city centre, therefore we construct the graph from a map area that is centred on the city centre and has a circular shape of the same radius. Each edge is weighted by the length of the road between intersections that correspond to nodes. The lengths on edges allow us to use algorithms on the graph to find lengths of paths. We simplify the graph by discarding nodes and edges where there is no intersection between roads and creating a single edge instead. This makes the number of nodes and edges substantially smaller, which allows us to perform computations on the graph quicker. We then find the largest strongly connected component in that directed graph to make sure we can create paths between all pairs of nodes. This is not generally the case because we took a circular cutout of the map and there were nodes that were only strongly connected to other nodes outside of that cutout. Then we project the graph to the local projection of the city as detailed in Section 3.4. We use the stereographic projection which conveniently sets the coordinate origin point to the centre of the graph.

3.3 Common dataset preprocessing

We work with taxi trip datasets from New York, Rome, and San Francisco. Each user might have multiple trips in the same dataset but this information is not available to us, thus we consider trip and user as having a one-to-one correspondence.

In evaluation we work with random samples of trips of sizes 1,000, 2,000, 3,000, 5,000, 10,000, and 20,000 to keep the computation time short. Our angular dual space transform method needs all trips to be within a circle of a set radius. Thus we only use the trips that fall within a circle for evaluation of all methods to be able to compare between different methods, as all of them should use the same input data to be comparable.

Finally, we match the coordinates of the sampled trajectories to graph nodes for Rome and San Francisco datasets which have full trajectories. We find the nearest graph node to each coordinate. We add a node to the path if the distance between it and the coordinate is not larger than 100 metres. The path between two consecutive added nodes is fully established using bidirectional Dijkstra's algorithm. This is needed because the matching process can skip some nodes needed for a valid path on the graph. An example of 10 random trajectories from San Francisco dataset matched to graph nodes can be seen in Figure 3.1.

The resulting graph paths are validated. First, they must have a minimum length

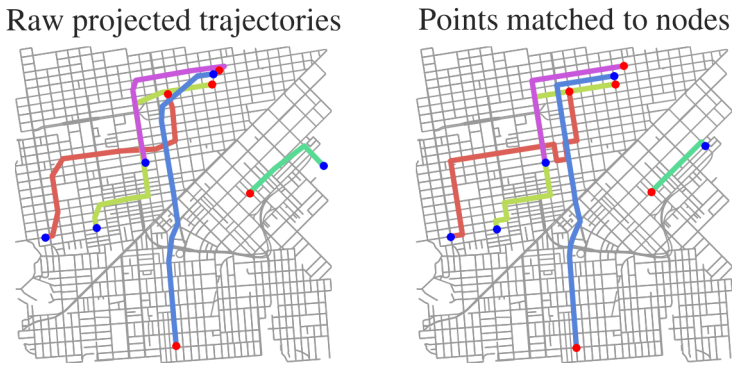


Figure 3.1: Trajectories before and after their coordinates are matched to the graph nodes and path between them interpolated with the Dijkstra's algorithm.

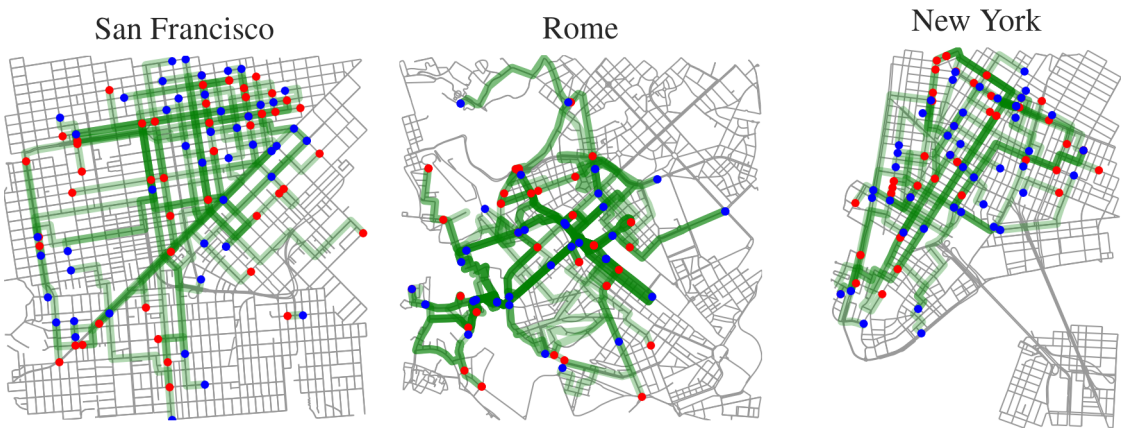


Figure 3.2: Network graphs of 2 km radius in each city and 50 processed routes sampled from the corresponding dataset. The start points indicated in red and end points in blue.

of 100 metres to make sure we do not work with unreasonably small trajectories. Second, the path length has to be smaller than the scaled Euclidean distance between its endpoints. We use a scaling factor of 3. This ensures that we skip the trajectories that go back to their start points after taking another direction. We need this constraint because our angular dual route construction method works solely on endpoints, thus such trajectories would be misleading. 50 random samples from a circle of 2 km in each dataset after matching to the road network are shown in Figure 3.2.

3.4 Projection

We want to calculate distances between points and thus we need to convert trip endpoints into spherical latitudes and longitudes into local projections in a 2-dimensional plane. For this reason, we are using a stereographic projection which distorts the surface around the defined local area the least. This projection is not equidistant, but, in general, there are no equidistant projections that preserve distances between arbitrary points. For each of the following datasets, we define the stereographic projection taking the centre coordinates of the corresponding city as the latitude of origin and central

meridian. We use the same corresponding local projection of the city both to project the coordinates in the following datasets and the map networks of the cities as explained in Section 3.2.

3.5 Taxi trip endpoints in New York

We are using the first file from the New York City taxi trip dataset obtained using a New York Freedom of Information Law request by Whong [2013]. The file contains 14,776,615 pickup and dropoff coordinates from the month of January 2013. Out of these, 420,830 are within 2 km circle around the city centre, which is 40.7127° N, 74.0059° W.

This dataset only contains the pickup and dropoff coordinates and does not contain the whole trajectory coordinates, thus we will not be able to calculate total cost travelling using the taken route in real life versus our constructed route accurately. We will use the bidirectional Dijkstra's algorithm to find paths and distances between the endpoints instead. This simplification will not account for the road preferences such as choosing major roads and less traffic as real taxi trajectories presumably would.

Because the road network in the city centre of New York – like in Manhattan – is composed of rectangular blocks and parallel roads, we make an observation that there will be many paths between two points that have the same length. Thus in such setting, in theory, the shortest path problem might not have a unique best solution.

3.6 Taxi trajectories in Rome

We are using the *CRAWDAD* Rome taxi dataset [Bracciale et al., 2014]. It contains taxi trajectories of 320 cabs from the month of February 2014 and there are 21,817,851 coordinates in total. The city centre of Rome is at 41.9028° N, 12.4964° E.

This dataset contains the coordinates annotated with the taxicab numbers. It has the whole trajectories of cabs but does not have the constituent trips separated. Because of that, we use Ramer–Douglas–Peucker curve simplification algorithm [Douglas and Peucker, 1973] to find the simplified curves. Then we split the original cab trajectories at the points that were kept in the simplified curves if the trajectory satisfies the minimum length (1 km) and minimum straight line distance between endpoints (500 m) constraints. The resulting trajectories do not entirely reflect real trips, however, they do reflect the occupied taxi travel patterns in the city.

3.7 Taxi trajectories in San Francisco

The mobility traces of taxi cabs in San Francisco are taken from Piorkowski et al. [2009]. The data is collected over 30 days in the San Francisco Bay Area. The files

contain the trajectories of 535 taxis during their whole ride, including the time without passengers. Unlike the Rome dataset, each coordinate also has an attribute whether the taxi is occupied, thus we were able to write an algorithm to reconstruct the actual trips. The total number of trips in this dataset is 453,346. There are 67,185 that are within a disc of 2 km radius around the city centre, which is located at 37.7749° N, 122.4194° W.

3.8 Synthetic datasets

We have also created two trip segment datasets helping to observe properties of the clustering algorithm. First, there is a synthetic dataset imported from a hand-drawn vector image files containing line segments. It is used to construct a visualisation of the properties of our methods. Second, there is a dataset of trip segments where their endpoints are placed randomly inside the circle of radius r at coordinates

$$\left(yr\cos\left(2\pi\frac{x}{y}\right) \quad yr\sin\left(2\pi\frac{x}{y}\right)\right) \quad \text{where} \quad x, y \sim \text{Uniform}[0, 1).$$

This dataset contains uniformly randomly placed segment endpoints inside a circle. It will provide us insights about how the line segments and points appear when transformed into the angular dual space and what are the properties of this space.

Chapter 4

Route construction using submodular optimisation

We ask how to route vehicles in a city in an automated way to maximise the usefulness of the system with limited resources for its users given timestamped trajectory data from people travelling with GPS trackers. We would like to know where to put k vehicle routes to maximise the usefulness of the transportation system for its users. To do that, we specify that the transportation system is useful if passengers are not required to walk a large distance. More specifically, our route construction problem is defined as follows: "If each user is willing to walk a maximum distance d, where to put k vehicle routes?"

In this chapter, we investigate a method of route construction based on the submodular optimisation to maximise the number of satisfied users.

4.1 Submodular optimisation

A **submodular function** is a set function which maps a set $S \subseteq \Omega$ to a real number and has the property that the marginal increase in its value caused by the addition of a single element to its input set S is smaller or equal as the size of S increases.

An example of a submodular maximisation problem is selecting a set of size k of predetermined subareas, possibly overlapping, that would cover the largest possible area. The value of the submodular function is the covered area. The marginal coverage depends on other subareas in the selection and more selected subareas mean less marginal gain from each additional individual subarea.

As defined in Mirzasoleiman et al. [2016], a submodular problem maximises a submodular function subject to a cardinality constraint

$$\max_{S \subseteq V} f(S) \quad s.t. \quad |S| \leq k.$$

Suppose a function $f(S)$ represents the total benefit of selecting a set S. A set function $f : 2^V \rightarrow \mathbb{R}$ is submodular if it satisfies the following equation: for all $A \subseteq B \subseteq V$ and

for all $x \in V \setminus B$

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B).$$

Also, f is monotone iff $f(A) \leq f(B)$ for all $A \subseteq B \subseteq V$.

The submodular problem is NP-hard for many classes of submodular functions. Nemhauser et al. [1978] have shown that a greedy algorithm that begins with an empty set and iteratively improves the current solution with an element of the maximum incremental submodular function value

$$v^* = \operatorname*{argmax}_{v \in V \setminus A} f(A \cup \{v\}),$$

until k elements have been chosen, is guaranteed to provide a constant factor approximation. The solution of size k from greedy algorithm has a submodular function value of at least a constant factor $(1 - 1/e) \approx 0.632$ of the optimal solution.

4.2 Submodular maximisation approach

In our scenario, the submodular optimisation maximises a submodular function, which maps a set of possible routes to a count of satisfied passengers in the dataset. A passenger is satisfied if he or she does not have to walk a distance greater than d metres in a single continuous walk. We use a greedy submodular maximisation algorithm to find the routes.

The candidate route set in our algorithm is equal to the trip set. This is an approximation of all possible paths in a road network graph because trying all possible paths in a road network graph would be intractable. If the number of trips is sufficiently large, it is a sufficiently good approximation of all reasonable routes for vehicles.

An example of the submodular maximisation constructing routes is given in Figure 4.1. It shows each addition of a new route, the trip endpoints of newly satisfied users, and the expanding area where users would be satisfied if they had their the trip endpoints there. We observe that the property of submodular function takes effect – the marginal increase of its value is getting smaller as there are fewer newly satisfied users at each iteration.

The pseudocode for the greedy submodular maximisation is given in Algorithm 1. The $\textsc{DistP}(l, p)$ function calculates the distance between a line string l and point p. Its time complexity is $O(m)$ where m is the number of points in l.

Let n be the number of trips (and users), m be the maximum number of points in a line string of a trip, and k be the number of routes to construct. The big-O notation here always refers to the worst-case time complexity. We assume sets are implemented as trees with $O(k)$ element lookup, insertion, and deletion.

On each iteration of route selection starting at line 4 we go through all trips as candidate routes at line 6. At line 7, we find the set of all dissatisfied users that would be satisfied with the new route. The size of the set is the increase of the submodular function

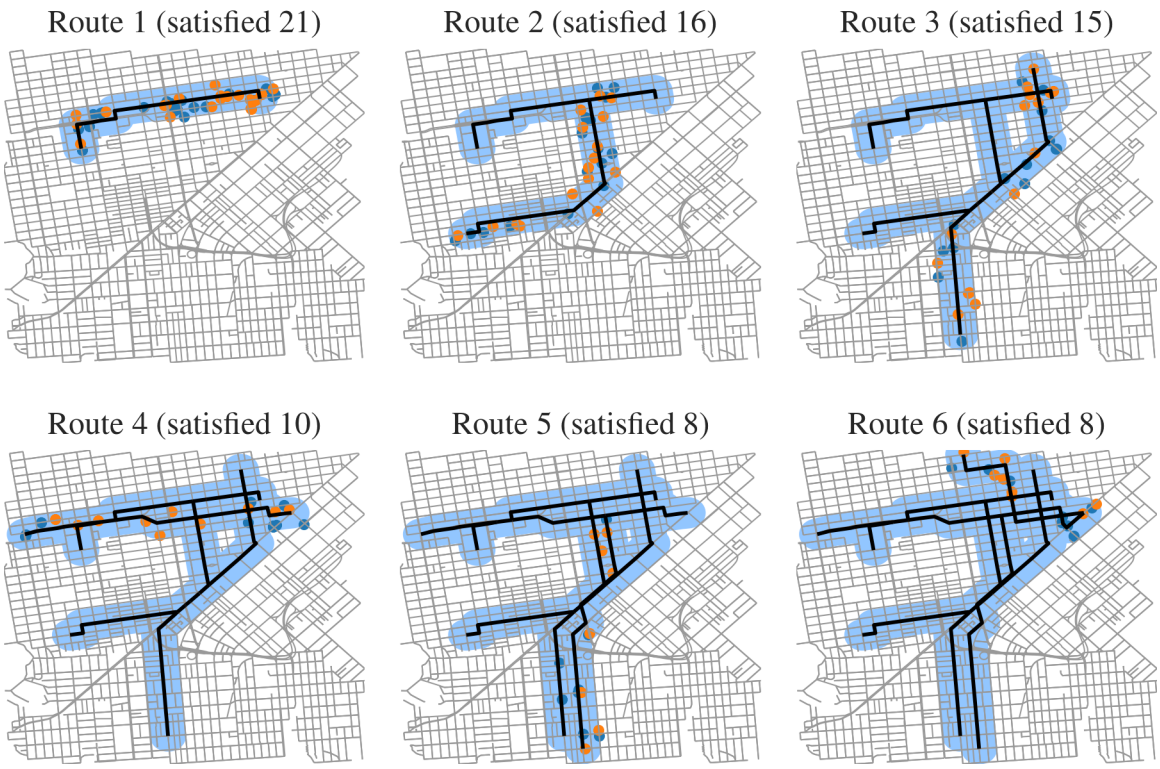


Figure 4.1: An example of the greedy submodular maximisation constructing six routes. The endpoints of users satisfied at each time step are shown. Forming six routes for the 2 km around the centre of the San Francisco dataset.

value. To find the set we iterate through all user trips and find distances from their trip endpoints to the candidate route, resulting in $O(nm)$. At lines 8–11, we save the set of maximum size. The inner loop thus has a complexity of $O(n^2m)$. At lines 13–15, we check whether we have any more satisfied users. At line 16, we remove the newly satisfied users from the dissatisfied set in $O(n^2)$. Lines 16–17 augment the current route set with the new route in $O(k)$. Thus the outer loop and the algorithm has a complexity of $O(kn^2m)$. If k is negligible, the complexity is $\boxed{O(n^2m)}$, square in the size of the dataset.

Algorithm 1 Greedy submodular maximisation selecting k routes that satisfy the most users.

```
 1: function GreedySubmodularMaximisation(trips, k, d)
 2:     dissat ← trips                                   ▷ Set of dissatisfied users
 3:     routes ← {}                                      ▷ Set of constructed routes
 4:     for i ← 1 to k do
 5:         maxSatisfied ← {}                            ▷ Set of users satisfied by current route
 6:         for r in trips \ routes do
 7:             satisfied ← {t ∈ dissat | DistP(r, t.start) ≤ d and DistP(r, t.end) ≤ d}
 8:             if |satisfied| > |maxSatisfied| then
 9:                 maxSatisfied ← satisfied
10:                 maxRoute ← r
11:             end if
12:         end for
13:         if |maxSatisfied| = 0 then
14:             break                                    ▷ Cannot satisfy any more users
15:         end if
16:         dissat ← dissat \ maxSatisfied
17:         routes ← routes ∪ {maxRoute}
18:     end for
19:     return routes
20: end function
```

4.3 Satisfaction maximisation allowing vehicle changes

At each time step, the submodular maximisation maximises the count of newly satisfied users that will be using a single newly selected route. We could also maximise the count of newly satisfied users that will be using all of the routes that have been selected up to now. To do that, we have to allow vehicle changes by walking at most d metres between vehicles.

The modification of the objective function of the algorithm to consider vehicle changes means that the algorithm is **no longer submodular maximisation** as the objective function does not give the guarantees of a submodular function. The marginal increase of its value caused by an addition of a new route might be larger as routes are added because a new route could satisfy many users that are going to travel using the new

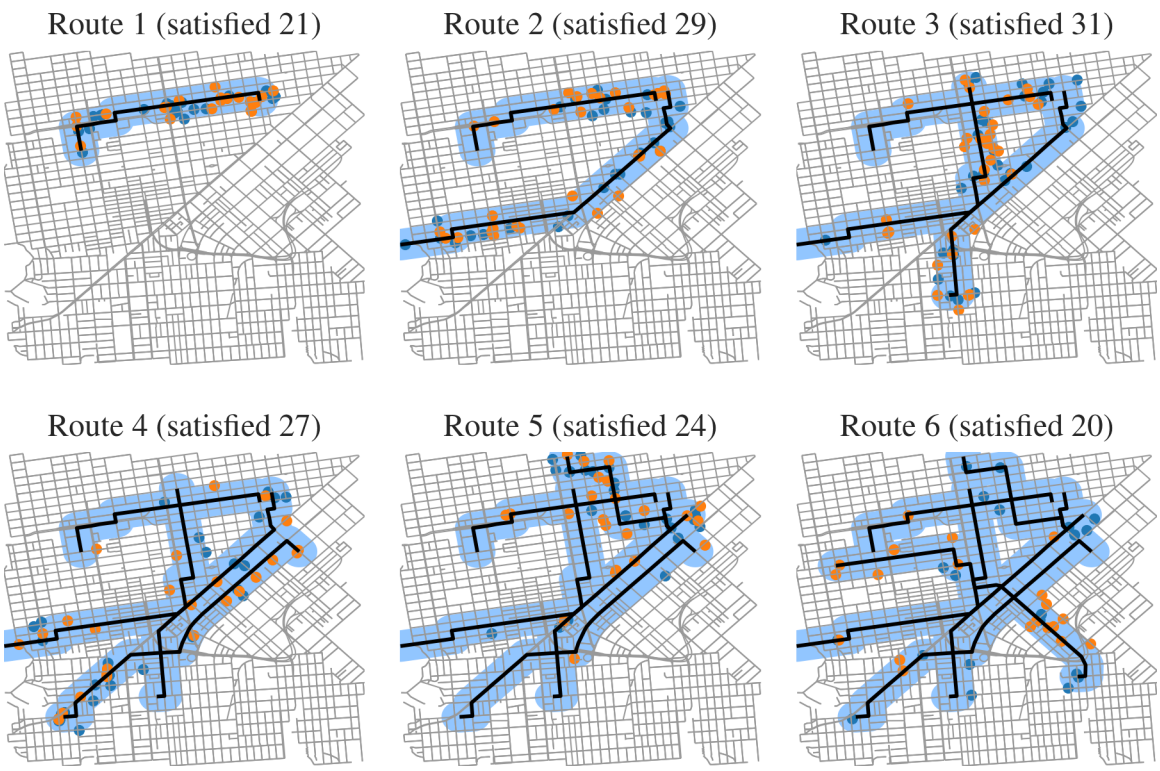


Figure 4.2: An example of the greedy maximisation constructing six routes. Additions of new routes, the trip endpoints of newly satisfied users, and the expanding area where users would be satisfied if they had their trip endpoints there are shown. Forming six routes for the San Francisco dataset 2 km around the centre of the city.

route and then change to other buses. We can no longer assume the invariant that we have found all routes that had larger marginal objective function values.

From our problem definition, a passenger is satisfied if he or she does not have to walk a distance greater than d metres in a single continuous walk. The constraint of walking d metres in a single continuous walk rather than in total in a trip arises partly because of our approach that looks at mutual reachability of routes. Two routes are **mutually reachable** if there is a path between them in a graph with the length of at most d metres. During the algorithm, newly selected routes are grouped into sets based on whether they are mutually reachable under this constraint.

Although unlikely, the maximisation could form two or more sets of routes. This would mean that a user cannot transfer between any two vehicles travelling on routes from different sets without exceeding d metres distance on foot. We left this as an option in the algorithm to make it flexible to add any route that maximises the objective function.

An example of the maximisation constructing routes is given in Figure 4.2. Compared to Figure 4.1 we can see that, as new routes are added, the users that have another endpoint at a distance from the new route are now being satisfied. Also, we see that the marginal increase of the objective function is no longer getting smaller.

The pseudocode of the greedy maximisation allowing vehicle changes for routes is

Algorithm 2 Maximisation allowing vehicle changes selecting k routes that satisfy the most users.

```
function MaximisationWithChanges(trips, k, d)
    dissatisfied ← trips                                   ▷ Set of dissatisfied users
    routeSets ← {}                ▷ Set of sets of mutually reachable constructed routes
    for i ← 1 to k do
        maxSatisfied ← {}                      ▷ Set of users satisfied by current route
        for candidate in trips \ ⋃_{S∈routeSets} S do
            reachableSets ← {S ∈ routeSets | ∃r ∈ S. DistL(r, candidate) ≤ d}
            routes ← ⋃_{S∈reachableSets} S ∪ candidate
            satisfied ← {t ∈ dissatisfied | ∃r ∈ routes. DistP(r, t.start) ≤ d and
                DistP(r, t.end) ≤ d}
            if |satisfied| > |maxSatisfied| then
                maxSatisfied ← satisfied
                maxRoutes ← routes
                maxReachableSets ← reachableSets
            end if
        end for
        if |maxSatisfied| = 0 then
            break                                          ▷ Cannot satisfy any more users
        end if
        dissatisfied ← dissatisfied \ maxSatisfied
        routeSets ← (routeSets \ maxReachableSets) ∪ {maxRoutes}
    end for
    return ⋃_{S∈routeSets} S
end function
```

shown in Algorithm 2. The DistL(l_1, l_2) function calculates the distance between two line strings by iterating through the points of both routes. Its worst-case time complexity is $O(nm)$ where n and m are the numbers of points in the two routes. In the following analysis, the DistP, n, k, and m are defined as in the previous section.

On each iteration of route selection starting at line 6, the candidate routes from the trip set that were not chosen before are evaluated. To find candidates at line 6 we check whether each trip is not already considered a route in time $O(nk)$. At line 7, we find the reachable sets of a candidate – there is at least one route in them that is reachable from the candidate route. To do that we need to iterate through all routes in route sets and call DistL, resulting in $O(km^2)$. At line 8, we put the routes in reachable sets and the candidate route into a single set, resulting in $O(k)$. At line 9, we find which users out of the dissatisfied user set are now satisfied after adding the candidate route to the system. The number of such users can be considered as the increase in the value of the objective function. This is done by iterating through the trips of dissatisfied users, going through the routes in the route set, and calling DistP twice for each route, resulting in $O(2nkm)$. At line 10, we check whether we found a route with a better objective function value than the last best candidate, it is constant time operation assuming the sets track their element count. The assignments in the conditional at line 11 are by reference, thus constant time. The loop between lines 6–15 is repeated $O(n)$ times and thus the total complexity of the loop is $O(nkm^2 + n^2km)$.

The remaining outer loop content does the following. At line 16, we check whether we have found a route that would satisfy at least a single person in constant time. If not, this means there is no additional route that would satisfy any more users. At line 19, we update the dissatisfied set by subtracting the found most satisfied set in $O(n^2)$ time. At line 20, we update the set of route sets by subtracting the sets that were merged and adding the merged set in $O(k^2 + k)$ time. The outer loop between lines 4–21 is repeated $O(k)$ times and the total complexity of the function is $O(k(k^2 + nkm^2 + n^2km)) = O(k^3 + nk^2m^2 + n^2k^2m)$.

The complexity of the algorithm if the number of routes k is negligible compared to n and m is $\boxed{O(nm^2 + n^2m)}$, quadratic in both the number of trips and points in a line string.

We also have implemented another approach based on merging polygons. The polygons are formed as a buffer of d metres around routes. A trip endpoint falling within polygons would mean that the associated user is satisfied. However, such approach is more computationally complex, as the distance calculations would be replaced by the point within polygon queries.

4.4 Performance of the methods

As seen in Figure 4.3, the maximisation with changes satisfies more users than the simple submodular maximisation during the execution of the algorithm when we look at the same user trip set using which the algorithm is constructing routes. Also, the

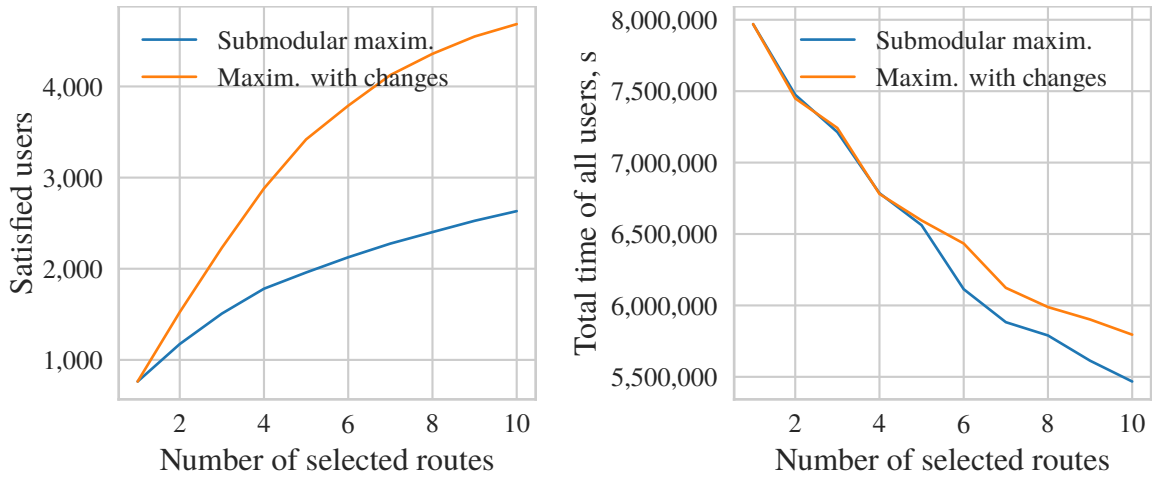


Figure 4.3: Left: increasing the number of satisfied users by adding more routes. Right: the total time users spend travelling when the increasing number of routes. San Francisco, 2 km radius, 5,000 trips, 200 m max. walk distance.

total time users spend travelling decreases linearly with each added route. Our method to calculate total time will be introduced in Section 6.2.

We note that the worst-case time complexity of both methods is square in the number of trips. Thus we want to look further for an alternative way of constructing the routes.

Chapter 5

Route construction in angular dual space

Following the route construction approach using the submodular optimisation in the previous chapter, we present an algorithm for route construction that would be unrelated and could be compared to it. This algorithm is based on dual space transformation and clustering.

5.1 Similarity in primal space

We approach the cluster formation problem of finding trips having a similar position in space and direction between endpoints. This would find a few directions that capture the travel patterns of the most users. We want to express the position and direction of trips as numerical values and have close values for similar positions and directions.

The distances in the road networks are similar to the distances in the Euclidean plane. 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 by Viana et al. [2016]. The authors conclude that "people tend to be oriented by the shortest paths" for transportation modes where people have "decision control". For this reason, we can convert trips, which are trajectories, into trip segments between pickup and dropoff endpoints. Then we want to aggregate trips that are roughly on the same straight line.

The approach to naively cluster passenger trip endpoints (x_0, y_0, x_1, y_1) in a 4-dimensional plane fails to detect partially overlapping trips and slight differences in the direction of trip segments as seen in Figure 1.2. But exactly those situations are important for the route construction problem because those passengers can still travel using the same vehicle. The naive clustering only detects the situation when the respective endpoints of trips are close together.

To alleviate this we can design a custom clustering distance metric to aggregate trips, which would take into account the angle between two trip segments $m = (A_m, B_m)$,

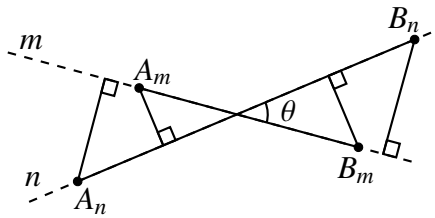


Figure 5.1: Components of the similarity measure for the two line segments m and n.

$n = (A_n, B_n)$ and a notion of distance between them. Also, it should be simple to calculate to be able to compute all pairwise distances during clustering.

We define the following similarity measure for two directed line segments based on the angle and mutual endpoint distances shown in Figure 5.1:

$$\mathrm{S_{AP}}(m, n) = \frac{\cos\theta}{1 + \max\{\mathrm{D}(A_m, n), \mathrm{D}(B_m, n), \mathrm{D}(A_n, m), \mathrm{D}(B_n, m)\}}. \tag{5.1}$$

Here, $\mathrm{D}(X, l)$ is the shortest distance between a point X and line l. The angle θ is the smaller angle between the two line segments, $\theta \in [0, \pi]$ and $\cos\theta \in [0, 1]$. The term in the denominator $\max\{\cdot\} \in [0, +\infty)$ and is zero when the segments are identical to each other, thus there is a constant term added to the denominator. Therefore the function range is $[0, 1]$.

The similarity measure is commutative: $\mathrm{S_{AP}}(m, n) = 1$ iff m and n are on the same line. It is also symmetric: $\mathrm{S_{AP}}(m, n) = \mathrm{S_{AP}}(n, m)$.

The intuition for this similarity measure is as follows. Two trip segments are more similar when the endpoints of both segments are close to another segment, this is reflected in the denominator. Also, they are more similar when the angle between them is small, this is reflected in the nominator, e.g. the nominator is one when two lines are collinear and zero when they are perpendicular to each other. The similarity is equal to one when two segments are identical and to zero when they are perpendicular to each other.

This similarity measure was constructed to find trips that are close to a common route. Particularly, we wanted to make sure that the similarity reflects the edge cases in Figure 5.2. Two segments can be similar if they are close to a common route even when they do not intersect as seen in (a). The similarity is proportional to the distance between segments as seen in (b) and (c). Similarity also decreases with increasing angle between segments as seen in (c).

To use this similarity measure in clustering algorithm we convert it to a distance measure using $\mathrm{D_{AP}}(m, n) = 1 - \mathrm{S_{AP}}(m, n)$. The k-medoids algorithm is used as explained in the next section.

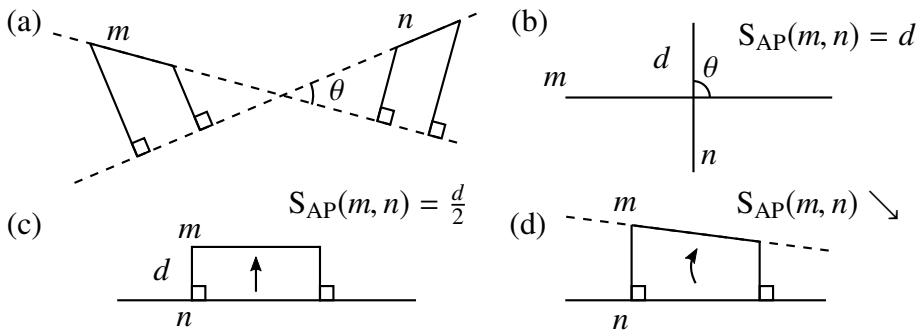


Figure 5.2: Edge cases of the similarity measure: (a) two segments do not intersect but close to the common route; (b) intersecting but orthogonal; (c) parallel to each other; (d) increasing the angle between the trips.

5.2 Clustering algorithms

The goal of clustering is to minimise the inter-class similarity and maximise the intra-class similarity. For the route construction task, we want to form k clusters. There are several clustering algorithms that can be used.

k-means. The aim of forming k clusters is to partition the data points into k clusters in which each data point belongs to a single cluster whose representative point is closest. The k-means problem uses the centroid as a representative of the cluster. Solving k-means optimally is NP-hard. The heuristic commonly used for k-means is called Lloyd's algorithm. It repeatedly reassigns data points to clusters with the closest centroids and then calculates new centroids of clusters. This algorithm has a worst-case time complexity of $\boxed{O(nkdi)}$ where n is the number of d-dimensional data points and i is the number of iterations needed for convergence [Ng and Han, 1994]. If the dataset has a 'clear' clustering, i can be negligible, and the algorithm is linear in the number of data points.

The k-means algorithm minimises the intra-cluster variance which is the sum of squared Euclidean distances. In the assignment step, k-means uses distance measure between the data points and centroids. In the centroid update step, k-means calculates the mean. For k-means to have a convergence guarantee after a finite number of iterations, the mean from the centroid update step must minimise the distances to centroids from the assignment step. This also means that we cannot use a non-Euclidean distance in the centroid update step and guarantee the convergence. Instead, for an arbitrary distance measure, we would have to use a different algorithm instead, like k-medoids.

k-medoids. The k-medoids problem is related to k-means but returns a data point as the cluster representative. It is also NP-hard. The heuristic commonly used for k-medoids is the Partitioning Around Medoids (PAM) algorithm introduced by Kaufman and Rousseeuw [1990]. The cluster representative update step minimises the sum of distances from a candidate cluster representative to each data point assigned to a cluster. As we are no longer using the mean, the distance measure used in both steps can be non-Euclidean, such as our primal distance measure.

The complexity of PAM is $\boxed{O(k(n-k)^2 di)}$ as shown by Ng and Han [1994]. It is square in the number data points. However, there are more recent heuristics for k-medoids problem. These include those that are based on PAM, such as CLARA based on sampling and CLARIANS based on randomised search; and also unrelated to PAM, such as an algorithm based on distance matrix that has the same complexity as k-means [Park and Jun, 2009]. See Park and Jun [2009] for an overview of recent developments.

5.3 Angular dual space

However, while k-medoids clustering in the primal space works in the example cases, it does not perform well using real trajectory datasets, as shown during evaluation in Section 6.3.

The approaches in the background literature used clustering in similar but not exactly the same setting. The approach that we take is to transform the space of the trip coordinates in a way that preserves and quantifies the position and direction of the trips.

We choose to transform the trips using the **angular dual space** which has several useful properties for our task. The transformation of a directed line segment AB into a pair of angles (α, β) is depicted in Figure 5.3. First, we define a circle around origin O that covers all segments of radius r. Then the trip segment AB is extended to a line. The two points of intersection between the circle and line are denoted A' and B'. The two angles that are formed at circle centre O between those points A' and B' and the x-axis are called (α, β) such that $\alpha \leq \beta$.

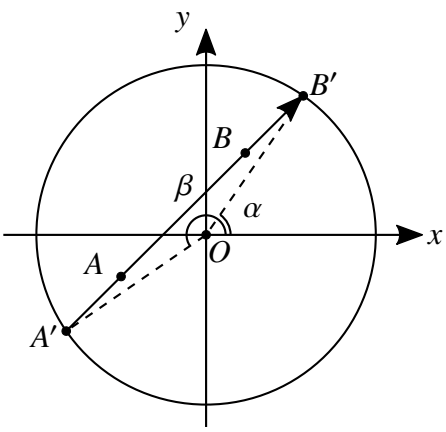


Figure 5.3: Transformation from a directed line segment AB in the primal space into a pair of angles (α, β) in the dual space.

We consider our segments representing trips to be directionless because we are building routes that will serve users travelling in both directions. We need to order $\alpha \leq \beta$ because otherwise each segment is represented twice as a point in the angular space $\alpha, \beta \in [0, 2\pi)$, because the distinction between α and β is arbitrary. Then the (α, β) points are not distributed throughout this whole rectangular portion of space but only in the upper left triangle above $\alpha = \beta$ line. This is a way to establish a single representation

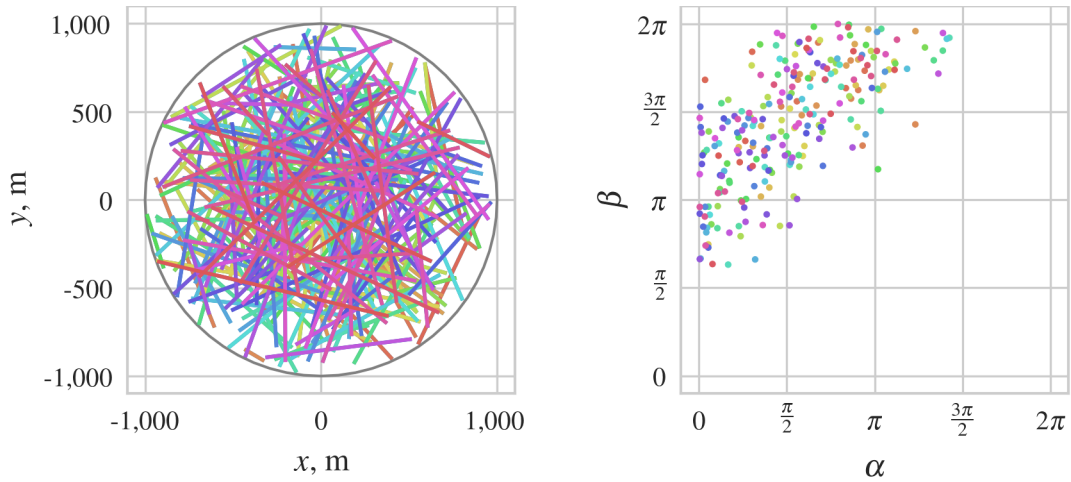


Figure 5.4: Randomly placed 500 segments in the primal space and their corresponding points in the dual space after sorting $\alpha \leq \beta$.

of points in the dual space. The Figure 5.4 shows 500 random segments as points in the dual space after sorting.

We can observe several properties of the angular dual space. There are not going to be points close to $\alpha = \beta$ line, because these correspond to infinitely short segments near the perimeter of the circle. Concurrent lines in the primal space correspond to points on the same curve in the dual space. Moreover, collinear intersection points in the primal space correspond to curves intersecting at a single point in the dual space. An example of both of these properties can be seen in Figure 5.5.

An example of the described geometric construction applied on the San Francisco dataset can be seen in Figure 5.6.

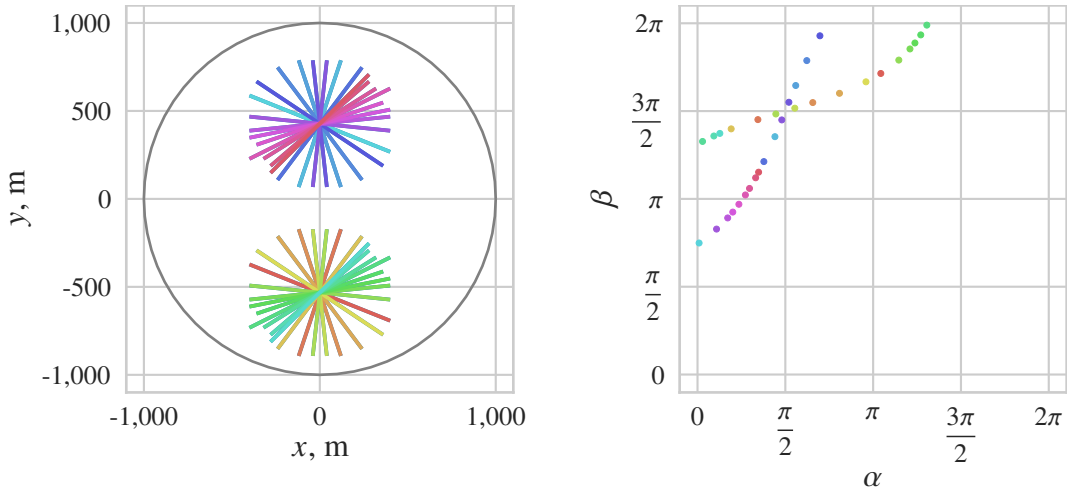


Figure 5.5: Concurrent lines in the primal space correspond to points on the same curve in the dual space.

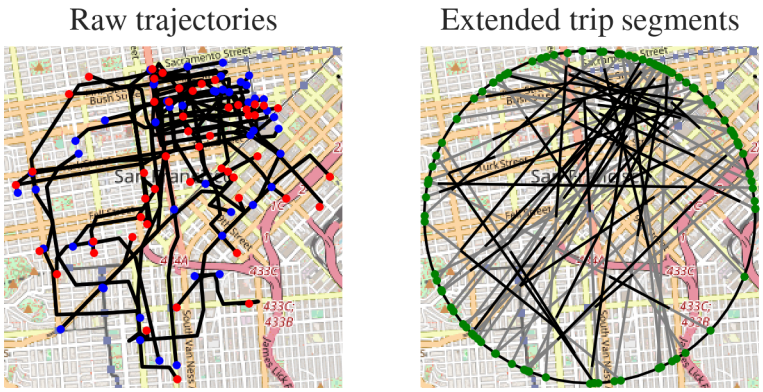


Figure 5.6: An example of 50 trajectories in San Francisco extended into lines to find their representation in the angular dual space.

5.4 Clustering in angular dual space

For our San Francisco example, the angular dual space and the cluster assignments are shown in Figure 5.7. The algorithm for route construction in the angular dual space described in the previous section consists of the following:

1. find a circle that covers all segments,
2. extend the trip segments to lines and find the line–circle intersection points,
3. find the two angles in the dual space corresponding to the points, and
4. find k clusters using a clustering algorithm with a Euclidean-like distance adapted to the dual space, and take the centres of each cluster.

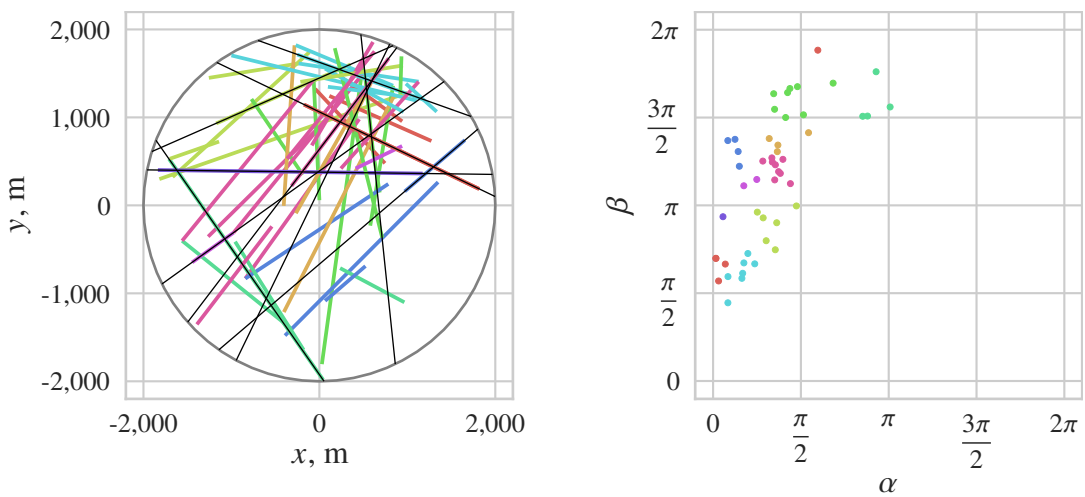


Figure 5.7: The same 50 trips segments in San Francisco clustered in the angular dual space. The colour represents the cluster assignments and lines in black are the cluster centres.

5.5 Clustering distance metric

Now we describe the details of the distance metric between the two points used for clustering in the angular dual space.

As explained in Section 5.3, a point in the angular dual space is described by a pair of angles (α, β). Using angles results in the topology of the space to become a torus with a wrapping around boundary at $\alpha' = 0 = 2\pi$, $\beta' = 0 = 2\pi$. This plane is still Euclidean locally but not globally. The shortest distance between two points $A = (\alpha_0, \beta_0)$ and $B = (\alpha_0, \beta_0)$ might go through that boundary. To account for this in the clustering algorithm we modify the Euclidean distance to choose either the distance between 0 and 2π or the one that wraps around at $0 = 2\pi$ resulting in

$$\mathrm{D}(\alpha_0, \beta_0, \alpha_1, \beta_1) = \sqrt{\min(|\alpha_0 - \alpha_1|, 2\pi - |\alpha_0 - \alpha_1|)^2 + \min(|\beta_0 - \beta_1|, 2\pi - |\beta_0 - \beta_1|)^2}.$$

However, this is not enough for useful results. As also explained before, because the segments do not have a direction in our setting, we sort the angles such that $\alpha \leq \beta$ to establish a single order. This results in all points moving from the lower right triangle to the upper left triangle symmetrically around the $\alpha = \beta$ line when we plot β against α. The issue is that there could be a point moved this way near the line $\alpha = 0$ and a point near the line $\beta = 1$ that would be otherwise close together because of the torus topology. An example of such case is shown in Figure 5.8. One solution would be the duplication of points into both triangles, but that would result in two cluster assignments for each point. Therefore, we choose to change the distance measure to find the distance between points as if they were mirrored around $\alpha = \beta$ line:

$$d_{AB} = \min\{\mathrm{D}(\alpha_0, \beta_0, \alpha_1, \beta_1), \mathrm{D}(\alpha_0, \beta_0, \beta_1, \alpha_1)\}.$$

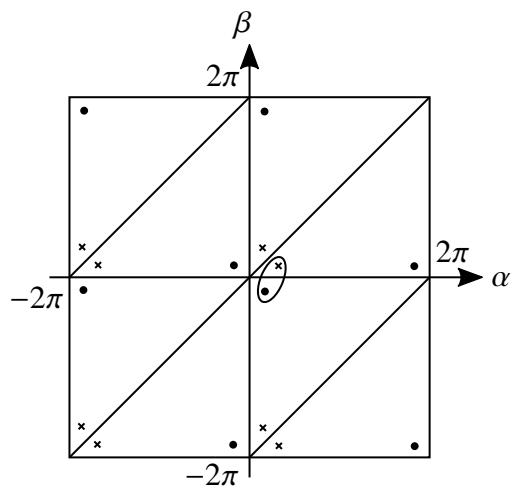


Figure 5.8: Visualisation of the repeating plane because of the torus topology and the symmetry around $\alpha = \beta$ axis. The shortest distance between the two points indicated as cross and dot resulting from the topology and symmetry is shown.

5.6 Cluster properties

Now we present some observations about the resultant clusters. When segments are in a similar position in the primal space and have a similar but varying slope, the points in the dual space are located in an elongated cluster. An example of clustered points and associated line segments can be seen in Figure 5.9.

When the radius of the circle is increased, the points are spread out in the dual space more closely and it is harder to distinguish them to form clusters. An example of the same set of segments is given in Figure 5.10, which shows a change of the point pattern in the dual space after increasing the radius of the circle. This means that we have to be careful to set the circle radius that would be appropriate for the area.

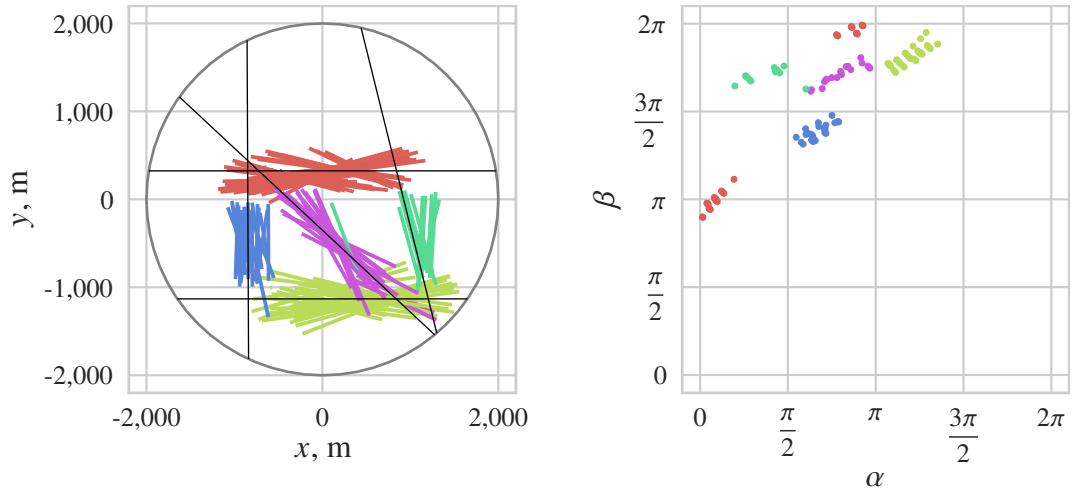


Figure 5.9: An example set of line segments in the primal plane (left), their corresponding points in the angular dual space (right), cluster assignments indicated in colour, and resulting routes in black. Using 2 km circle radius.

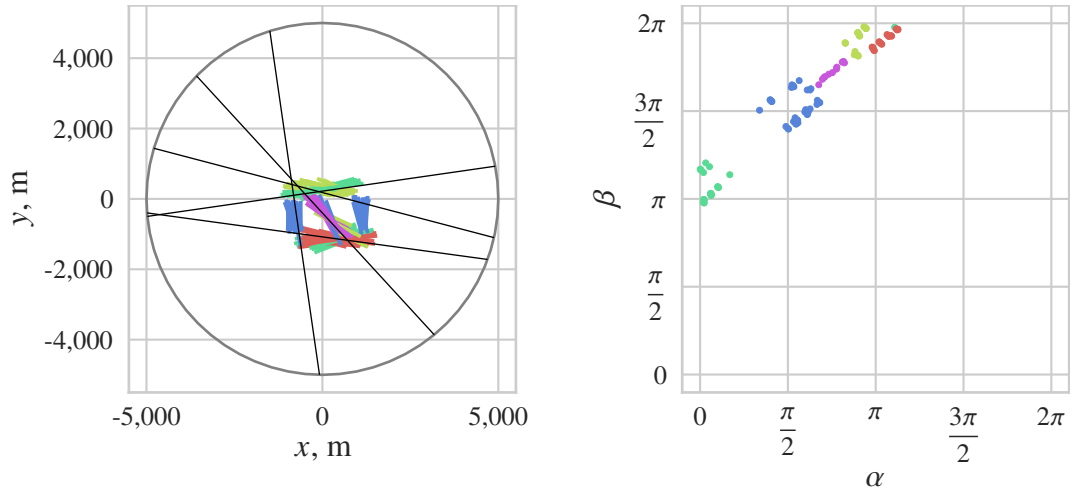


Figure 5.10: The same example set of lines clustered using the angular dual transform with an increased circle radius of 5 km.

Chapter 6

Evaluation

In this chapter, we introduce the baselines used for evaluation, evaluation measures, evaluation setup, and evaluate our proposed route construction algorithms.

The pipeline of the experiments consists of the following:

1. constructing a graph for each city representing a map network using the OpenStreetMap data,
2. preprocessing New York, Rome, and San Francisco datasets and map trajectories into graph paths,
3. constructing the baseline datasets,
4. performing angular dual transform of trip segments from the datasets and cluster the resulting points,
5. performing submodular optimisation of trajectories from the datasets,
6. performing routing in the graph between the endpoints resulting from the two methods, and
7. calculating the total time and satisfied ratio metrics.

We keep the trips represented as both the trajectories and graph paths. After matching the trajectories to graph paths, the trajectories are updated to have the same coordinates as the nodes in the graph path. The angular dual transform takes into account solely the endpoints of the trajectories, however, because we use k-medoids, we take the real trips corresponding to the cluster centres as routes. The submodular optimisation uses the whole trajectories. Therefore our resultant routes are always real and plausible trajectories.

6.1 Baselines

We have developed two baselines to be able to compare the values of the evaluation measures between them and our proposed methods.

The first baseline is k routes constructed using naive hierarchical Euclidean clustering of trajectory endpoints (x_0, y_0, x_1, y_1) in a 4-dimensional plane. This method does not construct reasonable routes because it looks at the source and destination endpoints separately. This idea is described in Section 1.1.

The second baseline is placing k equidistant routes vertically and horizontally. We do this geometrically in the bounding box of width w and height h. We place $\lceil \frac{w}{w+h} \cdot k \rceil$ lines vertically and $\lfloor \frac{h}{w+h} \cdot k \rfloor$ lines horizontally equidistant from each other. Then we find the nodes in the graph that are closest to the endpoints of such lines. We use the Dijkstra's algorithm to find the paths between the nodes.

We also define a walking baseline such that we do not construct any routes at all and can calculate the total time users spend if they choose to walk.

6.2 Evaluation measures

We define two evaluation measures that stem from our research question. First, we quantify the utility of the transportation system to users by calculating the total time all users spend using the system. Second, we quantify the satisfaction with the system by calculating the ratio of users that do not have to walk more than k metres in a single contiguous walk.

6.2.1 Total time

The total time all users spend using the system is calculated in a graph setting. As mentioned in Section 3.2, an edge in the graph is weighted by the length of the corresponding road between intersections. We find the total walking time by summing the lengths of all paths in the dataset divided over the walking velocity set to 1 m/s. We find those lengths using the Dijkstra's algorithm.

Then we find the total time spent using the constructed routes with changes between vehicles allowed for each set of constructed routes. We give the edges new weights representing time to travel across that edge. If that edge is part of any constructed vehicle path, it is given the weight of its original length divided by the vehicle velocity set to 9.2 km/h = 2.56 m/s. If it is not, we give it the weight of its original length divided by the walking velocity. Then we apply the Dijkstra's algorithm using the new weights between each pair of trip path endpoints which gives us the travel time.

6.2.2 Satisfied ratio

The number of satisfied users represents the users that do not have to walk more than d metres in a single continuous walk.

We calculate the satisfied ratio by converting the routes as graph paths to their geospatial representation as line strings. Then we apply the *buffer* operation on them, which

constructs polygons that consist of area d metres around the line strings. This represents the area that would satisfy the users if their trajectory endpoints would fall within it. We apply the *within* operation for each trip endpoint pair on the union of all areas. Thus we get the number of satisfied users given a set of routes.

6.3 Evaluation results

6.3.1 Travel time evaluation

The plots of the total time cumulative distributions are means to visualise the number of users that have completed their trips after a certain amount of time has passed. In Figure 6.1 we show the distribution of the user travel times that use our maximisation methods (greedy submodular maximisation and maximisation with changes), clustering methods (k-medoids clustering in the primal and angular dual planes), and three baselines (equidistant routes, 4-dimensional Euclidean clustering, and walking) for the San Francisco dataset.

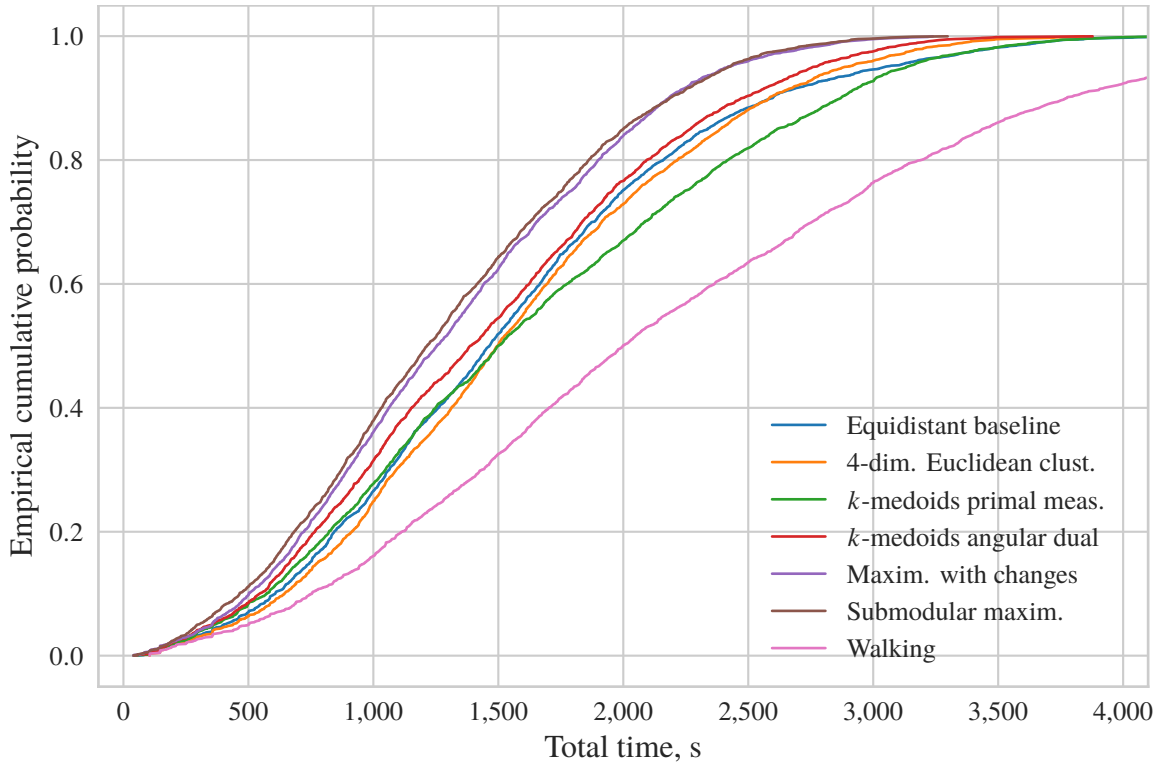


Figure 6.1: Empirical cumulative distributions of total time spent in the system using different methods for the construction of 6 routes. Results for 5,000 taxi trajectories in San Francisco, 2 km around the city centre.

We observe that the maximisation methods obtain the best total travel times for all users, even though they are not explicitly maximising the total time. The k-medoids clustering in dual space is close to the baselines. Moreover, the clustering in the primal space

Method	San Francisco (h)	Rome (h)	New York (h)
Walking	3,063	5,609	2,220
Equidistant baseline	1,990	2,252	1,772
4-dim. Euclidean clust.	1,971	2,270	1,681
k-medoids primal measure	2,309	2,253	2,065
k-medoids angular dual	2,125	1,934	1,678
Submodular maxim.	**1,519**	**1,604**	**1,289**
Maxim. with changes	1,610	1,791	1,398

Table 6.1: Total time in hours spent in the system using different methods and datasets. San Francisco, 2 km radius, 5,000 trips, 10 routes, max. walk length 200 m.

using a custom distance measure performs worse than the two baselines for around half of the trips. This suggests that there are certain travel patterns that our custom distance measure does not capture.

We have two hypotheses why the maximisation methods are able to perform visibly better than the angular dual space and primal space clustering methods.

- The maximisation is a greedy method trying to maximise the number of satisfied users and provide a satisfying route to one specific subset of the users localised on the map. This is in contrast to the clustering approaches that try to fit to the whole dataset looking for a global approximation.

- The disadvantage of the clustering approaches is that they try to satisfy all users and do not allow disadvantaging a subset of users that have travel patterns that are very dissimilar to the majority of the users. However, from the CDF graph, we do not see that the submodular optimisation methods would particularly disadvantage a subset of users as their curves are to the left of the curves of other methods throughout the range of all users.

Another way to visualise the total time distribution is a box plot of total times when travelling using the routes from different methods shown in Figure 6.2. It depicts the distribution of the total time through the quartiles. Boxes represent the inter-quartile ranges (IQR) between the first and third quartiles, the median is shown inside as a line. Whiskers extend to $1.5 \cdot$ IQR beyond the first and third quartiles. Outliers beyond them are shown as points. These indicate users that have long travel times potentially because of their unusual trajectories that are not served by the constructed routes. The IQR and median of the k-medoids angular dual clustering and submodular maximisation are slightly below the two baselines (equidistant and 4-dimensional clustering), meaning that some of the trips have shorter total times when using our constructed routes.

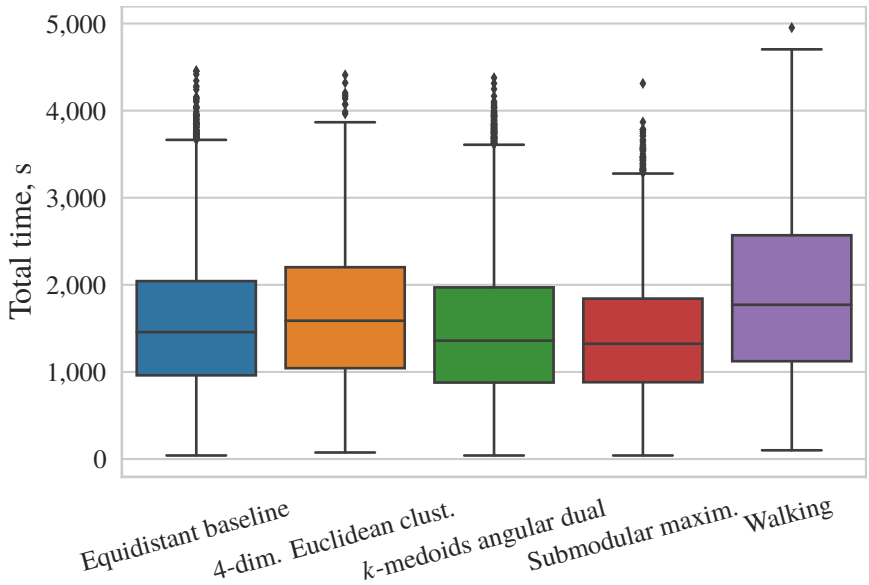


Figure 6.2: A box plot showing travel time distributions using different methods and baselines. San Francisco, 2 km radius, 10,000 trips, 6 routes, max. walk length 200 m.

	San Francisco		Rome		New York	
Method	Ratio	Count	Ratio	Count	Ratio	Count
Equidistant baseline	0.676	3,380	0.768	3,840	0.632	3,160
4-dim. Euclidean clust.	0.234	1,168	0.348	1,740	0.322	1,610
k-medoids primal measure	0.126	632	0.319	1,597	0.240	1,199
k-medoids angular dual	0.348	1,740	0.587	2,935	0.552	2,758
Submodular maxim.	0.805	4,027	0.838	4,189	0.930	4,649
Maxim. with changes	**0.937**	4,684	**0.967**	4,833	**0.990**	4,950

Table 6.2: Ratios of satisfied users using different methods and datasets. 10 routes 2 km radius around the respective city centre using 5,000 trips, max. walk length 200 m.

6.3.2 Satisfied ratio evaluation

As seen from Table 6.2, the submodular maximisation achieves the highest satisfied ratio. This is because the submodular maximisation is directly maximising the number of satisfied users. These numbers depend on the chosen maximum walking distance and the extent of the dataset. This is seen from the fact that just by placing equidistant routes achieves a comparable satisfied ratio.

6.3.3 Runtime evaluation

We can evaluate the runtime of the proposed methods. The calculations in this evaluation were performed using DICE lab computer which has the following specifications: Intel Core i5-6500 CPU @ 3.20GHz × 4 processor, 15.6 GiB memory, and Scien-

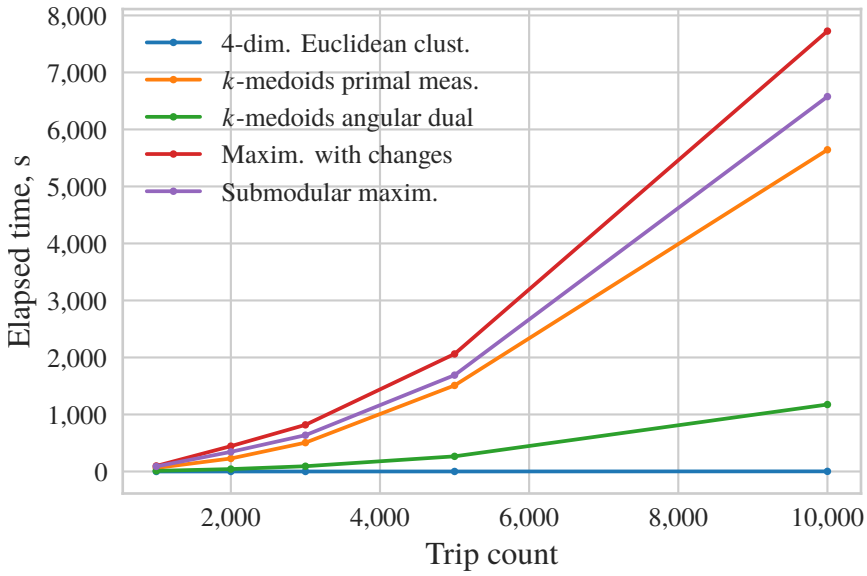


Figure 6.3: Elapsed system-wide time during clustering and submodular methods on different dataset sizes. San Francisco, radius 2 km, 6 routes, max. walk length 200 m.

Method	San Francisco (s)	Rome (s)	New York (s)
4-dim. Euclidean clust.	0.009	0.009	0.011
k-medoids primal measure	1422.459	1396.990	1415.395
k-medoids angular dual	536.260	516.350	532.513
Submodular maxim.	2549.670	2510.446	2340.270
Maxim. with changes	2600.856	2709.938	2018.878

Table 6.3: Running time in seconds of the proposed algorithms on different datasets. Constructing 10 routes 2 km radius around the respective city centre using 5,000 trips.

tific Linux 7.4 operating system. The elapsed time was calculated using Python 3.6 *time.perf_counter()* function. The difference between the returned values provides the elapsed time in fractional seconds measuring the system-wide time.

As seen before, the worst-case time complexity of the submodular maximisation is asymptotically square in the number of input trajectories. The clustering approaches that use k-medoids are also square in the number of input trajectories. In practice, they find the solutions quicker than the submodular optimisation because they do not perform geospatial operations on each point in the trajectories, which adds a large constant factor to the time complexity. Also, using other heuristics for k-medoids problem could make the algorithm more computationally effective. As shown in Table 6.3, the submodular optimisation takes relatively long time compared to clustering in the angular dual space. A high increase in the elapsed time for the submodular optimisation relative to other methods when increasing the trip count can be seen in Figure 6.3. This suggests that, while submodular optimisation produces better routes, the dual space clustering might be more suitable for route construction on large datasets and travel areas.

Chapter 7

Conclusion and further work

7.1 Conclusion

This work presented two new methods for route construction in a road network given the travel trajectories when our objective is user satisfaction. We define user satisfaction twofold: as the minimum time spent travelling and as not having to walk a large distance to a vehicle.

The first method is based on submodular optimisation, where we greedily maximise the ratio of satisfied users by choosing paths that provide the largest increase in the number of users that walk less than d metres. This method produces reasonable routes although it takes a substantial time to execute. Setting an appropriate maximum walking distance allows this method to be used for various scales and modes of transport, from city-wide bus networks to country-wide rail networks, and various users, for example, passengers that can and cannot travel far on foot.

The second method is based on the angular dual space transform and clustering in dual space. The advantage of this method is being faster than the submodular optimisation and thus it could be used on larger datasets. This aspect is also important in dynamic situations when the datasets change and routes should be recomputed, for example, in response to changes in the road network, demand, population, or season. Another advantage is that there is only a single hyperparameter, the radius of the circle, that needs to be set to construct routes. The disadvantage is that the method tries to satisfy all users and not the majority, which is a problem because there is no clear clustering of all the trips.

As a more general result, this work proposed a method to cluster trajectories converted to line segments using a duality transform. The proposed duality method could also be useful for other types of applications, such as travel pattern discovery and image processing.

7.2 Improvements

We have some ideas that would build on and improve on route construction using the angular dual space.

The transformation to the angular dual space is an interesting mathematical concept which could be further researched. The angle values found by using a single large circle are more uniformly distributed throughout the dual space, have less clear clustering, and that negatively affects the quality of the constructed routes. It would be interesting to investigate a method in which the area would be subdivided into multiple circles. Clustering using these subproblems separately would then be used to obtain the coverage of the whole area. A larger area in the primal space covered with multiple small circles would prevent analysing unclear clusters. However, it would raise the question how to combine the routes constructed by solving these subproblems into a single solution.

Another idea is using a different clustering method. An important observation is that the clusters in the angular dual space have an elongated shape. This suggests that a density-based clustering method, such as DBSCAN, could be used. An issue with DBSCAN is the setting of its parameters *eps* and *minPts* which would need to be calibrated for a particular travel demand pattern and circle radius. Moreover, DBSCAN does not have a cluster count parameter and could mark most points as outliers if it cannot distinguish clear clusters. This could be solved using an outlier detection method during the preprocessing of the dataset. Another idea for a different clustering method is using Mahalanobis distance with k-means clustering because it inherently produces elliptically shaped clusters.

Bibliography

Geoff Boeing. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

Lorenzo Bracciale, Marco Bonola, Pierpaolo Loreti, Giuseppe Bianchi, Raul Amici, and Antonello Rabuffi. CRAWDAD dataset roma/taxi (v. 2014-07-17). Downloaded from `https://crawdad.org/roma/taxi/20140717`, July 2014.

Anne-Sarah Briand, Etienne Côme, K Mohamed, and Latifa Oukhellou. A mixture model clustering approach for temporal passenger pattern characterization in public transport. *International Journal of Data Science and Analytics*, 1(1):37–50, 2016.

Etienne Come, Njato Andry Randriamanamihaga, Latifa Oukhellou, and Patrice Aknin. Spatio-temporal analysis of dynamic origin-destination data using latent Dirichlet allocation: Application to Vélib' bike sharing system of Paris. In *TRB 93rd Annual meeting*. Transportation research board, 2014.

Michiel de Hoon, Seiya Imoto, and Satoru Miyano. The C clustering library. *Institute of Medical Science, Human Genome Center, University of Tokyo*, page 11, 2003.

David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2): 112–122, 1973.

Lang Fan and Christine L Mumford. A metaheuristic approach to the urban transit routing problem. *Journal of Heuristics*, 16(3):353–372, 2010.

Song Gao, Jiue-An Yang, Bo Yan, Yingjie Hu, Krzysztof Janowicz, and Grant McKenzie. Detecting origin-destination mobility flows from geotagged tweets in greater Los Angeles area. In *Eighth International Conference on Geographic Information Science (GIScience'14)*, 2014.

Diansheng Guo, Xi Zhu, Hai Jin, Peng Gao, and Clio Andris. Discovering spatial patterns in origin-destination mobility data. *Transactions in GIS*, 16(3):411–429, 2012.

Leonard Kaufman and Peter J Rousseeuw. Partitioning around medoids (program PAM). *Finding groups in data: an introduction to cluster analysis*, pages 68–125, 1990.

Dheeraj Kumar, Huayu Wu, Yu Lu, Shonali Krishnaswamy, and Marimuthu Palaniswami. Understanding urban mobility via taxi trip clustering. In *Mobile Data Management (MDM), 2016 17th IEEE International Conference on*, volume 1, pages 318–324. IEEE, 2016.

Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604. ACM, 2007.

Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization. *Journal of Machine Learning Research*, 17(238):1–44, 2016.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 144–155, 1994.

Mahmoud Owais, Ghada Moussa, Yousef Abbas, and Mohamed El-Shabrawy. Simple and effective solution methodology for transit network design problem. *International Journal of Computer Applications*, 89(14):32–40, 2014.

Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341, 2009.

Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. CRAWDAD dataset epfl/mobility (v. 2009-02-24). Downloaded from `https://crawdad.org/epfl/mobility/20090224`, February 2009.

Daniela Rus, Emilio Frazzoli, Patrick Jaillet, Ruofei Ouyang, Bryan Low, David Hsu, and Marcelo H. Ang. Jr. Clustering origin-destination pairs with Gaussian mixture models, 2011.

Aman Verma and Lynette Cheah. Extracting vehicle trip patterns using a radial-based clustering method. In *Transportation Research Board 95th Annual Meeting*, number 16-3514, 2016.

Aline Carneiro Viana, Eduardo Mucceli, Carlos Sarraute, Jorge Brea, and José Ignacio Alvarez-Hamelin. On the regularity of human mobility. *Pervasive and Mobile Computing*, 2016.

Chris Whong. FOILing NYC's taxi trip data. Downloaded from `https://chriswhong.com/open-data/foil_nyc_taxi/`, March 2013. Accessed 10 Jan 2018.

Ow Yi Xian, Mandar Chitre, and Daniela Rus. An approximate bus route planning algorithm. In *Computational Intelligence in Vehicles and Transportation Systems (CIVTS), 2013 IEEE Symposium on*, pages 16–24. IEEE, 2013.

Xi Zhu and Diansheng Guo. Mapping large spatial flow data with hierarchical clustering. *Transactions in GIS*, 18(3):421–435, 2014.