Linköpings universitet
Institutionen för teknik och naturvetenskap (ITN)

Master's thesis

Traffic flow estimation from cellular network data

by

Nils Breyer

September 14, 2015

Supervisor: David Gundlegård

Examiner: Clas Rydergren

Abstract

Today, usually mathematical models are used to estimate the traffic flows in a transportation network based on two datasources: travel surveys and link counts from sensors. Both of these datasources are expensive to obtain and therefore limited to a small group of participants for surveys or a small subset of the roads for link count sensors. This thesis investigates the possible use of cellular network data to estimate the flows on the roads, which could be a relatively cheap large-scale datasource. The method presented in this thesis contains a complete method that aims to estimate link flows only based on cellular network data which includes the extraction of trips, the estimation of the travel demand, recovering the routes used by users and a network loading.

The validation of the method shows that cellular network data helps to recover the original route that a user used, but the Lazy Voronoi Routing algorithm presented in this thesis is not able to recover the original routes as precise as it would be necessary to get reliably link count estimations only from cellular network data. The combination of cellular network data together with classic datasources like link counts however, could help to improve traffic flow estimations.

Keywords: traffic modelling, network loading, cellular network, call detail records, geospatial algorithms

Acknowledgements

Bringing my studies to an end with this thesis, I want to express my gratitude to everyone that supported me during my studies and especially during the process of writing this thesis. Besides the the Californian weather that kept my motivation up, this would not have been possible without the support of all of you.

This includes my supervisor David Gundlegård and my examininer Clas Rydergren, by providing inspiration and continous feedback during my work. I would also like to thank Alexandre M. Bayen for the inviation to collaborate with the Megacell group at UC Berkeley, which was a great honor. I want to thank everyone in the group for making my stay in Berkeley possible and as pleasant as possible. Special thanks go to Alexey Pozdnukhov for providing the necessary data used in this thesis as well as Jerome Thai, Cathy Wu, Chenyang Yuan for fruitful discussions and technical help. I'm looking forward to further collaborations with you.

I also want to thank Ångpanneföreningens Forskningsstiftelse for the scholarship that made my travel to Berkeley possible by covering a big part of the costs. Last but not least I want to thank my parents Gerd Breyer and Marita Breyer for their unconditional support during my studies.

Acronyms

CDR Call Detail Record 10, 13, 14, 15, 24, 26, 27, 31, 32, 36, 37, 38, 39, 42, 43, 60, 61, 62, 70, 71, 83, 87

D4D Data for Development 61

GIS Geographic Information System 18, 33, 34

GPS Global Positioning System 10, 26, 27

IPFP Iterative Proportional Fitting Procedure 16, 17, 45

L.A. Los Angeles, USA 34, 41, 45, 48, 57, 71, 72, 78

MSC Mobile switching center 13

SQL Structured Query Language 33

SSTEM simulated STEM data 48, 71, 77

TAZ Traffic Analysis Zone 30, 31, 44, 45, 78

U.S. United States of America 44

WGS 84 World Geodetic System, 1984 18

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Estimating the traffic flows in a transportation network is important both for long-term planning of new infrastructure as well as in the short-term for traffic control. Today, usually mathematical traffic models are used that estimate the traffic flows using two datasources: travel surveys and link counts that are obtained manually or with the help of sensors on the road. Both of these datasources are expensive to obtain, thus limited to a very small group of participants respectively a small subset of the roads.

On the other hand did the use of mobile phones explode in the last decades. Device manufacturers are already using data collected from their users in order to estimate the traffic state on the roads in real-time. For traffic planners and researchers however, there is no way to get access to the same data collected by the Global Positioning System (GPS) sensors of phones. Notwithstanding this fact, an alternative is to use carrier-side cellular network data. For network operating reasons and billing purposes carriers already know approximately where their users are located, making the data relatively cheap to obtain. Even though cellular network data has usually a much worse time and spatial resolution compared to GPS data it could be used as an additional source of information about people's mobility. Not only is this data available in a much larger scale (in terms of the number of people in the sample) than other sources, but it also qualitatively reveals more information compared to for example link counts. While link counts only give information about the traffic flow on the link, mobile phone data could also be used to tell where people that are using a link are coming from and where they are going.

In fact, in the last decade already research has been conducted, especially focusing on estimating home-/and work-locations as well as travel demand from cellular data. However, to estimate *link flows* (the number of people/vehicles using a certain link), an exact route on the road network has to be estimated from the cellular network data. This is the main focus of this thesis.

1.1 Aim of the thesis

The aim of the thesis is to develop and implement a method that estimates flows in a transportation network using cellular network signalling data. Cellular network signalling data contains the information where a user was at specific points in time. However, the resolution both in space as well as in time is very low. In the case of Call Detail Records (CDRs) for example, only when a user actually made a call the information to which cell he/she was connected is available. The central research question treated in this thesis is:

> "How can cellular network data be used in order to estimate the link flows on the road network?"

To answer this question, the thesis aims to present a method that processes cellular data in order to build a time-sliced *OD-matrix* that provides an estimation of the travel demand at different times of the day. Finally the data will be used to estimate the routes that users took and assign the link flows to the transportation network in form of a data-driven network loading. A big advantage of cellular network data is that it is available for a much bigger sample than travel surveys or traffic counts. Therefore a secondary aim is, that the method is algorithmically efficient and can be run for a bigger city and data for 1 million users in several hours rather than months.

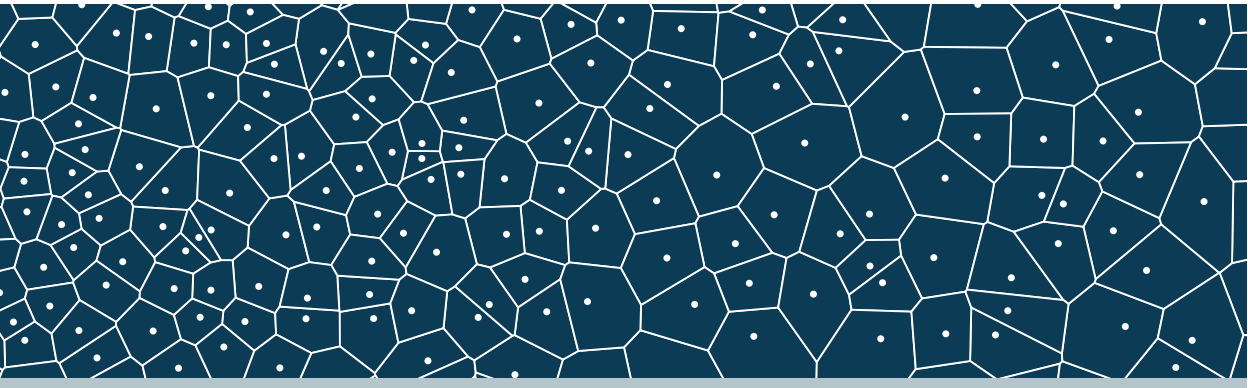
1.2 Structure of the thesis

The remainder of this thesis is organized as follows: Chapter 2 gives a basic introduction in all relevant fields that are necessary, to understand the designed method while chapter 3 will give an overview of existing studies in the field of cellular network data for traffic analysis. In chapter 4 the traffic flow estimation method is introduced, while the following chapters 5, 6 and 7 describe each of the components of the method in greater detail. Chapters 8 and 9 present the results for two datasets which the method has been tested with. Finally in chapter 10 and 11 the strengths and weaknesses of the method are discussed and an outlook on possible future work is given.

Chapter 2

Background

The method for estimating traffic flows from cellular network data presented in this thesis is built using existing knowledge about cellular networks, traffic modelling and geospatial algorithms. This chapter gives an introduction into all relevant research areas necessary to understand the model. This overview includes an explanation of the tools and algorithms used and will explain the most crucial terms that are used later on. While this chapter aims to focus on the parts necessary to understand the model, more detailed information about the relevant areas can be found in the literature referenced throughout this chapter.



2.1 Cellular networks

This section aims to only give an overview over how the cellular network data relevant for traffic analysis is collected. A more detailled description on how cellular networks work can be found in Walke (2002). With the rise of mobile communication in recent decades, cellular networks became ubiquitous all over the world. With the exception of some rural areas, even in poorer countries access to mobile telecommunication networks has become increasingly common. The range of an antenna to which a cellular device has communicate in order to connect to the network is limited by the laws of physics. Therefore to cover a whole city, carriers typically have to divide the area into cells each of which is covered by one cell tower, hence the name *cellular network*.

Whenever a user is called or makes a call, the network has to know in which of the cells the user is currently located in order to setup the connection. This information can be exploited to estimate the location of the user, as the cell tower locations are known (Rahnema, 1993). Cellular networks are designed hierarchically, where the individual cell towers form the lowest level. All cells in an area are connected with a Mobile switching center (MSC). The MSC itself then communicates with the central infrastructure of the carrier. This is where the information when and from which cell users made a call is stored for billing purposes. Even though operators might have access to more information about each position (like detailed signal quality measurements for multiple cells allowing a relatively accurate position fix) that could also be used, the minimum information necessary for mobility analysis is an identifier of the user, a timestamp and the cell id of the cell that the user was connected to at this time. An example of such a dataset is shown in table 2.1.

Even though there might be more types of data that could be obtained from cellular networks, this thesis only considers the following two datatypes:

- CDRs: For each event like a call or a text message that was sent, a record is saved in a database,
- *STEM* data (handover data for data connections): A record is saved whenever the user connects to a different cell, usually because the users physically moves out of the range of a cell.

As the illustration in figure 2.1 shows, is the time resolution of CDR

Table 2.1: An example of an (artificial) cellular network dataset

User ID	*Timestamp*	*Cell ID*
1	2013-10-01 06:50:00	1
1	2013-10-01 08:10:00	3
2	2013-10-01 08:20:00	2

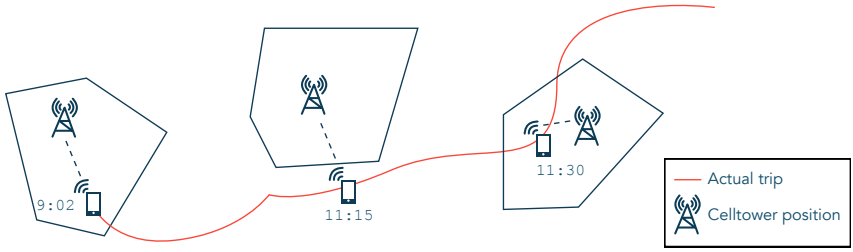


Figure 2.1: Illustration of CDR data collection. The connected celltower is only saved whenever the user makes a call (times given in the picture).

data very low and fully depends on how often the user makes calls. This also implies that part of trips or whole trips might be missed in the data (like the last part of the trip in the illustration). This also introduces an uncertainty about when a user started moving. For example, the user in the example made the last call in the first cell at 9:02, but probably started the trip later.

When a user is actively using the phone, so called handover events happen when the user leaves the range of the current cell in order to hold the connection (Tekinay et al., 1991). Modern smartphones maintain a continous data connection for example in order to receive incoming notifications. Therefore a big part of the day can be covered with handover data of these data connections that contains every cell the user connected to. Access to this data gives an even more precise picture of the movement of the user than CDRs. However, still not all users are using phones with a permanently active data connection and therefore the bias in terms of how different user groups contribute to the data might be different.

Inherent to cellular networks is also that each cell-tower has a limited bandwidth that is shared among the users of the cell. Therefore carriers have to place antennas more densely in urban areas than in rural areas in order to offer a good quality of service. This implies that the cell area that a tower covers are typically smaller in dense city areas giving a more precise information about the location of the user. The area that a tower covers (the cell shape) is not uniform in reality (Tettamanti et al., 2012). There are many factors that influence to which cell tower a user connects to apart from the distance, for example:

- the power of the antenna
- the landscape (heights that block the radio waves)
- buildings in the area that cause reflections and block the radio waves
- how much load a cell currently has (sometimes user are switched to another cell because it has less users on it)

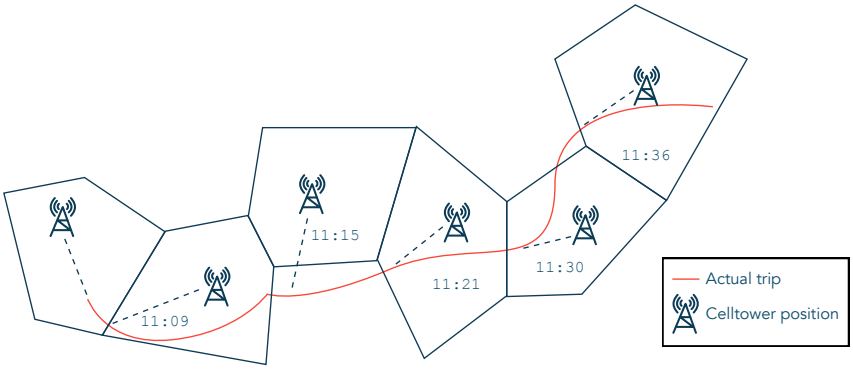


Figure 2.2: Illustration of handover data for data connections (continous connection). The connected celltower is saved at each handover event (whenever the user connects to another cell).

For this reason it is not easy to precisely describe the area covered by a cell. This is especially true when the only information available is the position of the cell towers. The easiest method to model the coverage is a *Voronoi diagram* (see chapter 2.3.4) which is also used in this thesis. However, it is important to keep in mind that such a Voronoi diagram is not a very accurate description of the cell coverage. Therefore both with CDR or handover data, the user might actually be outside of the estimated coverage area of the cell he/she was connected to in some cases (see the user position at 11:15 in figure 2.1 and 2.2).

As for traffic analysis, we are interested in the movement of the user through the cellular network. These movements can be described by a sequence of cells that the user traverses, which we will from now on call a *cellpath*.

Definition 2.1: *Cellpath*. A cellpath is a sequence of cells $p = (b_1, \ldots, b_n)$. To each trip t of a user belongs a cellpath $p(t)$ that contains all cells that the user connected to during the trip (upon CDR or STEM events) (including the start- and end-cell of the trip).

2.2 Traffic analysis

Traffic modelling and the estimation of traffic flows on a transportation network is important both for long-term planning purposes to find out how the transportation network could be improved and where problems exist but also in the short-term in order to detect overcrowded roads and to advice alternative routes to users. This section will give a brief overview of the terms and models used in traffic modelling that are relevant for this thesis.

2.2.1 Estimating link flows

Historically it has been a problem to estimate the flows on each link of a network precisely as there are basically only two common methods to gather that information: link counts and travel surveys. Link counts simply mean that the flow is measured on a link (the link flow) using either equipment like sensors or manual traffic counts. However, link counts are costly and therefore typically limited to a very small subset of the links in the network. Surveys are used to get information about how often and where people travel in order to estimate the demand between different areas. However, surveys are costly too and are therefore limited to a small number of participants from which the total travel demand has to be estimated using a travel demand model. The travel demand typically described by an OD-matrix which contains a number of travellers for each pair of origin-destination zones.

A model that is commonly used to estimate the flow on each link of a network based on statistical data is the so called *4-stage model* (McNally, 2008). This model consists of the 4 stages "Trip production" which is based on the attractiveness of each area estimated from statistical data, "Trip distribution" which creates the OD-matrix, "Modal split" which divides the travel demand among the available modes (for example car, public transport, bicycle, walking) and finally the "Trip assignment and network loading" which selects a route (e.g. which links are used in the network) for each trip and aggregates this information to link counts.

One of the major problems with traffic assignment methods is that the route choice depends on the costs (usually travel time) of each link as users tend to minimize their travel time. The travel-time of a link depends on the the length of the link and it's *free flow speed*. However, in reality travel time increases as soon as there are a certain number of vehicles on the same road. Therefore the travel time on a link is usally described by a so called *volume-delay function* that takes both the free-flow travel time and the flow on the link into account.

In traffic modelling a so called *user equilibrium* is used to account for this interdependency between travel time and flows (Patriksson, 2015). A network loading is in user equilibrium state when all alternative routes between two locations have the same cost (travel time). This also implies that also users are using the route that is optimal for them, as there is no other alternative route with a lower cost. Simplified, a user equilibrium can be calculated by shifting flows belong alternative routes until all routes have the same cost. Note that the user equilibrium does not coincide with the *system optimum* which minimizes the combined travel time for all users instead of the cost for each single user. Although user's tend to minimize their travel time, in reality there is a certain variation. This is because people might not have perfect knowledge about the current travel times on all links or have other reasons to choose a different route. Therefore sometimes stochastically models are used to model the route choice.

2.2.2 Iterative proportional fitting

Given a $n \times m$ matrix P, Iterative Proportional Fitting Procedure (IPFP) is a method to rescale the matrix given the margin totals (column respective row sums)

$$r_i = \sum_{j=1}^{m} p_{ij} \tag{2.1}$$
$$c_j = \sum_{i=1}^{n} p_{ij} \tag{2.2}$$

are known. The method was first described by Deming et al. (1940). Later it was discovered that it can be used in order to rescale a matrix. The produced works by iteratively first adjusting the rows to the margin totals and then the columns. The algorithm stops when the change falls below a threshold (conversion criteria). The algorithm conserves the ratios between the values in the matrix as good as possible and has been proven to converge (Fienberg, 1970).

Algorithm 1: Iterative proportional fitting

Input: matrix $n \times m$ matrix P
Output: scaled matrix $\hat{P}$

while *not converged* **do**
  for *k,l* **do**
    $P' := P$
    `//Adjust rows`
    $p_{k,l} := \frac{p'_{kl}}{\sum_{j=1}^{m} p_{ij}} r_k$
    `//Adjust columns`
    $P' := P$
    $p_{k,l} := \frac{p'_{kl}}{\sum_{i=1}^{n} p_{ij}} c_l$
return $\hat{P} := P$

In traffic modelling, IPFP can be used in order scale an OD-matrix. For example to estimate how the travel demand would change when the population increases in a city or if an OD-matrix was calculated for a small sample if users, to scale it to the whole population.

2.2.3 The GEH statistic to compare linkflows

To validate a network loading result we need to compare the estimated link flows in a network with ground truth link flows. A naive metric like calculating the average difference over all links implies not much about the quality about the network loading. An average difference of 0 could either mean that the network loading matches perfectly, but also that many links

have to high flows and many other links to low flows at the same time. Using the absolute value of the relative difference for each link would be a more useful metric. However, this would value each link equally, which is not very helpful either. For example might a difference of 50% on a link that is only used by 10 people be acceptable, while this would not be acceptable for a freeway used by several thousand people in an hour. Therefore often the so called *GEH* statistic (named after it's inventor Geoffrey E. Havers) is used in traffic engineering to compare link flows (Balakrishna et al., 2015).

The GEH value for a link with an estimated flow of q_e and a ground truth flow of q_t is defined as follows:

$$GEH(q_e, q_t) = \sqrt{\frac{2(q_e - q_t)^2}{q_e + q_t}} \tag{2.3}$$

According to common practice, values of less than 5 are considered a good match. Values bigger than 10 are considered unacceptable. For a whole network loading 85% of the links should achieve a value less than 5.

2.3 Geospatial algorithms and geographical information systems

The third area relevant for the model besides cellular networks and traffic modelling are geospatial algorithms. To process data that is related to a position on earth, like antenna positions and the road network, a number of existing algorithms was used for the implementation of the model. This section will describe the tools and algorithms used in this thesis.

2.3.1 Geographical information systems

Some data used in the traffic flow estimation method, like the road network as well as the cell tower positions, is related to geospatial positions. Handling this type of data raises questions about how to encode the positions in order to save the data and how to efficiently query the data necessary for the current calculation. Fortunately geospatial data is relavant in many areas and therefore systems that solve most of the general problems with this type of data has already been developed. These Geographic Information Systems (GISs) can store geospatial data in a database allowing to query the data efficiently and also provide common geospatial operations (for example check if two features are intersecting, compute the union of polygons etc.). An important part of a GIS is also a tool that allows to create visualizations from the data, which is both important for presenting the results of the computation as well as to stepwise debug and verify that an algorithm works as expected (Fotheringham et al., 2013).

One of the more complicated parts in handling geospatial data is to map positions on the ellipsoidal-shaped earth onto a two-dimensional coordinate

system. GIS systems support a variety of coordinate reference systems and it depends on the use and the area of study which is most appropriate. For the implementation of the traffic flow estimation method presented in this thesis the widespread World Geodetic System, 1984 (WGS 84) is used. The reason is that this system works worldwide and therefore allows to use the same code with data from different parts of the world. For accurate distance calculations in metric units however, in some place the coordinates are transformed into a regional coordinate system, as WGS 84 only allows to calculate distances in spherical degrees.

2.3.2 Finding shortest paths

Finding shortest paths in a graph is one of the classical problems in computer science. It is relevant in particular for the route estimation of travellers (see chapter 7). Even though travellers are not always taking the shortest path, they tend in general to prefer a shorter path over a longer one.

Given an edge-weighted graph $G = (V, E)$, where V is the list of vertices and E the list of edges, the shortest-path problem is to find a set of connected edges between two vertices $s \in V, t \in V$, such that the combined weight (cost) of the edges is minimal.

The most famous algorithm to solve this problem has been proposed as early as 1956 by Edsger W. Dijkstra (Cormen, 2009). It works basically by exploring the whole network from the start node and calculating the shortest paths to all other vertices in the graph and has a complexity of $\mathcal{O}(|V|^2)$. As the algorithm is very simple it is widely used. However, as the computation time grows quadratic with the number of vertices, the algorithm performs very bad when used in large-scale networks like road networks on regional or even continental level.

As optimal pathfinding is a common problem relevant for many use-cases, attempts have been made to reduce the calculation time with new algorithms. One of these more sophisticated algorithms is the A* algorithm, that has the same worst case performance as Dijkstra, but reduces the average calculation time in practical uses with road networks (Goldberg et al., 2005) by exploiting the spatial locations of the vertices in order to evaluate the vertices in a more efficient order.

Luxen et al. (2011) presented an algorithm that uses contraction hierarchies in order to speed-up the routing. This algorithm works by converting the original graph into a graph optimize for routing. This contracted graph is created by defining a metric of importance of the nodes and recursively removing "unimportant" nodes and replacing them with shortcut edges. The contracted graph is a directed acyclic graph which allows a efficient route calculation using a bidirectional Dijkstra with a very small number of vertices to be explored. An implementation is available under the name Open Source Routing Maching (*OSRM*[1]) and is available as *Open Source*.

[1] see http://project-osrm.org

2.3.3 Line simplification

Line simplification is another algorithmic problem, that certain variants of the network loading procedure presented in chapter 7 makes use of. The problem is for a given line (described by a list of its point coordinates) to remove points from the line while maintaining the shape of the line as good as possible. This type of problem is well-known in computer graphics, as it can be used in order to compress vector data saving both memory and computation time when using this data.

The most popular algorithm that solves this problem was suggested independently by Ramer (1972) and Douglas et al. (1973). The algorithm recursively divides the line into two segments at the point that has the biggest distance from the segment. If the distance of a point is below a given threshold (a parameter of the algorithm), it is being removed from the line. This procedure keeps the most extreme points of the line while removing those that don't change the shape of the line much. The algorithm is widely used as it is very simple to implement and has a complexity of just $\mathcal{O}(|V| \log |V|)$ in average case, while the worst case complexity is $\mathcal{O}(|V|^2)$ though (V is the set of vertices of the line). Figure 2.3 shows an example of how the algorithm works.

2.3.4 Voronoi diagram

A Voronoi diagram divides a plane into areas from a given set of points (sites). Each area or polygon in the diagram marks the area of the plane for which the point from the given point set that is inside the polygon is the closest out of all points from the point set. For example if the point set is consisting of bus stop locations, a Voronoi diagram would show for which area a given bus stop is the closest. Voronoi diagrams have many applications like finding the closest point of interest for a given location or modelling the coverage of each object in the point set. They are commonly used in geospatial analysis and computer graphics. A Voronoi diagram is dual to the Delaunay triangulation, which is often used in computer graphics. Computing the Voronoi diagram can be done in linear in the size of the set of points given (Aggarwal et al., 1989).

In this thesis a Voronoi diagram is used in order to model the coverage of the antennas. Using the antenna locations as the input point set, we get a partition of the plane into polygons, also called *Voronoi cells* (see figure 2.4). We interpret the Voronoi cell of an antenna as the area that is covered by that cell, even though antenna coverage is much more complicated and depends on many more factors like landscape, buildings, power of the antenna etc. than only the euclidean distance. However, as we have no more information about the antennas except their position, the Voronoi diagram is an easy and acceptable representation of the antenna coverage (Baert et al., 2004). It is also widely used in the literature about cellular data for traffic analysis (Gundlegård et al., 2015; Fillekes, 2014; Wu et al., 2015).

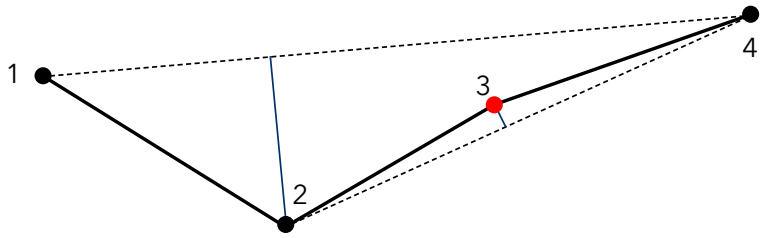


Figure 2.3: Example of a line simplification using the Ramer-Douglas-Peucker algorithm. First the point with the maximum distance from the straight line between point 1 and 4 is searched, which is 2. Assuming the distance is above the threshold distance, the line is split at point 4 and same is recursively repeated for the segments 1–2 and 2–4. For the segment 1–2 recursion stops, as there are no more points in the segment. In segment 2–4 the point with the maximum distance is point 3. Assuming that this distance is below the threshold, point 3 is removed.

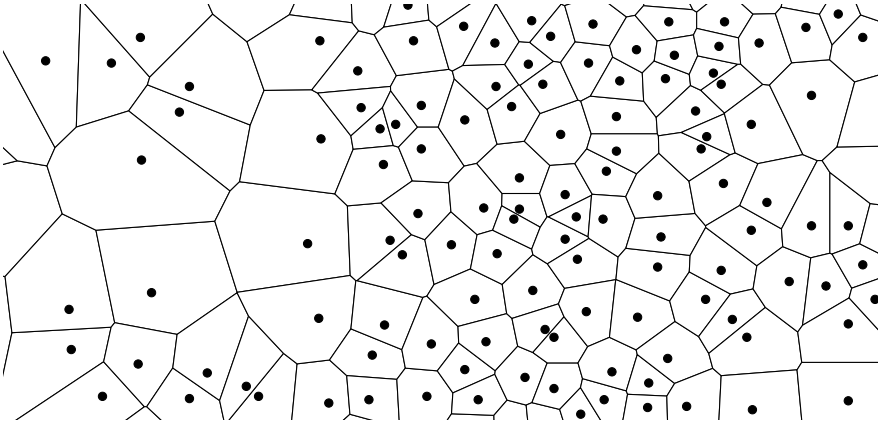


Figure 2.4: An excerpt of the Voronoi diagram based on the antenna positions for Dakar, Senegal

2.3.5 Clustering point sets

Some calculation steps in the traffic flow estimation method grow quadratic with the number of cells (antennas) used as input. A natural way to reduce the computation time is therefore to reduce the number of antennas used. Similar to the line simplification problem, the goal is to reduce the number of points (antennas) without loosing too much information, as any change to the antenna positions can make the result more inaccurate.

This problem can be solved using point set clustering. The goal of clustering algorithms is the grouping of nearby objects, allowing them to be merged into one single object. A problem that is also relevant for pattern recognition and classification tasks. There exist many different clustering algorithms. Depending on the use-case different critera to join points and metrics to measure similarity are appropriate (Estivill-Castro, 2002).

To the more popular clustering methods belong centroid-based clustering like the k-means clustering algorithm and hierarchical clustering (Murtagh et al., 2012). Most centroid-based methods require the number of clusters to be specified in advance, while hierarchical algorithms usually use a linkage criteria to join points subsequently to a cluster. Besides the linkage criteria one has also to chose a distance metric that is used to evaluate the linkage criteria.

To cluster a point set of cellular antennas it is more natural to define a linkage criteria, stating that antennas with a distance below a certain threshold should be grouped together than to require the number of resulting clustered antennas to be specified in advance. Therefore an hierarchical clustering algorithm using a single linkage-criteria is used. Single-linkage means, that the distance between two clusters is defined as the minimum distance between any pair of points in two clusters:

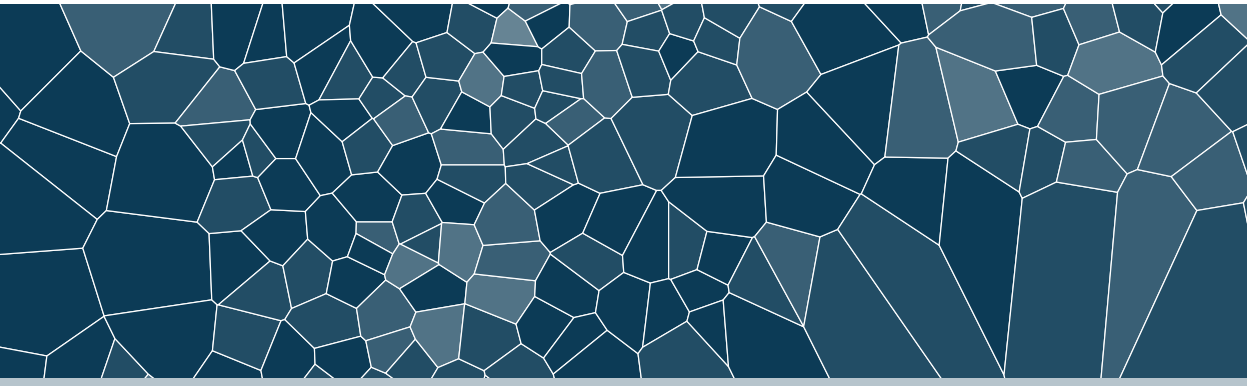
$$\min \{\, d(a,b) : a \in A,\, b \in B \,\} \tag{2.4}$$

Here A and B are two clusters defined by the set of objects that they contain.

Chapter 3

Literature study

Despite the fact that cellular network data for traffic analysis is a relatively new approach, already a reasonable amount of research has been done that shows its potentials. Especially many papers are raising awareness for the advantages of using cellular network data over classical data sources like surveys or traffic counts. Many papers focus on the estimation of the traffic demand (OD-matrices), while only limited research has been done regarding route choice and assigning the traffic demand to the actual links traffic/road network.



3.1 Cellular network data for traffic analysis

Using cellular network data for traffic analysis has a number advantages compared to data collected in travel surveys or using traffic counts (manual or using detectors). Caceres et al. (2007) as well as Steenbruggen et al. (2013) name for example the much bigger sample size due to the possibility to collect data for every cellphone user instead of just a small group like in surveys. Already Rose (2006) supports this with an intercept survey of drivers made at traffic signals in Melbourne, Australia. The findings showed that 61% of the total travel in Melbourne (based on travelled distance) is made with a mobile phone that is switched on and could therefore be captured in cellular network data.

In contrast to using, for example, traffic counts, cellular network data covers in principle all travel modes and not only car traffic (Gundlegård et al., 2015). However, it should be noted, that especially walking trips are often shorter than the cell size making those trips very difficult to capture (especially in the case of CDR data). Other advantages described by Caceres et al. (2007) are the much shorter update interval (possibly real-time compared to several years when using surveys) and the better spatial coverage compared to detectors (that only can cover a selected number of links). In order to be able to adopt existing methods for estimating the travel demand in a network, Caceres et al. (2007) proposes to use virtual link counts "located on the borders between the LAs [Location areas] of GSM network".

Even though the large sample size of users when analysing cellular network data, Rose (2006) points out that the sample might not be representative for the whole population because it's possibly biased depending on the customer base of the carrier that the data is from. Also certain groups might be over/underrepresented because of a deviating mobile phone usage.

Another problem with using CDRs for traffic analysis is discussed by Gundlegård et al. (2015): the poor spatial resolution and accuracy. The spatial accuracy is especially diminished by the fact that an area can be served by multiple antennas (overlapping cells). Also the coverage can be different in different directions from the antenna due to wave propagation depending on how the area formed (obstacles, heights etc.). Therefore Gundlegård et al. (2015) conclude that a Voronoi diagram (see chapter 2.3.4) can be a poor representation of the cell area. However, if no data in addition to the cell positions is available, a Voronoi diagram is an easy and widespread way to estimate the real cell coverage. Another bias that Gundlegård et al. (2015) mentions is the time bias which is specific to using call detail record (CDR) data. As the number of available positions directly corresponds to the number of calls and messages made respectively sent at a given time of the day, more trips might be extracted from the data when a user makes more calls on a day (not necessary meaning that more trips actually have been made) leading to the users contribution to the

travel demand being overestimated. Analysing the time distribution of captured events (calls and messages) on a cellular network dataset from Senegal Gundlegård et al. (2015) noticed that users are more likely to call in the evening compared to the morning.

Steenbruggen et al. (2013) provide a comprehensive meta-study of traffic analysis based on data from GSM networks. It compares 17 different studies that where conducted between 1994 and 2009 in the field made all over the world, mostly with a focus on traffic management and short-term prediction of traffic. The meta-study concludes that most studies made focus on the traffic parameters travel speed or travel time. Other traffic measures like the flows are considered less often. Also most of the studies focused only on limited road stretches rather that a whole network of roads.

3.2 Travel demand estimation from cellular network data

Even though cellular network data is usually available for a relatively big sample of users, it doesn't contain everyone. To estimate the total travel demand therefore it is not enough to add up all trips that can be extracted from the data. Instead some kind of model to scale the available data to the whole population is necessary. Such models for travel demand estimation from cellular network data have been developed for example by Ming-Heng et al. (2013), White et al. (2002), and Calabrese et al. (2011).

Ming-Heng et al. (2013) did a case study using CDR data to estimate the travel demands in the Kansas Metro Corridor. The model processes the data in several steps. First the data is filtered, such that it doesn't contain any irrational positions (that for example would required unrealistically high travel speed from the previous position). Then trips are detected using an activity based model, where each user either is in movement or not at any given time. Using a time window it is determined when a trip is assumed to have ended and reached the destination. The trip data is finally aggregated to a dynamic OD-matrix. Comparing with official traffic counts, Ming-Heng et al. (2013) discovered that the estimated travel demand from the CDR data was about 17.6% of the daily traffic demand according to the official counts.

In Calabrese et al. (2011) a similar method for estimating travel demand using cellular network data is presented: First trips are extracted from the data, which are in a second step aggregated to a dynamic OD-matrix. The trip extraction uses certain limits like a minimum time and a maximal distance to determine when a new trip starts. Also antennas close to each other are fused together to cope with calls made from different cells because of network balancing rather than real movements. To estimate the travel demand for the whole population (comparable to official travel demand data) from the available sample the method uses different scaling factors when

aggregating the trips to an OD-matrix.

One important advantage of using cellular network data to estimate travel demand is pointed out by White et al. (2002): While measurements at specific links (with detectors) yield only the sum of all route flows that use that link, cellular network data can give information about the origin and destination of individual users. For a set of link flows often many different feasible OD matrices can be found. With the information on the trajectories of users, more information is available to estimate the actual OD flows. On the other hand, White et al. (2002) also point out that the availability of information about individual user is problematic for privacy reasons and can have legal implications in many countries. White et al. (2002) identify this as one of the key issues with using cellular network data for traffic analysis.

3.3 Route choice and network loading

The route choice estimation problem is to find the most likely route on the road network given a list of traversed cells. It is important in order to find out which links flows should be assigned to during a network loading. There is only a limited number of publications on recovering routes from cellular network data so far. However, the existing studies use very different methods in order to solve this problem.

Hoteit et al. (2014) propose an interpolation method in order to connect the sparse data points of CDR data. This trajectory reconstruction is independent from a road network and is therefore not directly usable for a network loading. Some methods like Becker et al. (2011) and Wu et al. (2015) use an approach where the alternative routes for each origin-destination pair are pre-calculated. Using classification they select the route that matches best with a given cellpath. The route alternatives can for example be determined using a traffic assignment, simulation or by using a general routing algorithm on the route network that can return multiple alternative routes.

Other authors like Gundlegård et al. (2009) use methods that are similar to what is often used for the map-matching problem of GPS traces like nearest-neighbour classification or Bayesian classification. However, this study assumes that information on the signal strength of several antennas nearby is available which allows a much better positioning than if only the connected cell is available and only whenever the user makes a call.

Fillekes (2014) compares different map-matching methods in order to reconstruct trajectories on a road network from CDR data. These methods include for example the often used and very simple approach to map a CDR to the closest point on the map to the antenna position. It also includes more sophisticated methods like using the center of gravity of the Voronoi cell instead of the antenna position and to prefer links with higher importance when matching a CDR to the map. This importance can for example be measured by the degree of centrality of a link or by other attributes like

the road category, speed limit or the length of the link. The work also contains a comparison between different similarity measures to compare the reconstructed route to an original ground truth route (that was supplied by GPS in the study). Using these similarity measures, she concludes however, that "these rather low values indicate that the methods developed in this study do not work well on a general level" (Fillekes, 2014).

A promising work on the problem has been done by Leontiadis et al. (2014), who use the idea to transfer the cells that a user visited during a trip into probabilities that particular links where use on the journey. Before calculating a route, they modify the cost of the links in the network making it cheaper to use links that are inside the coverage of the cells that the user visited. Their *Cell** algorithm applied to CDR data (however, they have a model of the sector coverage of the antennas instead of just using the Voronoi diagram) achieved a median error of 230m when comparing to ground truth GPS trajectories. The idea of this algorithm is the core of the Lazy Voronoi Routing algorithm presented in this thesis (see chapter 7.4).

3.4 Privacy issues

From the raw cellular network data it is possible to track single user's positions which poses a number of privacy issues. In the case of CDR data the position is only known when an event occurs (the user makes a call or sends a text message). However, even this timely very sparse data can reveal the most important places of a person (for example home- and work locations) when it is aggregated over a longer time period. More detailed data than CDRs like handover data could even give a more or less complete movement profile of a person.

Zang et al. (2011) analysed 30 billion CDRs from a dataset for the USA covering a three month period including 25 million users and came to the conclusion that it is not possible to publish the raw data without compromising privacy. They recommend to either publish data with poor spatial or timely accuracy as a trade-off between privacy and the utility of the data for research purposes. One way that they mention in order to maintain users privacy is also the concept of differential privacy. It works by "adding the appropriate amount of noise to the result of queries, the benefit being that a user's presence or absence from the database cannot be determined by queries" (Zang et al., 2011).

As Andrienko et al. (2013) point out, carriers should make their customers aware of which data is used for which purpose. They could also offer users to opt-out of any use of their data that is not necessary to provide the network service. Becker et al. (2011) point out that the position is never very accurate as cell towers often cover an area of many square kilometers. They use the following measures to not disclose the privacy of single users:

- a pseudonymous identifier is used to identify users instead of a perma-

nent and known identifier like the phone number,

- no other demographic data about the user is linked to the pseudonymous identifier and
- results are only presented in aggregated form and no single traces of users are revealed

For the one dataset that is based on real data, these measures are also applied in this thesis. The dataset is also already preprocessed and obscured in several ways as described by Montjoye et al. (2014) and in chapter 8.1. The other dataset used in this thesis is based on simulated data and has therefore no privacy implications.

Chapter 4

Overview of the traffic flow estimation method

To goal of the traffic flow estimation method presented in this thesis is to estimate the traffic flows on the transportation network, given the data from a cellular network. In order to do that we design a method that processes the data step-by-step. The complete method presented in this thesis consists of 4 main components: Trip extraction (see chapter 5), travel demand estimation (see chapter 6), route estimation and network loading (see chapter 7). This chapter focuses on the interaction between these components and how the method relates to the classic 4-stage model assignment model used in traffic analysis, while the details of each component will be covered in their own chapters.

4.1 The classic 4-step traffic assignment model

The model of the traffic assignment procedure using cellular data presented in this thesis is loosely based on the classic 4-stage model for traffic assignment that is often used in traffic modelling (see figure 4.1). The model consists of the four steps "Trip generation", "Trip distribution", "Modal split", "Route choice and network loading" (McNally, 2008). In the classic model these steps work as follows:

1. **Trip generation.** As an input there is a certain number of origin locations and destination locations. The first step is the generation of trips. In this step, for each origin the estimated number of outgoing travellers O_i and for each destination the estimated number of incoming travellers D_i is computed. These numbers are the input for the second step.

   Each origin and destination represents normally an area with more than only one building (so called Traffic Analysis Zone (TAZ)). The values typically depend on the number of people living in the area or the number of working places, the touristic attractiveness of the area and so on.

2. **Trip distribution.** This step uses the O_i and D_i numbers to estimate how many passengers will go from each specific origin to each specific destination. As the number of people going from one location to another does not only depend on the attractiveness given by the O_i/D_i values, but also on the infrastructure connecting these locations, the cost (in terms of distance, travel time, monetary cost) for travelling between the locations has to be taken into account.

   The result of this step is a so called OD-Matrix that contains a value T_{ij} for each origin/destination-pair denoting the number of people travelling from origin i to destination j. To compute the matrix typically a gravity model is used (Evans, 1973).

3. **Mode choice.** This step takes all T_{ij} values as an input and separates the amount of travellers over the different travel modes (for instance car, public transport, pedestrian/cycle), depending on the costs for the different modes. Because the choice of modes is not only a rational decision minimizing the cost in reality, a stochastic element is

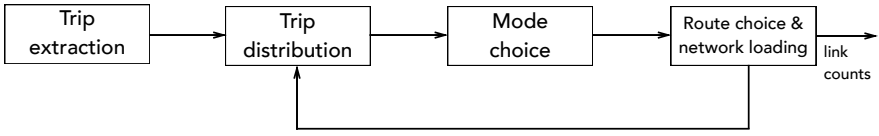


Figure 4.1: The classic 4-stage model for traffic assignment

often added in this step. One popular model for computing the mode distribution is the Logit model (Wen et al., 2001). The output of this step is a number of passengers T_{ij}^m travelling from i to j using mode m for each origin, destination and mode combination.

4. **Route choice and network loading.** In this final step, the flow on the road or public transport network is being computed. That is, travellers T_{ij}^m between i and j using mode m will not necessarily all take the same route over the road or public transport network. In order to do that also information about the network must be given as input (links, nodes, cost for each link).

   Using this input data, all possible routes can be computed and the travellers are then split according to the costs each route has. Finally the number of travellers is added to every link on the route they take, in order to compute the number of people taking each link (link flow).

In reality there is a problem, that prevents the 4-stage model from just being executed consecutively for once to get a final result. This is because step 2 already needs to know the cost that it takes between two locations in the network. This cost contains also the travel time, which again depends on how many vehicles use the same roads (that is the flow). Therefore the output of step 4 is an input of step 2. The practical solution to solve this problem is to execute step 1 first and then execute steps 2-4 not only once, but multiple times. After some iterations is the result usually stable and provides a good approximation.

4.2 The cellular data traffic flow estimation method

While the classic 4-stage model uses information about the road network and some form of census data that contains information about land use to model the attractiveness of the TAZ, the traffic flow estimation method based on cellular data presented in this thesis is data-driven assignment. It doesn't necessarily need census data, but instead it relies on large-scale data from a cellular network in order to trace peoples movements.

The whole method can be split into 4 main components that more or less correspond to the classic 4-stage model (see figure 4.2): "Trip extraction", "OD-matrix estimation", "Cellpath routing" and "network loading". In contrast to the classic model, a separate mode choice estimation is not part of the method yet. Estimating the mode choice from very sparse data like CDRs is very challenging and could easily be a project on its own. To use such a method in real traffic planning application, however, for most applications, some kind of mode choice model would need be integrated in the method. As of now the model estimates the total travel demand and assigns it to the transportation network.

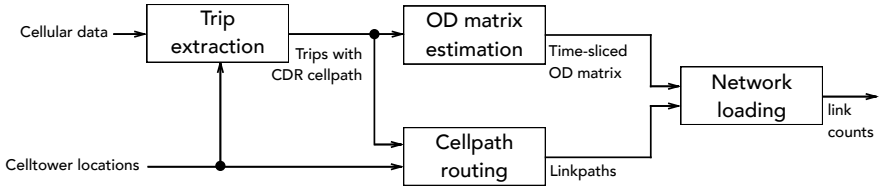


Figure 4.2: The cellular data traffic flow estimation method

Another major simplification of the presented method compared to the classic 4-stage model is that there is no feedback between the steps, meaning that the components are executed linearly just once instead of an iterative process as in the classic model. The purpose of the feedback loop in the classic model is to shift people from a congested road to another road if that gives a shorter travel time. The reason we can skip this feedback loop, but still to some extend model route changes due to congestion is that we have the cellpaths that users took. If one of the routes is congested and the user chooses a route different from the shortest path assuming no traffic, this will be manifest in a changed cellpath at least in the case that the routes don't share the exact same cellpath.

The four components of the method work as follows:

1. **Trip extraction.** This component roughly corresponds to the trip generation in the classic 4 stage model. It reads the raw cellular data (CDR or STEM) and tries to distinguish between movement and stand-still, as we are only interested in movements. In contrast to the classic model, the connection between origin and destination is already given from the data as every trip has a start and end-cell. The result is a trip table containing additionally to the start- and end-cell of the trip all cells in-between (the cellpath) and the information when the trip started and ended.

2. **OD-matrix estimation.** This component corresponds to stage 2 in the 4-stage model. When extracting trips from cellular data, we will always just get a sample of the whole population, as not all people have the same carrier and are not always carrying a switched-on mobile phone either. In order to estimate the travel demand for the whole population in an OD-matrix, this component scales the available trips to give representation of the total travel demand. It also splits trips among the different time-slices of the OD-matrix. The result is a time-slice OD-matrix, meaning that there is one sperate OD-matrix for each interval (for example each hour).

3. **Cellpath routing.** In the end we want to calculate the flows on the links of the transportation network. Therefore we need to convert each cellpath into a route on the network (a *linkpath*). As cells can

be cover a large area and contain many roads that possibly could be part of the linkpath, this is a challenging task. As part of this thesis a couple of different cellpath routing algorithms will be presented. This component is a part of the fourth stage in the classic model. The result is a linkpath for every cellpath.

4. **Network loading.** In this final component, which in the classic model is also included in the fourth stage, the travellers are assigned to the road network. This works by going through the OD-matrix that was calculated in step 2. For each OD pair the flow is distributed among the different cellpaths from the trip table. For each cellpath then the corresponding flow is added to each link that is part of the linkpath calculated in step 3. The flows resulting from all OD pairs are aggregated and the result is a link flow on every link on the transportation network as it is in the classic 4-stage model.

Even though this method has been completely implemented in order to be able to run a complete network loading, the biggest focus was on the cellpath routing component. One of the reasons for that is, that no ground truth data was available to validate the trip extraction and OD-matrix estimation components.

Note that the model doesn't contain any component that corresponds to the mode choice stage in the classic 4-stage model. This means that the total travel demand in terms of number of people is assigned to the transportation network. In order to estimate for example the vehicles on the road, a mode choice component of some form should be added to the model. While the method presented here assigns all traffic using routes optimized for cars onto the road network, a mode sensitive network loading should apply different routing algorithms optimized for each mode.

4.3 Implementation

The method was implemented in the *Python* programming language and the Structured Query Language (SQL) inside a *Postgres* database. The main reasons for this choice of tools were:

- Python code is relatively easy to read and maintain and makes it easy to write tests verifying the code,
- There are good mathematical libraries like *NumPy* and *SciPy* available for Python,
- Postgres serves as a data-storage for the large-scale data and provides efficient indexing to access the data fast,
- the *Postgis* extension for Postgres allows to perform geospatial operations directly in the database and

- using a Postgis database makes it easy to use GIS tools to visualize the results and debug the code.

The traffic flow estimation procedure can be run from a single Python script that runs the components of the method. The traffic flow estimation is automated and the scripts creates the necessary tables in the database and load them with data. In the end the network loading result can be fetched from the result table or directly viewed in a GIS application like *QGIS*. As all steps are Python scripts on their own that are just called by the main script, all steps can also be executed individually and there is no need to start over the whole procedure from the beginning each time. For example, if an error occurred the procedure can be started at the same step again. The code is also prepared to support different datasets. The code for parsing the input files (for example containing the antenna positions or the cellular data), is separated from the rest of the code and can be changed as necessary for a new dataset. It is also possible to skip the loading of data into the database completely and manually load the data into the database.

The biggest challenge during the implementation of the method was to cope with the big amount of data. While the naive implementation of the method would have taken weeks to execute the code was optimized for performance in many different ways bringing down the computation time for a dataset for Los Angeles, USA (L.A.) to a couple of hours. The following strategies were used to optimize the code:

- Parallelization of all computation intense steps,
- make the code memory-efficient by only loading small batches of data into the memory at a time,
- consequently use indexing in Postgres where appropriate and
- the usage of a custom Python/*C++* wrapper for OSRM for very fast route computations.

To easily adopt parallelization of almost all steps, a small framework was implemented which is used throughout the code. This framework is based on the standard Python multiprocessing library which allows to distribute a computation to multiple processes that communicate the results to the main process. Python already has a built-in `map`-function which can be used to apply the same function to every element in an list. However, this map-function executes linearly and blocks the process until the complete list has been handled. The framework implemented for this thesis features two classes:

- `ParMap`: Executes a function on every element of an array like the normal `map` function, but splits the work into batches that are dispatched to a number of processes (as many as there are CPU cores available) and collects the result in a list and

- `MapReduce`: Simultaneously applies a function to each element that returns a list of key, value pairs and aggregates the key, value pairs using a reduce-function and collects the result in a dictionary.

The concept of MapReduce is commonly used in cluster computing in order to process or aggregate large amounts of data (Dean et al., 2008). To illustrate the concept, let us assume we have a list of integers and we want to calculate how often every unique number appears in the list. Using the concept of MapReduce, we would define a map- and a reduce-function as follows:

```
def mapf(x):
    return [(x,1)]
def redf(x):
    k,v = x
    return (k,sum(v))
```

The map function returns a single key,value-pair where the key is the input element and the value always one (representing one occurrence of the number in this case). The reduce function gets a tuple of the form `(key, [value1, value2,...])` and aggregates the values by calculating their sum and returning a new `(key,value)` pair where the value is now the sum of the aggregated values for the key.

Let us assume the variable `L` is a list containing 50000 times the value 1 and 1000 times the value 2. Then we can create a new `MapReduce` object using the two functions we just defined and apply it to our list `L`:

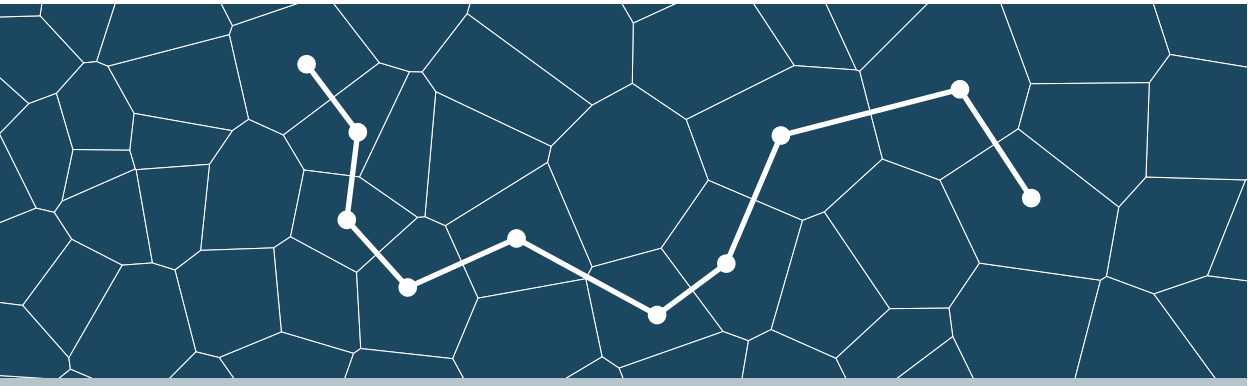
```
mapreducer = util.MapReduce(mapf,redf)
r = mapreducer(L)
```

Now `r` will be a list containing two key-value pairs: (1,50000) and (2,1000) giving us the counts for each number. The advantage of using this framework is that computation is automatically done in parallel processes using all available CPU power. The reduce function is called periodically when the map function has supplied a sufficient amount of new key, value pairs. This allows to constantly aggregate the data, which means in the example that only the sum of the values so far has to be kept in memory not all of the key, value pairs. MapReduce is for example used in the network loading step of the traffic flow estimation procedure, where the keys are the link-ids and the values the flows. Despite the big amount of data this allowed to keep memory usage for the network loading at a level of just of just a few megabytes.

Chapter 5

Trip extraction

Raw cellular network data only contains the cell a user was connected to at a given time from which the position can be estimated. However a lot of these records may be at the same position. For traffic analysis however we are only interested in the movements that users make between different locations. This chapter will deal with the question how to extract trips from the data. Essentially, this is about the question how to define the start and end of a trip. Imposed by the insufficient data quality of especially CDR data, this is a non-trivial question.



5.1 Extracting trips from CDR data

When extracting trips from CDR data, we are facing difficulties due to the incomplete information (the position is only known at times where the user initiates a call or sends a text message). As this is less of a problem with STEM data, the approaches for trip extraction presented in this thesis are substantially different for CDR and STEM data.

Due to the sparsity of CDR data, movements might be non-reconstructable (in case the user made no call during the movements). But even if the user made at least two calls in different locations, it is unclear where and when he or she started and ended the trip, as the calls might have happened during the trip. Other issues are that a change of the cell that the user is connected to does not always imply that the user actually moved. Connecting to a different cell can also happen when the network notices that a certain cell has a high load and it is better that some users connect to a cell that has more free capacity.

The method used to extract trips from CDR data for the experiments is the one that has been presented by Gundlegård et al. (2015). It consists of two steps:

1. Estimate a home and work cell for every user
2. Extract trips for every user from the CDR data

The basic idea of this method is to extract as many trips as possible from the data. It does this by "snapping" trips to two important locations (home and work) that are defined for each user. This allows to extract trips even if a user just made a single call.

5.1.1 Home- and work cell estimation

Given a dataset for a sufficiently long time (at least one or two weeks), it is relatively easy to estimate in which area a user lives and works. This can be done just by querying which cell is most often occurring during daytime (presumably the work location) and nighttime (presumably the home location). The algorithm of Gundlegård et al. (2015), also requires a minium distance between the two locations. The reason is that the cellphone might connect to different nearby cells for the same location at different times due to load balancing.

The method could even be extended to a more generic variant, where CDR events of a user spatially clustered in order to find all important places for the person (this might be more than just home and work). Such a method has been proposed by Isaacman et al. (2011). Validation with a group of volunteers showed a median error of 0.9 miles respectively 0.83 miles for the home- and work-locations estimated from CDR data, which is impressive given resolution of the data.

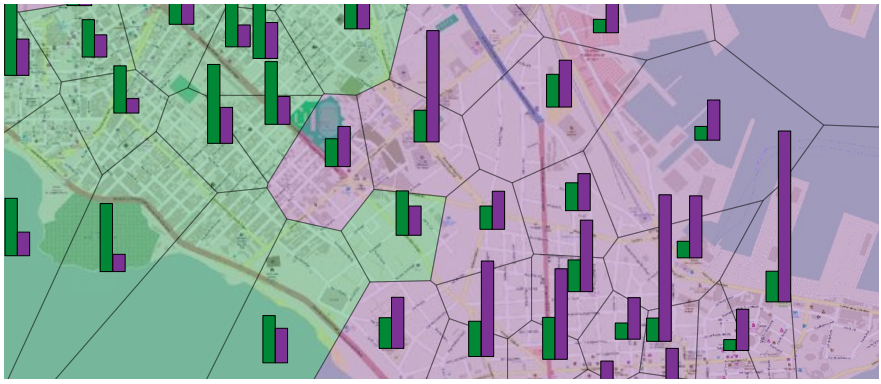


Figure 5.1: Home/workplace estimation for a part of Dakar, Senegal. Green areas indicate more homebases than workbases and purple areas more workbases than homebases.

Once home- and work-cells have been estimated for every user, it is easy to aggregate the data in order to find out which areas are more residential and which are more workplace areas depending on the relation between the number of home- and workplaces in estimated in every cell. Figure 5.1 shows such a visualization for a part of Dakar. Comparing with the map data, that the city center in the southeast as well as the harbour in the northeast have more workplaces, while the residential areas in the west have more homebases, which seems plausible.

5.1.2 An algorithmic approach to extract trips

The method that was used to extract trips from the Senegal CDR dataset works as outlined in algorithm 2. It basically scans through the list of available CDR events for every user and day and uses the two functions `detect_trip_start()` and `detect_trip_end()` in order to determine which parts in the sequence of positions define a trip.

The `detect_trip_start()` function requires a distance bigger than a threshold d_{min} between the first positions in the trip in order to filter out switching between neighbors cells to be detected as movement. In order to complete the sparse data, the algorithm makes a few assumptions in order to enrich the trips with likely start and end points (based on the previously extracted home- and work locations):

- The first trip of the day always starts at the homelocation,
- the last trip of the day always ends at the homelocation,
- start and end positions of a trip are "snapped" to the home- or work-location of the user if the first/last position is closer than a threshold distance (d_{max})

Algorithm 2: Main logic of the detection algorithm

Data: A list P_{ud} of (t, c) tuples for user u and day d where t is a timestamp and c a cell
Result: A set of trips T

```
trip_active := False; trip_ended := False
for (t, c) ∈ P_ud do
    if not trip_active then
        trip_active = detect_trip_start()
    if trip_active then
        trip_ended = detect_trip_end()
    if trip_ended then
        T.append(store_trip())
```

The details of the method including the functions to detect start- and ends of a trip are described in Gundlegård et al. (2015).

5.2 Trip extraction from STEM data

The method described in 5.1 is built on the assumption that only very few positions are available for each user and day. While this is true for CDR data, STEM data offers much more continous data as it is based on handovers for data connections (which are usually persistent at least for smartphones). Therefore if STEM data is used as input, a different approach to extract trips should be used.

A possible way could be to use a continous metric that is calculated based on the STEM data for a user and indicates how likely the user moved at each point in time. If this value continuous to stay above a threshold for a certain time the user probably made a trip. As no real STEM dataset was available for this thesis such a method has only been implemented in early stage and it was not used for a complete network loading. An example is shown in figure 5.2, where a movement efficiency metric was used that is based on the distances between antennas that a user connected to within an hour. For example if the user is connected to an antenna close to the one that he/she was connected to one hour ago, the value will be close to 0. If he/she is connected to an antenna that is far away from the one that the user was connected to an hour before, the value will be close to 1.

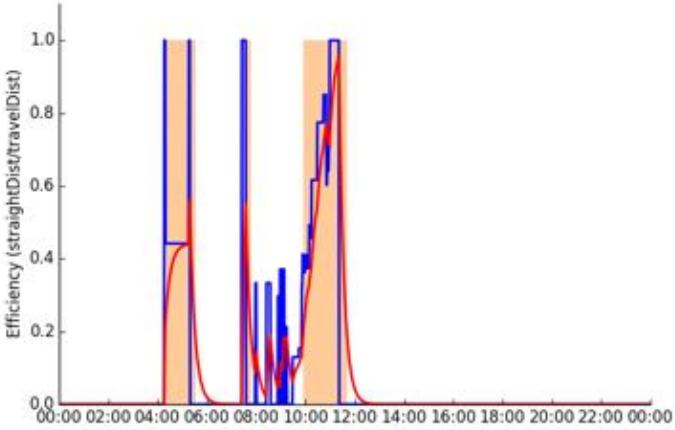
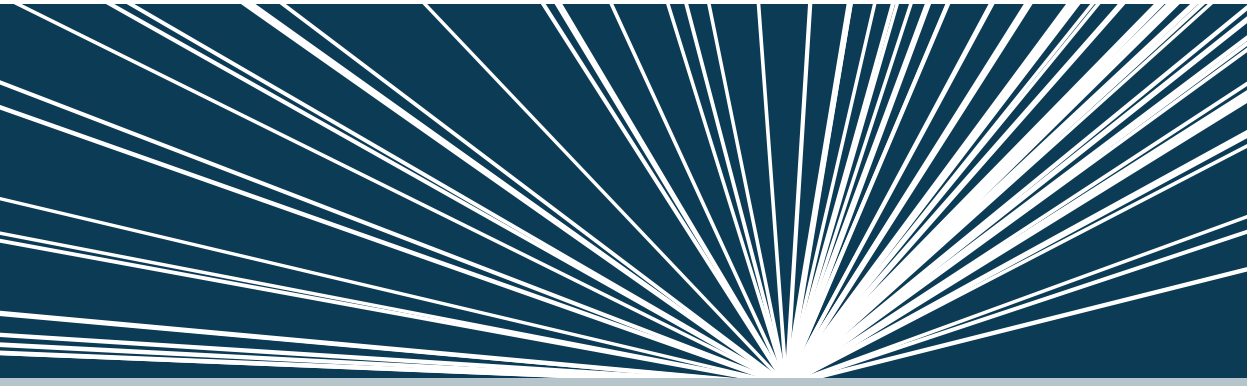


Figure 5.2: Illustration how trip extraction from STEM data could work using a movement metric. The blue line is the original value, while the red line was filtered through a lowpass filter. Shaded areas indicate possible trips.

Chapter 6

Travel demand estimation

As the available cellular network data always only represents a sample of the total population, a simple aggregation of the extracted trips is not enough to get a reasonable estimation of the actual travel demand. To estimate the travel demand for the whole population using the available sample, some form of scaling is needed. In this thesis two methods are used. The first one uses cellular network data only and was used for the Senegal dataset as there was no other data available. It both scales the travel demand to the whole population and temporally distributes it over time-slices of the day. Ideally however, we have additional information like official census data available. For the L.A. dataset a method developed by researchers at UC Berkeley was used that combines cellular network data and census data in order to calculate an OD-matrix.



6.1 Estimating OD flows from cellular network data without detailed census data

In the case that we only have cellular data and no other statistics about the population except the total number of people in the area of interest, we use a relatively simple scaling method (see figure 6.1). The basic idea is that we first estimate the total number of people living in each cell by distributing the total population using the home-locations that we have estimated from the cellular data (see chapter 5.1.1). All trips that the users that have their home base in a particular cell will be scaled such that they correspond to the population estimated for this cell.

The method generates a time-sliced OD-matrix, meaning that the travel demand is separately estimated for each hour of the day. We define that the OD-matrix for hour h contains all travel demand generated by trip that *started* in hour h. The method is optimized for CDR data, assuming that it has very sparse data points only. Therefore for many trips it is difficult to precisely determine their start- and end-times. Therefore we use a trip-time distribution created from the trips that have precise time information.

6.1.1 Trip scaling

As the extracted trips from the CDR data represent a sample of the total population, we need to scale it in order to get an estimation of the actual travel demand. To do so we first use the home-locations estimated described in chapter 5.1.1. We aggregate the data in order to assign a number of user that live in each cell out of the sample of user that we have data for. Using this and a value for total population of the whole area from an official source we estimate the "real" population in each cell as follows:

Definition 6.1: *Cell population*. The population in each cell (number of persons living inside the cell) is estimated as

$$cell\_population(cell) = total\_population \cdot \frac{homebase\_count(cell)}{total\_homebase\_count} \quad (6.1)$$

Here the *total_population* is the total number of people in the system

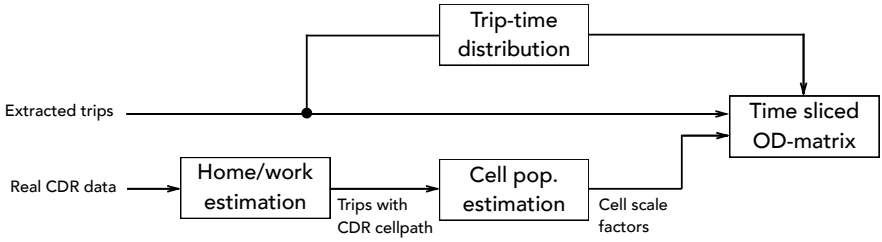


Figure 6.1: The OD-matrix estimation model

(e.g. from the official population statistics). Using this cell population we define a trip scale factor:

Definition 6.2: *Trip scale factor*. Let u be a user and h_u be the user's homebase cell. Then we define the trip scale factor as

$$s(u) = \frac{cell\_population(h_u)}{homebase\_count(h_u)} \tag{6.2}$$

For each trip the trip scale factor $s(u)$ for the user associated with the trip is used. For example if user 1 makes a trip and $s(1) = 4.8$ then, the trip will account for a travel demand of 4.8 people instead of just one. Note that as the scaling equation 6.2 is defined on per-cell level, a siginificant amount of data has to be available for the users in living each cell. If data is only available for 1% of the people living in a cell, there is a high risk that the travel demand estimation becomes very inaccurate.

6.1.2 Trip-time distribution

Many trips extracted from CDR data have a very unclear start/end time. For example, if only one call in the morning at home and one call in the afternoon at work was made, it is unclear when the trip actually happened during that time span. To compensate for that, we select all trips that are timely well-defined (where the time corridor is very small) and use the time distribution of those trips to guess the probabilities for different trip start times within the corridor.

Definition 6.3: *Trip start time corridor*. Given a trip where the first available CDR is from time t_{first} and the last CDR of the trip from t_{last}, we define the *trip start time corridor* $T_c := (t_{start}, t_{end})$ as follows:

$$t_{start} := \quad t_{first} \tag{6.3}$$

$$t_{end} := \quad t_{last} - \frac{d}{50km/h} \tag{6.4}$$

Here d denotes the total distance of the trip measured by the cumulated Euclidean distance between the antenna towers the user connect to during the trip in kilometers. We assume an average travel speed of 50km/h and exclude the time that the used must have used at least for the journey from the start time corridor.

The most simple approach to assign the trip to the OD time-slices, would be to calculate the number of time-slices (hours) that are included in the starttime corridor and then distribute the trip evenly among these time slices. For example a trip with a starttime corridor from 7:00-8:59 o'clock starting at cell a and ending at cell b would contribute the OD-flow by half to the 7 o'clock time-slice and by half to the 8 o'clock time-slice. Due to the timely sparse CDRs, the starttime corridor can be very long for many trips (in order of magnitude of several hours). Therefore spreading out the trip

evenly among the hours might not be very realistic. For example if a the starttime corridor goes from 1:00 to 8:00 it is much more likely that the trip was started in the morning between 6:00 and 8:00 and very unlikely that it started during nighttime. Therefore using a uniform distribution is not optimal.

Instead we use the following trick to get a better distribution: First we only select the trips with a very good start time information from the dataset (starttime corridor $\leq 60min$). Then we calculate a timedistribution based on these trips with good information only both for each OD-pair and globally for all trips. Finally this distributions are applied instead of the uniform distribution to spread out the trip over the starttime corridor. In detail the process works like this for each trip in an OD pair:

1. Calculate the start time corridor T_c
2. Check if the trip-time distribution for the OD pair sums up to at least 8 inside T_c
    a) If yes, distribute the trip using the OD pair trip-time distribution
    b) Else, distribute the trip using the global trip-time distribution

This ensures that we use OD-pair specific information if there is enough data available. However, if there are only very few trip in an OD pair, we fallback to the global time-distribution for all trips.

6.2 Estimating OD flows by fusing cellular network data with census data

Making a good estimation of the traffic demand solely using cellular network data is impossible as no carrier covers all people travelling in an area. Even though the data can be used as a sample from which the demand of the total population can be estimated as described in chapter 6.1, the sample, which is defined by the customer base of one specific carrier, might not be representative of the whole population. To solve these issues it is useful to combine the cellular data that gives the information which areas people frequently travel between with official census data that can give information about where people live and work.

The development of such a method was not part of the thesis. However for some of the experiments in chapter 9 a method that fuses cellular network data and census data was used, which was developed by researchers at UC Berkeley. The idea of the method is to fuse census data about the number of home- and workplaces with cellular network data in order to get an idea which home- and workplaces are connected (where do people work who live in a certain area?). This results in a OD matrix that describes the commute demand from home to work and vice versa. Trips to other destinations except home and work are disregarded.

The method uses data from the official United States of America (U.S.) census. For the census, the area is divided into TAZs (see figure 6.2). The size the TAZs can vary a lot from $100m^2$ up to more than $1000km^2$. Generally the TAZs are smaller in dense areas and much bigger in sparsely populated areas. Additionally real STEM data of a big carrier in the area was used to estimate the home- and work locations of users. No use of the time information and other positions of the users is made. The commute matrix was calculated as follows:

1. Find the home and workplace cell tower (search for the cell used most often during day and nighttime respectively) for each user.

2. Move the home/workplace location taking census data into account.

3. Scale the OD-matrix using IPFP using census and car modal split.

Instead of using the centroid of the home/work cell directly, the position is moved using census data in step 2. This is done by "distributing" the user among TAZs close to the home/work antenna based on distance between TAZ and antenna and number of home/workplaces from census. In step 3 IPFP is used in order to scale the matrix created in step 1–2 from the sample to represent the complete population. This can be done as the margin totals known from census (number home/work locations in every TAZ). It is also taken into account that the number of people travelling is not equal to the number of vehicles. Therefore modal split and occupation factors for cars/busses are used in order to get an OD-matrix that represent the traffic flow in terms of vehicles rather than in terms of the number of people.

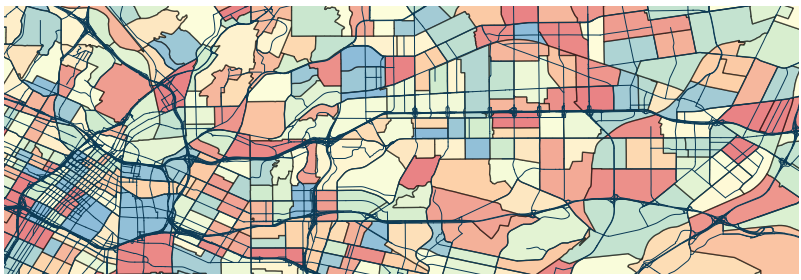
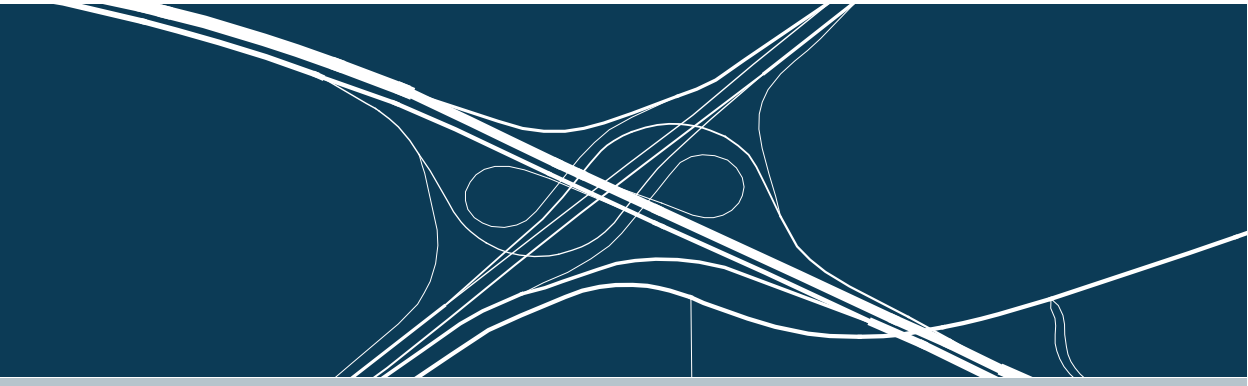


Figure 6.2: TAZs in the L.A. region and road network for reference

Chapter 7

Route estimation

To perform a network loading based on the demand stored in an OD matrix, the flows in each OD-pair have to be distributed among the different possible routes. From the cellular network data, we know the different cellpaths of the users. This gives approximate information about which way each user took. However, to estimate the flows on the different roads in the network, we need to estimate which links the users probably took. This chapter will introduce different algorithms to solve this problem.



7.1 Antenna clustering

The number of possible OD pairs between cells grows quadratic in the number of cells. As the route calculation/network loading has to be done for every single OD pair, computation time for this grows quadratic, too. Therefore it is desired to reduce the number of antennas before starting the network loading in order to lower the computation time. Of course reducing the number of cells will always have a negative impact on the accuracy of the route calculation, as not all of the original information is used. Clustering antennas is essentially a trade-off between computation time and route estimation accuracy.

The goal of antenna clustering is to reduce the amount of cells used in OD matrix and trip data to facilitate computation of routes while minimizing the impact on the result. In order to do so, a hierarchical clustering algorithm (see chapter 2.3.5) is used that groups all cells with a distance below a certain threshold into one cluster (single-linkage criterion).

The algorithm is implemented by first calculating the distance between all antennas. Based on this an abstract graph is created where the cells are the vertices. In the case that two antennas have a distance below the threshold, they are connected in the graph with an edge. Once the graph is build, the next step is to calculate its *connected components*. Each connected component of the graph forms a cluster. Finally all antennas in a cluster are replaced with a new virtual antenna at the centroid of the original antennas in the cluster. Figure 7.1 shows a single-linkage hierarchical clustering.

Definition 7.1: *Connected component*. A connected component of a graph $G = (V, E)$ is a subgraph $G_c = (V_c, E_c), V_c \subset V, E_c \subset E$ such that a path in G_c exists between any pair of distinct vertices $a, b \in V_c, a \neq b$ and no more vertices or edges can be added to G_E.

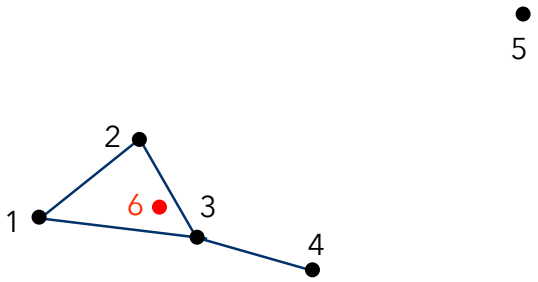


Figure 7.1: Antenna clustering using single-linkage hierarchical clustering. The edges represent distances between two antennas between cells that are below the distance threshold. Antennas 1-4 are in the same connected component and will therefore be replaced by the new antenna 6 at the centroid of 1-4. Antenna 5 has no neighbor antenna closer than the threshold distance and will therefore remain as it is.

The advantage of clustering the antennas in that way is that it tends to make the smallest Voronoi cells bigger, while the ones that already cover a big area in the original data are kept as they are. This means that the algorithm will remove detailed information where the information was very good before already. For example places where many people gather like stadiums or big shopping centers often have multiple antennas, which potentially gives information about in which area of the stadium or the shopping center the user was. This information is however not relevant when estimating the traffic flows on the road on a city-wide scale. For that it is enough to know that the user was in the ara of the shopping center or stadium, thus antennas that are very close together should be clustered.

One caveat with this method is that in theory clusters can grow infinitely (imagine a long chain of antennas with a distance below the threshold between them). For the L.A. simulated STEM data (SSTEM) dataset this did however not occur in practice when using a distance threshold of up to 500m. Running the clustering on the original set of 805 antennas yielded 426 clustered antennas with a threshold of 150m and 393 with a threshold of 500m. The computation time of the clusters was in the order of a few seconds.

7.2 Network simplification

The route estimation for each OD-pair involves a large number of shortest path computations. Reducing the computation time of each shortest path calculation speeds up the whole route estimation and network loading procedures significantly.

The two main ways to reduce shortest-path computation time are the choice of an efficient algorithm and reducing the size of the input data (i.e. the transportation network).

To reduce the network, we observed that many links in the transportation will actually never be used when calculating the *linkpath* for a cellpath (the links used most likely when travelling along a given cellpath). This is due to the fact, that the CDR gives no insight on the travel inside a cell area but only between cells. However many links will only appear in shortest paths, when the destination is actually inside the cell, but never when only passing through. As only the latter is interesting for the cellpath routing, an algorithm has been developed that simplifies the network such that all links only relevant for cell-internal traffic are being removed, while all links relevant for inter-cell travel are kept. Figure 7.2 shows an example of the effect of the network simplification.

Definition 7.2: *Border junctions*. For each Voronoi cell $v \in V$ the set of *border junctions* $B(v)$ contains all nodes inside v incident to any link crossing the border v.

This algorithm significantly reduces the network size and thereby the

Figure 7.2: Example of the network simplification for a part of Dakar. Red links are removed in the simplified network, blue lines show the Voronoi cells

Algorithm 3: Network simplification

Data: Full transportation network N_{full}, set of Voronoi cells V
Result: Simplified network N_{simple}

```
1 for v ∈ V do
2     Add all links crossing the boundary of v to N_simple
3     Find all border junctions B(v)
4     Between all pairs of junctions in B(v), calculate the shortest path
5     Add all links used by at least one shortest path to N_simple
```

average route computation time. At the same time it doesn't change the routes computed, as long as they start and end in a border point, which is always the case for the cellpath routing algorithms presented in 7.4:

Proposition 7.1: *Inter-cell path integrity*. If P is a shortest path in N_{full} with start and end at arbitrary border junctions s and t in N_{full}, then all links in P are also part of N_{simple}.

Proof: Let P be a shortest path in N_{full} with start and end at arbitrary border junctions s and t in N_{full}. Let B_P be the subset of the junctions that P passes by which are border junctions to a Voronoi cell. For any consecutive junctions (u, v) in B_P, either (1) u and v are in different Voronoi cells or (2) u and v are in the same Voronoi cell. In case (1) they are connected by a direct link between u and v that crosses the border between two Voronoi cells, which is part of N_{simple} due to line 2 in the algorithm. In case (2) the links between u and v in P are equal to the shortest path between two border junctions of the same Voronoi cell, which is added in line 5 in the algorithm to N_{simple}. As all links between subsequent border junctions in B_P are part of N_{simple} and the start- end endpoints of P are the first respectively last point in B_P (as they are border junctions by assumption), all parts of P are in N_{simple}.

7.3 Calculating optimal waypoints

Two of the routing algorithms that will be presented in chapter 7.4 use waypoints in order to make the calculated route follow the cellpath of the user. The most naive routing algorithm, which is actually used in some research papers as a first approximation, would be to use the antenna locations as waypoints. The resulting route would start at the first antenna tower, follow the shortest path to the second antenna in the cellpath and so on. However as we know that one cell-tower can cover an area with a radius of several kilometers, the assumption that the user directly passes by the cell-tower does not hold. In fact, such an algorithm would generate lots of unrealistic flows caused by the detours to reach the cell-towers.

To avoid this problem we want to use a more realistic selection of waypoints for a certain cellpath. As it is guaranteed that to enter a Voronoi cell, the user has to pass through one of the border junctions of the cell it is enough to consider the border junctions as possible waypoints. The most naive way to find a set of optimal waypoints would be to select one border point in each cell as waypoint and calculate a shortest path between all waypoints. This would be repeated for all possible combinations of waypoints and the set of waypoints that gives the lowest total travel time, would be selected as the optimal set of waypoints.

However this brute-force approach is not realistic to be used in a real-world scenario as the high number of possible waypoint combinations for each cellpath requires an unreasonable amount of costly shortest-path computations. Therefore, we reduce the number of combinations to check by splitting the cellpath into parts.

Definition 7.3: *Cellpath part*. Let $p = (b_0, b_1, ..., b_n)$ be a cellpath. Then we call $p^i = (b_{i-1}, b_i, b_{i+1})$ the i-th *cellpath part* of p.

The new simpler problem can be boiled down to the following: Given a *cellpath part* (a, b, c) consisting of three consecutive cells in a cellpath, select a waypoint in cell b that is *optimal* when coming from cell a and heading to cell c. Optimal here means, that the addition of the waypoint to the user's route has the lowest negative impact on the travel time among all candidate waypoints. This problem is much faster to solve as

1. only the candidate waypoints for every cell in the cellpath have to be checked, not all combinations (for example for a cellpath $[4, 7, 2, 20]$ if there are 10 candidate waypoints in cell 7 and 8 in cell 2, only $10+8 = 18$ waypoints have to be check instead of $10 \times 8 = 80$ combinations)

2. waypoints have to be calculated just once for each unique *cellpath part* (see definition 7.3) and finding all waypoints for a certain cellpath can be done using a lookup table.

In order to further reduce the computation time, candidate waypoints could be preselected using the straight-line distance they would give and

only calculating the real shortest path for a certain number of candidate waypoints that give the best straight-line distances. However depending on the road network, the assumption that waypoints closer to the straight line are not always guaranteed to give the shortest path in travel time. Also, shortest-path calculations using a contraction hierarchy showed to be so fast that the benefit of this heuristic is fairly small and does not out-weight the possible errors caused by choosing suboptimal waypoints. In all experiments in this thesis a real shortest path is therefore calculated on the road network for each waypoint in the candidate set. The waypoint that gives the route with the lowest travel time (based on free flow speed) is saved for each of the cellpath parts (see algorithm 4). In the algorithm, $B(b_i)$ is again the set of border junctions of cell b_1 (see definition 7.2) and $d_c(a, b)$ is the minimal cost for a route between points a and b.

Algorithm 4: `waypoint`(p^i)

Input: cellpath segment $p^i = (b_1, b_2, b_3)$
Output: waypoint w
1 a := node in $B(b_1)$ with lowest distance to antenna position b_2
2 c := node in $B(b_3)$ with lowest distance to antenna position b_2
3 **return** $s \in B(b_2)$ *with lowest* $d_c(a, s) + d_c(s, c)$

Figure 7.3 illustrates the selection of a waypoint for the cellpath segment consisting of cells A, B and C. The heuristic first selects a start and end node in A and C by line-of-sight distance to the antenna in B. Note that these start- and endpoints are only used as helper points for the waypoint selection. They are never added to the final route for the cellpath. For candidate waypoints points 1, 2, 3 (the border points of B), a shortest path is calculated using OSRM (see chapter 2.3.2) and the lowest cost path gives the final waypoint (in this example probably point 2). Lowest cost here means shortest travel time according to OSRM (which is based on the attributes of the underlying OpenStreetMap links about permitted max speed, type of the road etc.).

Note that a real implementation of this algorithms needs to cover the special case where a Voronoi cell has no point inside it as well. The implementation used for the experiments in this thesis uses a candidate waypoint set of 10 border junctions with the smallest distance to the antenna of original cell, if the original Voronoi cell contains not a single intersection. However for the tested datasets this problem occurred only with few cells (not more than 2% of the cells) for both datasets.

The selection of the startpoint for a route based on a given cellpath works similar to the waypoint calculation expect that it only uses the first two cells in the cellpath (see algorithm 5). Analogously, the selection of the endpoint only used the last two cells in the cellpath. This selection of start and endpoint is used for all of the cellpath routing algorithms, that

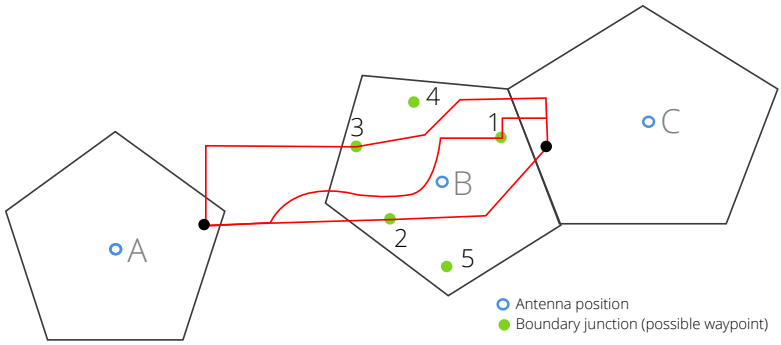


Figure 7.3: Illustration of an optimal waypoint selection in cell B when coming from A and heading to C

are presented in the following section.

Algorithm 5: `startpoint`(p_s)

Input: cellpath start segment $p_s = (b_1, b_2)$
Output: startpoint w

```
1 b := node in B(b_2) with lowest distance to antenna position b_1
2 return s ∈ B(b_1) with lowest d_c(s, b))
```

7.4 Cellpath routing algorithms

After executing the preparation steps described in chapters 7.2 and 7.3, we now need to finally define an algorithm that maps a given cellpath to a route on the road network. To do so we can basically use two sources of information: the road network giving the travel times and thereby allowing it to find shortest paths between arbitrary points in the road network and secondly the cells that the user visited (the cellpath). The following sections introduce three different algorithms mainly differentiated by how they balance between the two information sources.

7.4.1 Shortest-path Routing

The first algorithm used in the experiments is called *Shortest-path Routing* and is a very simple algorithm, mainly for comparison. It doesn't use any of the cellpath information apart from the first and last cell in the cellpath and only relies on the shortest path between start and destination cells (see algorithm 6). The resulting network loading using this algorithm is one where streets have infinite capacity and everybody has perfect knowledge

about the free flow travel times on each links and always uses the path with the lowest travel time. In the algorithm `route`(a, c) represents a functions that calculates the shortest path between the two points a and b and returns the links used on the route.

Algorithm 6: Shortest-path Routing

Input: cellpath $p = (b_1, b_2, ..., b_n)$
Output: linkpath $l = (l_1, l_2, ..., l_m)$
1 $a :=$ `startpoint`(b_1, b_2)
2 $c :=$ `endpoint`(b_{n-1}, b_n)
3 $l :=$ `route`(a, c)
4 **return** l

The reason to run this very simple algorithm is for comparison only. Comparing the result of this algorithm with algorithms that actually use the cellpath information can give insights about how these algorithms change the result and in which parts of the network people use routes different from the shortest path in reality. There may be many reasons that people take a different route than the shortest path, but one of them certainly is that the shortest path is overcrowded and it is therefore better to take another alternative route. So a comparison of a algorithm following the real cellpath with the shortest-path scenario can possibly give an indirect indicator of a traffic jam on a road section.

The shortest-path scenario serves also a base scenario for the validation. The goal for all algorithms using cellular data is of course to make an estimation of the link flows that is at least as good as the naive shortest-path network loading which is not using this information. So outperforming this base-scenario is the minimum goal for the cellpath-routing algorithms presented in the following sections.

7.4.2 Strict Voronoi Routing

The *Strict Voronoi Routing* algorithm in contrast to Shortest-path Routing relies fully on the information given by the cellpath of a user. It is built around the assumption, that the user has to enter every Voronoi cell that is in the cellpath (meaning that he has to use at least one link inside the cell. Between the cells a shortest path is calculated on the road network. The computation of a route for a given cellpath is performed according to algorithm 7. In the algorithm the function `route(x,y)` is assumed to return set of links that belong to the shortest path between nodes x and y. As algorithms behind this function, shortest path algorithms like *Dijkstra* or $A*$ can be used. To get a realistic route estimation, the link costs used for the shortest path computations reflect the free flow travel times, not only the distance.

Algorithm 7: Strict Voronoi Routing

Input: cellpath $p = (b_1, b_2, ..., b_n)$
Output: linkpath $l = (l_1, l_2, ..., l_m)$

```
1 a := startpoint(b_1, b_2)
2 c := endpoint(b_{n-1}, b_n)
3 l := route(a, waypoint(p^1))
4 for i ∈ {2, ..., n − 1} do
5     l := l ∪ route(waypoint(p^{i−1}), waypoint(p^i))
6 l := l ∪ route(waypoint(p^{n−1}), c)
7 return l
```

The function $\texttt{waypoint}(p^i)$ returns the optimal waypoint for the i-th part in the cellpath that was calculated according the method described in chapter 7.3. This means that for every cell in the cellpath (apart from the first and last) exactly one waypoint inside the cell is selected.

7.4.3 Lazy Voronoi Routing

The problem of a strict Voronoi routing, as presented in chapter 7.4.2, is that it is built on the assumption that the user has to enter every cell in the cellpath. Even though the fact that the user connected to an antenna implies that he was somewhere near the antenna, Voronoi cells are not a perfect representation of the real coverage of the cell. The real coverage depends on many factors like the landscape (heights), the power of the antenna as well as it's orientation. Also other factors like network balancing where the operator switches the user to another cell because it has more capacity can be a reason why the user is connected to a particular cell. So, there is no guarantee that the user really entered the Voronoi cell. It is more likely that the user was inside the cell than in a place in the other end of the city, but he/she might as well just passed by the Voronoi cell closely without actually entering the Voronoi cell. This can lead to a route that does not share a single common link with the actual route of the user when using as in the example illustrated in figure 7.4 (a). Therefore we define a new algorithm that will be called "Lazy Voronoi Routing" from now on. This algorithm relaxes the constraint to enter every cell as in the strict Voronoi routing, while it still uses the information about the visited cells by making it more likely to pass through those cells or nearby.

The algorithm is based on an idea proposed by Leontiadis et al. (2014). They suggest, instead of inserting waypoints to make the route follow the cellpath, to modify the cost on the links of the road network before calculating a shortest path. For example if we assume the the cell with ID 5 is in the cellpath, all links inside cell 5 will get a lower cost in order to make it more likely for the estimated route to pass through this area. This still leaves the

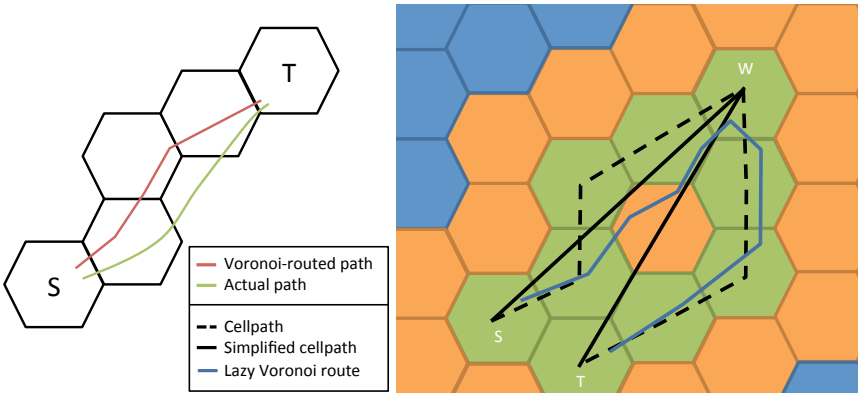


Figure 7.4: Both figures show a route calculation for a cellpath starting at cell S and ending in cell T. Figure (a) on the left illustrates a case where Strict Voronoi Routing fails. Black outlined Voronoi cells outline the cells inside the cellpath. Figure (b) on the right illustrates the Lazy Voronoi Routing. Green Voronoi cells are part of the cellpath, orange cells are close to a cell in the cellpath and remaining Voronoi cells are shown in blue.

possibility for the route to pass by the cell if there is a much better way that doesn't cross the cell at all (in the case that it has still a lower cost even after modifying the link costs). There is a high chance that using this method in a case like in figure 7.4 (a) will recover the correct route of the user.

However, in real cellular data, cellpaths are often not as straight forward following the line-of sight as in figure 7.4 (a). Some trips will also contain more round-trip-like cellpaths as in figure 7.4 (b) (green cells are part of the cellpath), where start (S) and target cell (T) are very close together or even identical, but the cells in-between still indicate that the user probably moved. In that case only modifying the cost for the visited cells will not make the route follow the cellpath in any way, instead most likely only the shortest path between S and T would be returned, which is not a correct representation of the users movement.

This problem is tackled in an algorithm that will be called Lazy Voronoi Routing from now on, which combines the idea of the Strict Voronoi Routing with the modified link cost routing proposed by Leontiadis et al. (2014). The algorithm consists of three main steps (see algorithm 8):

1. Choose an intersection as start- end endpoint (this works exactly the same as in all other algorithms as well).

2. Segmentize the cellpath using line-simplification.

3. Calculate the route using waypoints as in Strict Voronoi Routing based

on the simplified cellpath, but using modified link costs when calculating the route for each segment.

Algorithm 8: Lazy Voronoi Routing

Input: cellpath $p = (b_1, b_2, ..., b_n)$
Output: linkpath $l = (l_1, l_2, ..., l_m)$

$a := \texttt{startpoint}(b_1, b_2)$
$c := \texttt{endpoint}(b_{n-1}, b_n)$
$m, S, p_s := \texttt{segmentize}(p)$
$l := \texttt{routeLazy}(a, \texttt{waypoint}(p_S^1), S^0)$
for $i \in \{2, ..., m-1\}$ **do**
  $l := l \cup \texttt{routeLazy}(\texttt{waypoint}(p_s^{i-1}), \texttt{waypoint}(p^i), S^{i-1})$
$l := l \cup \texttt{routeLazy}(\texttt{waypoint}(p_s^{n-1}), c, S^{m-1})$
return l

Step 2 treats the cellpath as a geometry by connecting the antenna towers that are part of the cellpath to a line (dashed line in figure 7.4 (b)). This line is processed by the Ramer-Douglas-Peucker line-simplification algorithm (see chapter 2.3.3), which creates a line that only contains the most extreme points of the original line in order to keep its shape as good as possible. We use the points that are kept after the line-simplification process in order to split the original cellpath into segments at these cells. The function $\texttt{segmentize}(p)$ in the algorithm takes a cellpath and returns the number of segments (m), a vector containing a list of cells included for each segment and ($S = (S^0, ..., S^{m-1})$) and list containing the simplified cellpath $p_s = (p_s^0, ..., p_s^{m-1})$.

In step 3 the actual route is calculated using a waypoint like in Strict Voronoi Routing, but only for the cell that are part of the simplified cellpath (which could be just one or two instead for an original cellpath with 20 cells depending on it's shape). For each segment then a route is calculated from the waypoint that belongs to the first cell in segment that the cell waypoint that belongs to the last cell in the segment. Instead of the original link cost $c(l_i)$ for link l_i a modified link cost $c_m(l_i)$ is used in the route calculation, which is defined as follows:

$$c_m(l_i) = \begin{cases} \alpha \cdot c(l_i) & \text{if } \exists c \in s: \quad l_i \text{ intersects with } V(c) \\ \beta \cdot c(l_i) & \text{if } \exists c \in s: \quad l_i \text{ intersects with } V_E(c) \\ c(l_i) & \text{otherwise} \end{cases} \tag{7.1}$$

Here s is the set of cells in the cellpath segment and $V(c)$ the Voronoi cell of cell c. $V_E(c)$ is a polygon that represents the buffer zone with a width of d_E around $V(c)$. In the algorithm the function $\texttt{routeLazy}(a, b, S)$ calculates the shortest path from point a to point b using modified costs in and around

the cells in S and returns the list of used links on the route. In total there are four parameters used in the algorithm:

- d_S, threshold distance used for line-simplification (lower values cause more segments/waypoints)
- d_E, distance up to which the link cost is reduced if it is not in a cell in the cellpath but very close
- α, cost reduction factor for links inside the visited cells ($0 < \alpha < 1$)
- β, cost reduction factor for links near a visited cell ($\alpha < \beta \leq 1$)

7.5 Network loading

The presented algorithms allow to estimate one route for a given cellpath. In the future this could be extended to the k most likely routes for a given cellpath. For the network loading however, we want to load the travel demand for an OD-pair using all different cellpaths that can be found in trips between this OD-pair, as people can take different routes. When analysing all trips made in an OD-pair, some cellpaths will occur more often, while others might only appear once. By dividing the number of occurrences of each cellpath by the total number of trips for every OD-pair, a cellpath distribution can be obtained. This cellpath distribution is used to obtain the probability that a specific cellpath is used in an OD-pair. This probability is used in order to distribute the total travel demand in an OD-pair among the different cellpaths.

For each cellpath a route on the road network is estimated using one of the cellpath routing algorithms. Looking at the route calculation for one cellpath, this could give routes as shown in figure 7.5. While the Shortest-path Routing algorithm yields a route between start- and end-cell with the lowest free-flow travel time, the Strict Voronoi Routing add waypoints in every cell that is in the cellpath in order to make the route follow the cellpath. The figure also shows that waypoints are sometimes not placed optimal, leading to small detours being added to the route. The Lazy Voronoi Routing in this case perfectly recovers the original route by lowering the costs on links that are within the cellpath.

For the network loading, this route estimation is repeated with the chosen algorithm for every cellpath in every OD-pair. Then the OD-flow is assigned to the different routes based on the cellpath distribution. For example if there are three different cellpaths for an OD pair and one of them appeared in two trips, while the two others only appeared in one trip each the route that belongs to the first trip would get 50% of the flow assigned, while the two other only get 25%. A way to reduce the computation time of the network loading is to limit the maximum number of different cellpaths used

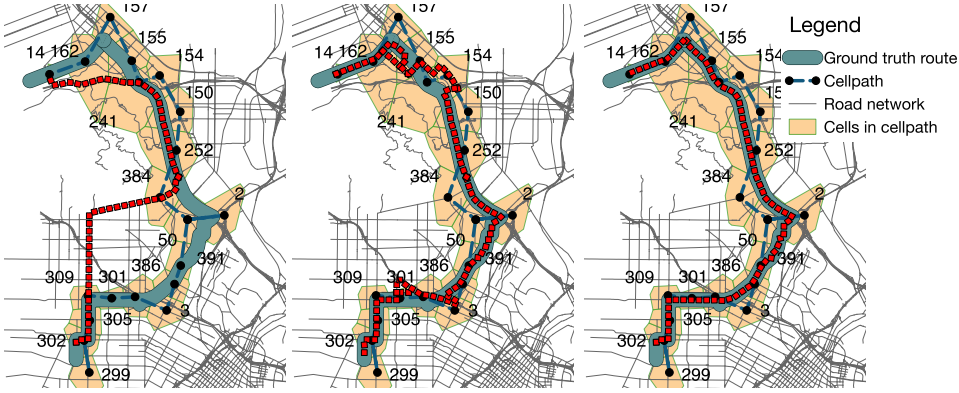


Figure 7.5: Example of a route calculation with all three algorithms for L.A.. In red from left to right: (a) Shortest-path route, (b) Strict Voronoi route, (c) Lazy Voronoi route. The default parameters as stated in table 9.3 were used.

per OD -pair by only used the n cellpaths with the highest probabilities in the cellpath distribution.

For a single OD-pair the network loading can look like in figure 7.6. The example is based on data for L.A. and uses an upper limit of five different cellpaths. For Shortest-path Routing the network loading is trivial as only one route (the shortest path between origin- and destination cell) is calculated which gets all the flow assigned. For the cellpath based algorithms the flow is split between five different routes by assigning most flow to the cellpath that occurred most often in trips. It is also visible that Strict Voronoi Routing which tries to exactly follow the cellpath by placing a waypoint in every cell adds flows caused by detours that are added to the route, while Lazy Voronoi Routing gives a smoother network loading with more logical routes. In order to estimate the total link flows, this network loading is repeated for every OD-pair and flows assigned to the same link are aggregated using a Map/Reduce based algorithm (see chapter 4.3).

Even though the route choice might be different at different times of the day in reality, this is currently not reflected in the model. Instead the cellpath distribution used is global for each OD-pair based on all available trips in the OD-pair.

Figure 7.6: Network loading for a single OD pair with (a) Shortest-path routing (b) Strict Voronoi Routing and (c) Lazy Voronoi Routing. The black polygons show the Voronoi cells of the origin and destination of the OD-pair. Numbers indicate the flow assigned to the links. The default parameters as stated in table 9.3 were used.

Chapter 8

Case study: Dakar, Senegal

The presented traffic flow estimation model has been applied to a CDR dataset for Dakar, Senegal in order to test the method. Unfortunately there was no ground-truth data available which made it impossible to formally validate the results. The dataset is based on real data, which makes it possible to see if the algorithms work in general with noisy data. We will also compare the differences in the network loading that are caused by using the three different routing algorithms discussed in chapter 7.4.

8.1 Dataset

For the first case study, a CDR dataset for the country of Senegal which is based on real data of a *Orange* (a big cellular network operator in Senegal) that has been opened for research purposes as part of the Data for Development (D4D) competition is used. The dataset contains CDRs for 320.000 unique users of the network. Every time that the user makes a call or sends a text message, a Call Detail Record is saved. Each record consists of the id of the user, the time when the call was made and most important the cell ID that uniquely identifies to which cell tower the user was connected to when starting the call as shown in table 2.1). In order to get the information where the cell is, another table contains the geo-coordinates of each of the 1666 cell towers that are distributed over the whole country of Senegal.

For privacy reasons and in order to not reveal sensitive information, the data has been obfuscated in several ways. Montjoye et al. (2014) give a detailed description on how the data has been preprocessed. One of the obfuscations is that the timestamps of the CDRs have been rounded down to 10min intervals. This means for example that the call belonging to the timestamp 06:50:00 may have taken place anytime between 6:50:00 and 6:59:59.

Another obfuscation has been applied to the cell-tower locations. First a Voronoi diagram based on the real cell-tower locations was calculated. Then a random location inside each Voronoi cell of the real cell-tower is selected. This location is the one that is available in the dataset that has been made available for research. The effect of this is that the carrier doesn't reveal the exact positions of its antennas and the accuracy of the estimation where a user was at a given time becomes less accurate. Also, in order to prevent tracking of users over a longer timeframe, all users get a new id assigned every two weeks in the dataset. For the experiments in this case study, only one two week interval with constant user ids was used.

Not all 9 million users of the carrier are part of the available dataset. Instead only users that fulfilled the two following criteria have been selected:

1. the phone had at least 1 CDR on 75% of the days in the two-week interval and

2. the phone had not more than 1000 CDRs on average per week (as these are presumably machines or shared phones)

Out of all users that fulfilled these two criteria, a random sample of 320.000 was made available in the D4D dataset. For all experiments in this thesis the data for the two week interval 07.01.2013 – 20.01.2013 was used. This dataset contains about 44 million CDRs for the selected 320.000 users. Unfortunately no other data apart from the CDR data that could have been used as ground truth was available for Senegal.

To estimate the routes of users, a road network is necessary besides the cellular network data. For this case study a road network from the

OpenStreetMap[1] project was used as a source for the network. The data is available to anyone and collected through volunteers of the community. Therefore there is no guarantee that the network is 100% correct. In fact random checks showed some minor issues with certain small roads in Dakar, but they should not be significant for the network loading. For this case study the complete road network was used (see table 8.1 and figure 8.1) and simplified using the algorithm described in chapter 7.2) which reduced the number of links by 32% from 50885 to 34540 (see table 8.1).

Problematic with the raw OpenStreetMap data is, that it is not routable. The reason is that a link in the OpenStreetMap data can span over several intersections. This is inappropriate both for routing applications and to present the network loading. For example could several sections of the same road have different flows if there is an intersection in-between, but it might have just one ID in OpenStreeMap. There for it is impossible to map flows to the OSM IDs.

The solution is to make the network routable before using it for the routing or network loading. Existing tools like *osm2po* can perform this task by splitting all ways at every intersection and giving the split links new IDs. The result is a routable graph where every link is connected to at most two intersections/vertices. The new IDs can be used to uniquely assign a flow to them during network loading. The routable graph also needs to contain a cost for every link. Tools like osm2po can compute a free-flow travel time that can be used as a cost using factors like the length of the link, the class of the link (motorway, residential etc.) and the permitted maxspeed.

8.2 Trip extraction

Extracting the trips from the CDR dataset using the method described in chapter 5.1 yielded about 2,2 trips per day and user. In total 2,7 million trips could be extracted for the two-week period over the whole country. The number of trips extracted for each user remains fairly constant over the week as shown in figure 8.2, with a small dip on Sundays (days 13 and 20).

Figure 8.3 shows how the well-defined trips are distributed over the day based on their starting time. This trip-time distribution has a peak just before 12:00 and a big peak in the evening around 20:00. This distribution differs from what the usual morning/afternoon commute pattern with peak in the early morning and afternoon. However, we did not have acces to information about how working hours and commuting behaviours look like in Dakar. A suspicion is however, that the trip-time distribution is biased by the calling behaviour of people. At times when people make more calls, information is available making more trips "visible". As users probably make less calls in the early morning, less information about their movements is

[1] http://www.openstreetmap.org

Table 8.1: Statistics about the used datasets for Dakar, Senegal. The CDR data contains the whole country of Senegal, however all experiments only analyzes the subset of the data relevant for the capital city (Dakar).

Senegal CDR data

Source	Orange
No. users	320000
No. cells	1666
Cellpath data	real
Antenna positions	real, obfuscated
Avg. cell area*	$6.41 km^2$
Timeframe	Two weeks in 2013
Time information	10min precision

Dakar road network

Source	OpenStreetMap
No. links	50885
No. links (simpl.)	34540
No. vertices	34996
Roads	all roads

*average cell area based on Dakar only

Figure 8.1: Road network used for Dakar based on OpenStreetMap data

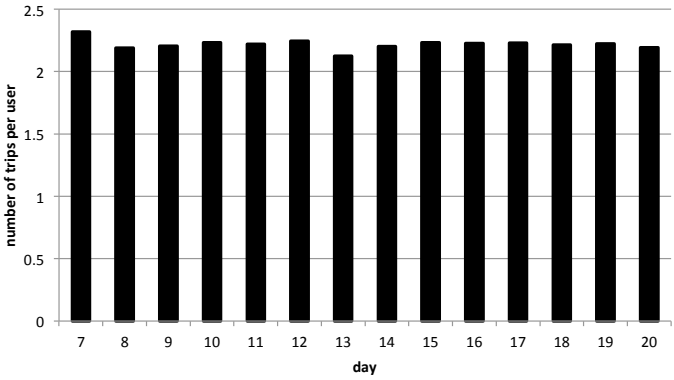


Figure 8.2: Trips extracted per user for each day in the Senegal dataset

available resulting in less extracted trips. This is an inherent problem of using CDR data for traffic analysis. A part of it is however compensated by the trip extraction algorithm that uses the time-distribution of well-defined trips to estimate realistic trip start times.

8.3 Travel demand estimation

Based on the extracted trips, a time-sliced OD matrix for Senegal was calculated according to the method described in chapter 6.1. To upscale the trips, a total population of 13,508,715[2] was assumed for the whole country. The OD-matrix used here was calculated based on the trips made on monday-thursday from a two week dataset. The reason is that these days should have fairly similar characteristics while typical weekend days might have different travel behaviour, requiring a separate OD-matrix.

The resulting total demand (all OD-pairs) for each time-slice of one hour is shown in figure 8.4. Except for the extreme spikes around midnight resulting from automatically added return-to-home trips at the end of the day, the distribution over time after scaling looks more realistic than the one of the extracted trips with peaks at 9 and 19 o'clock that could correspond to a morning and evening commute. The difference between the trip-time distribution and the travel demand time distribution is the result of using only well-defined trips (where the start and end time can be determined relatively precise fromt the data) as the basis for the time calculation instead of all trips. Adding all return trips to home when the last known position of a user was not at home around midnight is of course not optimal. Instead these trips should be probably spread out over a more realistic time period.

The data calculated makes it possible to get a detailed insight which

[2]Source: https://www.amnesty.org/en/countries/africa/senegal/

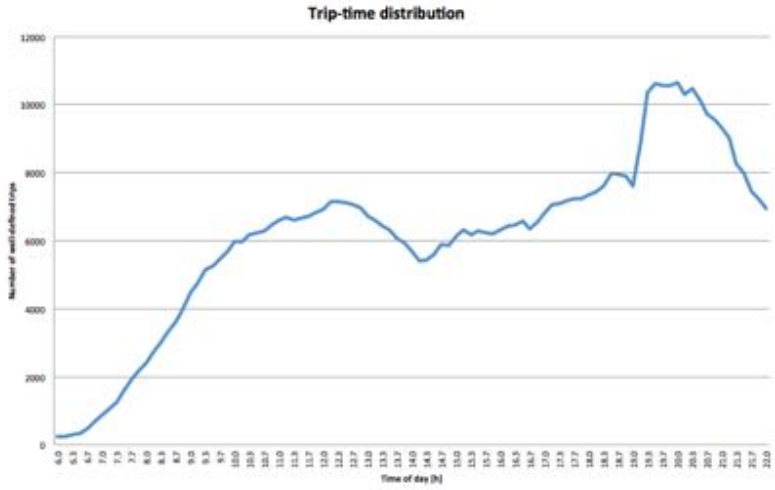


Figure 8.3: Trip-time distribution (two weeks dataset, monday-thursday). Automatically added back-home trips at the end of the day are not included in this graph.

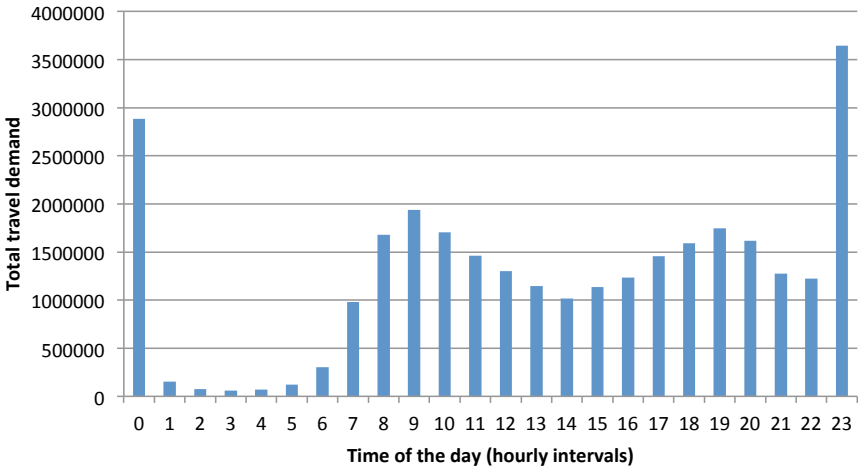


Figure 8.4: Total travel demand (all OD-pairs) estimated for each one-hour interval of a typical weekday

Figure 8.5: OD flows (number of people) in Dakar from one selected cell to all other cells between 16:00 and 16:59

areas of the town people travel between at different times of the day. An example is shown in figure 8.5, shows the travel demand from one selected cell to all other cells on a typical afternoon hour. Without other data to compare to it is however difficult to judge how realistic this data is. It should also be noted that the travel demand is based on upscaling of the trips for the whole population. This implies that the resulting OD-matrix contains the number of people travelling, which is not equal to the number of vehicles. In order to estimate the number of vehicles using the road network, information on the modal split would be necessary. Additionally it would be necessary to have information about the average load (number of people per vehicle). As such information was not available, travel demand for this case study is given in the number of people travelling which are assigned using routes optimized for cars.

8.4 Network loading

Figure 8.6 shows the estimated flows in Dakar at 8am. It sticks out that much traffic is assigned to main motorway that spans through the city from the city center in the south to the periphery in the east. As the used travel demand was given in terms of the number of people travelling (see chapter 8.3), the estimated link flows are also given in number of people. At first about 300000 travellers on certain links during one hour might seem to be a lot. However, the number of vehicles should be much lower than the number of people, since according to a Worldbank study[3] most people are either walking or using mini-busses, while the use of private cars is very uncommon. Assuming that the modal split of only 1% for cars respectively 17% public transport, we can roughly estimate the number of vehicles to 9188/hour. Assuming a capacity of 3000 vehicles/lane/hour, the main expressway in Dakar with 3 lanes could actually satuarate this demand, even though there

[3]Source: http://www.gtkp.com/assets/uploads/20091206-110339-8635-ssatpwp80.pdf

	car	public transport
modal split	1%	17%
people/vehicle	2	8
total travellers		350000
travellers	3500	59500
vehicles	1750	7438
total vehicles		9188

Table 8.2: Rough estimation of the number of vehicles on a link with 350000 travellers/hour in Dakar during rush hour

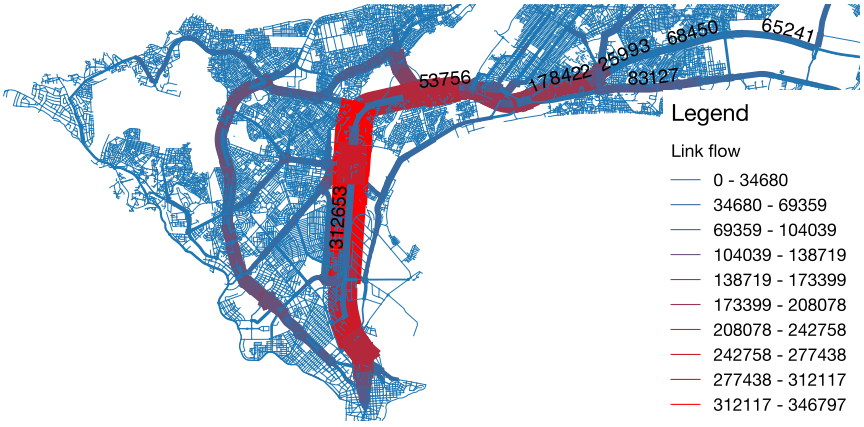


Figure 8.6: Link flows (number of people between 8:00 and 8:59 for a part of Dakar (red = high flow).

would probably be congestion. Note that this estimation is however based on guessed assumptions on the passenger load per vehicle, due to the lack of real data and therefore just a plausibility check and not a reliable estimation of the actual number of vehicles.

To show the impact of the time-sliced travel demand, figure 8.7 compares the flows on a highway segment in the morning to the flow in the afternoon. As it could be expected the main flow in the morning goes into the city center in the south, while in the afternoon the outbound direction of the motorway has a much higher flow that the inbound direction.

While all previous network loading results used the Lazy Voronoi Routing algorithm, we finally compare the differences when using different routing algorithms. As a base for comparison we use the network loading result that has been computed using Shortest-path Routing. In figure 8.8 the result of both Strict Voronoi Routing and Lazy Voronoi Routing compared to Shortest-path Routing is shown for a part of Dakar. As both algorithms increase the length and add more links to the estimated routes, it is not surprising that there is a general increase of flows in both cases. While

for Strict Voronoi Routing almost all links increase in flow, Lazy Voronoi Routing causes also some links to get less flow. This is an indication that even though a route is fastest according to the free flow travel time, people are using different route. This could have several reasons: e.g. the free flow travel time is incorrect or the real travel time on the link is longer due to traffic congestion making people choose different routes. Even though there might be other reasons why people choose a route different from the shortest path as well, this could be one indicator to find links in the network that are often congested.

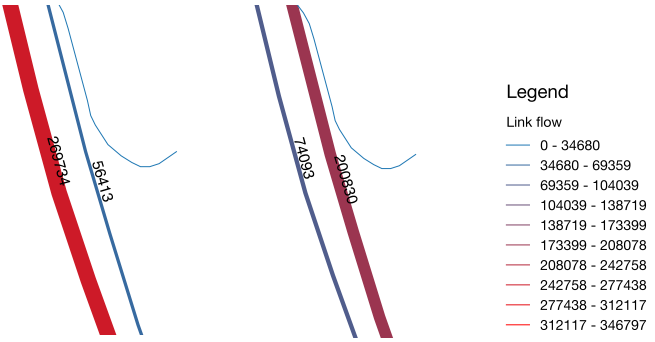


Figure 8.7: Link flows on a highway segment between 8:00 and 8:59 (left) respectively 16:00 and 16:59 (right)

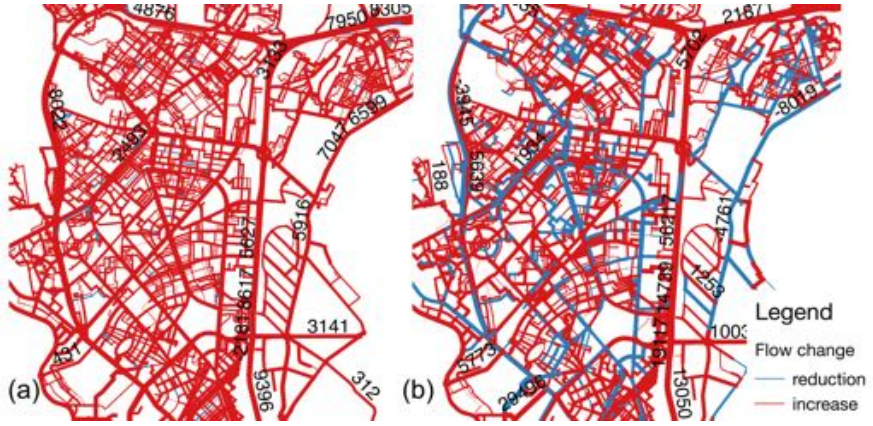


Figure 8.8: Change of the linkflows when using (a) Strict Voronoi Routing or (b) Lazy Voronoi routing instead of Shortest-path routing

Chapter 9

Case study: Los Angeles, USA

For the second case study, a dataset with very different characteristics compared to the Senegal dataset was used that covers a corridor along the I-210 highway in Los Angeles, USA. While the Senegal dataset contains obfuscated real CDR data, the Los Angeles dataset is synthetic apart from the antenna locations, which are slightly obfuscated positions of the real antenna positions in both cases. Apart from the data also the structure of the road network is very different and there is a much higher share of trips made by car in Los Angeles compared to Dakar, which also in general has very different socioeconomic circumstances. This case-study complements the first one as the synthetic data allows to perform a validation of the route estimation algorithms and the network loading.

9.1 Dataset

The lack of ground truth data for Senegal makes it impossible to validate the developed algorithms. Therefore a second dataset covering a corridor along the I-210 highway in Los Angeles, USA was used. As access to real cellular network data is very difficult due to privacy concerns and the confidentiality of the data (see chapter 3.4), this dataset is, in contrast to the Senegal dataset, mostly synthetic (however, some parts are based on real data).

Similar to the Senegal CDR data, the L.A. SSTEM data comprises a table containing the slightly obfuscated real cell tower locations of which there are 805 throughout the area of study. In contrast to CDR data, as used in the Senegal dataset, real STEM data uses data connections ($3G/4G$). This means that the cell that the user is connected to is not only saved when the user makes a call or sends a message, but rather continuously as modern smartphones maintain the data connection while the user is moving even if the user is not actively using the phone. This enables a much better time resolution of the recorded user positions.

Due to access limitation on the real STEM data, a simulated (SSTEM) dataset was generated. The process of running the simulation and generating the data was not part of this thesis. Instead the dataset was provided by researchers at UC Berkeley, who performed the simulation. However, to judge the results it is important to get an idea about how the dataset was generated. The process of generating the SSTEM dataset was performed according to the following three steps:

1. Calculate an OD-matrix based on real STEM data (see chapter 6.2 for details).

2. Run an agent-based traffic simulation that simulates one agent moving through the netwerk per unit of OD flow from the OD matrix and save its route. The simulation is done using a tool called *MATSim* using a model described by Balmer et al. (2009).

3. From the calculated routes, the surrounding cell is saved with a certain probability at every intersection in the route, forming the cellpath of the user.

The resulting OD-matrix for the L.A. corridor contains 690965 OD pairs with a strictly positive flow, the vast majority of them having a flow of 1 vehicle. As the generated OD-matrix only contains home/work trips, also the SSTEM data only contains trips between home and work locations for each user. Virtually, there is one dataset for the way from home to work ("morning commute") and one dataset for the trips from home to work ("afternoon commute"). In contrast to real STEM data and the Senegal CDR data, there is no time information associated with the visited cells (see example in table 9.1).

Table 9.1: An example of an (artificial) SSTEM dataset. Commute direction=0 means morning commute (way to work) and commute direction=1 means afternoon commute (way to home). The cellpath contains an array of visited cell-ids in their order of appearance along the route.

User ID	*Commute direction*	*Cellpath*
1	0	[45,26,78,34]
1	1	[34,198,803,65,45]
2	0	[23,30,42]
2	1	[42,33,23]

As the data was simulated, complete information on the simulated routes is available. This means that the routes that used the same links in the MATSim simulation can be aggregated in order to get virtual link count data (similar to real sensor data) which formes the first ground truth data that network loading results can be validated against.

As the road network, again OpenStreetMap was used as the source. Instead of using the full network all residential links were filtered out, thus only keeping roads with a class higher than residential (see figure 9.1). The road network simplification had only a marginal effect (see table 9.2). This can be explained by the fact that all residential links have been removed before already before the road network simplification algorithm was run. Another reason is that grid-like street systems like in many parts of Los Angeles yield many equally fast parallel routes usable for inter-cell travel.

9.2 Route validation

Comparing single routes reconstructed by one of the algorithms presented in chapter 7.4 from cellular network data with the original route allows to solely validate the cellpath routing component independently from the other components of the traffic flow estimation model. As the data for L.A. is based on simulated agents, we can compare to the original route of each user (see figure 9.2).

In order to run an automatic route validation for a number of randomly selected users, we need to define a measure that indicates whether a route is similar to the original one or not. Choosing such a similarity measure is not obvious as there are many possible ways to compare two routes. For example the work by Fillekes (2014) contains a comparison of 12 different of such measures. Some of these use *Hausdorff distance*, which is a measure to compare two geometries. The Hausdorff distance basically gets small if all points in one geometry are close to a point in the other geometry. However, for the network loading we don't only want to have a route that uses links that are close to the correct links but we want that as many links as possible match exactly with the correct links.

Table 9.2: Statistics about the used datasets for Los Angeles, USA

L.A. SSTEM data	
Source	UC Berkeley, AT&T
No. users	1160826
No. cells	805
Cellpath data	simulated home-work trips
Antenna positions	real, obfuscated
Avg. cell area*	$8.00 km^2$
Timeframe	Simulation based on data for April 2015
Time information	no time information

L.A. road network	
Source	Open-StreetMap
No. links	15068
No. links (simpl.)	13415
No. vertices	11405
Roads	all except residential

*after clustering using a minimum distance of 500m

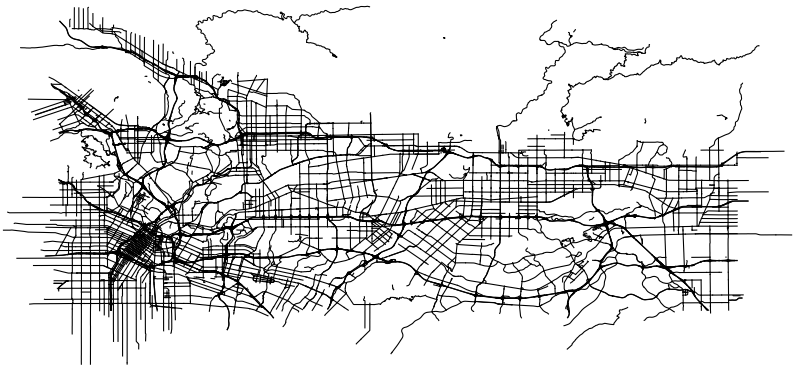


Figure 9.1: Road network used for Los Angeles based on OpenStreetMap data (excluding residential roads)

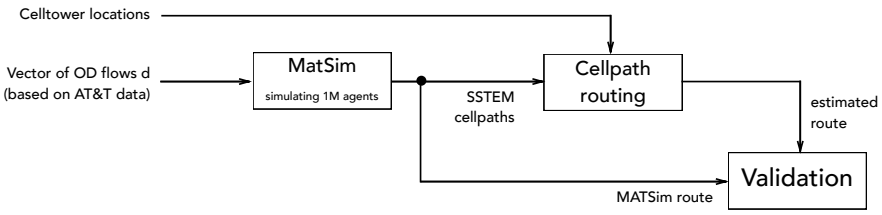


Figure 9.2: Route validation experiment setup

Therefore we use a similarity measure based on the intersections that are matching (definition 9.1). Basically it counts the number of common intersections in relation to the total number of distinct intersection in both routes. This method is also known as the *Jaccard index* of two sets (Pang-Ning et al., 2006). Note that instead of using the matching intersections we could in principle also compare the links directly. However, the MATSim road network uses different link ids and link geometries without intermediate points (only the intersections are kept), which makes it difficult to compare between the original road network used for estimating the routes and the MATSim links. For pratical reasons it was therefore easier to compare the intersections which are identified by their coordinates. It is very likely that a route that uses exactly the same intersections also use the same links, therefore this should not make a significant difference.

Definition 9.1: *Route similarity metric*. Let $r_a = (V_a, E_a)$ and $r_b = (V_b, E_b)$ be two routes defined by the vertices (intersections) (V_x) passed and the links (E_x) traversed along the route. Then we define their *route similarity* $s(a, b)$ as follows:

$$s(a,b) := \frac{|V_a \cap V_b|}{|V_a \cup V_b|} \tag{9.1}$$

Using this metric, we compare how good the three different cellpath routing algorithms recover the original route. To do so, a random sample of 1000 users is selected and then a route is estimated with all three algorithms for each user. For this experiment the default parameters shown in table 9.3 were used, which are used in all experiments for this case study if nothing else is stated. Note that lowering costs even outside of the visited cells was not used ($\beta = 1.0$, $d_E = 0$), as unlike in reality, always the correct cell was saved in the simulated data.

Calculating the similarity value for each of the 1000 users yields an average route similarity of 0.25 for the Shortest-path Routing algorithm which does not make use of the cellular network data (except for the start and end). As expected both Strict Voronoi Routing and Lazy Voronoi Routing achieve significantly higher similarities, which proofs that the use of the information given by the cellpath actually helps to recover the original route (see figure 9.3). Although the Lazy Voronoi Routing algorithm works best,

Table 9.3: Default parameters used in the experiments for Los Angeles, used if nothing else is stated

Parameter	Value
maximum antenna distance (clustering)	$d_a := 500m$
max. number of cellpaths used per OD pair	5
simplification tolerance (Lazy Voronoi)	$d_s := 0.03°$
Lazy Voronoi Routing cost parameters	$\alpha := 0.01$, $\beta := 1.0$, $d_E := 0$

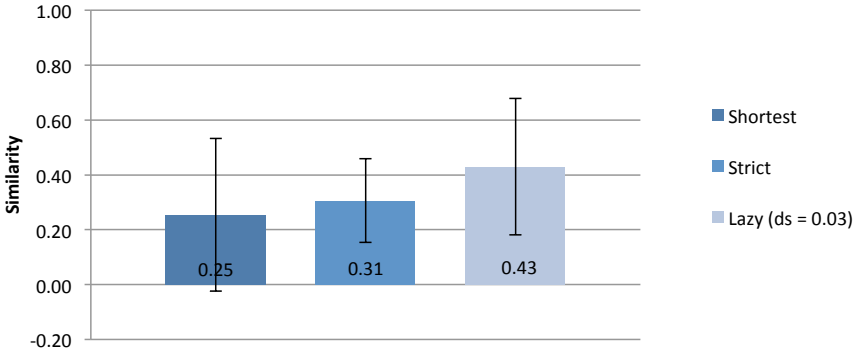


Figure 9.3: Results of the route validation: average similarity scores for the three different algorithms (based on 1000 randomly selected routes). Error bars show the standard deviation.

an average similarity of 0.43 is still far from the optimum. However, considering that cellpaths are often incomplete and given that cells can cover an area of many square kilometers it is not surprising that the similarity does not come close to 1.0.

Breaking down the comparison of the intersections shows that the reason for the low similarity in the case of Strict Voronoi Routing there are many points that only appear in the estimated route but not in the original route (see figure 9.4). This indicates that additional detours are added to the route and can be explained by a suboptimal placement of the waypoints.

An interesting aspect to further investigate is how much of the error is caused inside the start and end parts of the route. These errors are more or less unavoidable as the exact startpoint respectively endpoint inside the first or last is unknown. It can also happen that the first or last cell was not recorded, which is modelled using a certain probability that a cell is saved at every intersection in the simulated data. Reasons why that could happen even with real data include that the phone was turned off during a certain time or the battery was empty or the capturing of handover events is just not complete. In these cases it is impossible to recover the complete route.

9.3 Sensitivity analysis of the cellpath routing

The cellpath routing algorithms and especially the Lazy Voronoi Routing use a number of parameters that influence the quality of the recovered routes. A parameter that influences all algorithms is the minimum antenna distance used during the antenna clustering (see chapter 7.1) which is performed directly on the input data before any routes are calculated. The expectation is that be that Strict Voronoi Routing is largely uninfluenced by the antenna clustering as only the selection of the start- and endpoint could be slightly

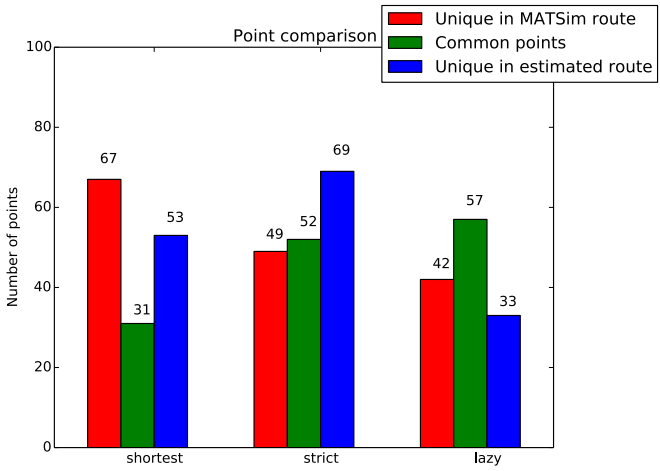


Figure 9.4: Comparison of the intersections on the route estimated by the three cellpath routing algorithms

affected. The route comparision with the original simulated routes confirms that (see figure 9.5). The algorithms that make use of the full cellpath information should benefit from the more precise location by a smaller cell size. While this holds for Lazy Voronoi Routing, this was not the case for Strict Voronoi Routing. While reducing the clustering distance from 1000m to 600m still improved the route quality, a further reduction seems to have no or even a slightly bad effect on the resulting routes. This might have to do with the increasing number of waypoints added when using smaller cells which also increases the risk that detours are added due to a bad waypoint placement.

All cells in a cellpath of the simulated data actually have been traversed by the user, as the simulation saves the Voronoi cell that the current intersection is inside. This behaviour is unlike real data, where based on the cell coverage and other factors a different cell might be saved. We therefore don't investigate the effect of changing the cost parameters for the Lazy Voronoi Routing. For the simulated data, setting the cost factor for links in the visited cells to a value like $\alpha = 0.01$, which is close (but higher than) 0, gave the best results. This is because it is guaranteed that one link inside each of the Voronoi cells in the cellpath was used on the route based on the design of the simulation. Using lower costs even around the visited cells did not improve the result. This is because this feature builds on the assumption that not always the correct cells are stored in the cellpath, which is true in reality, but not for the simulated data.

Instead we focus on the simplification tolerance parameter d_s of the Lazy Voronoi Routing. The lower the simplification tolerance distance is set, the

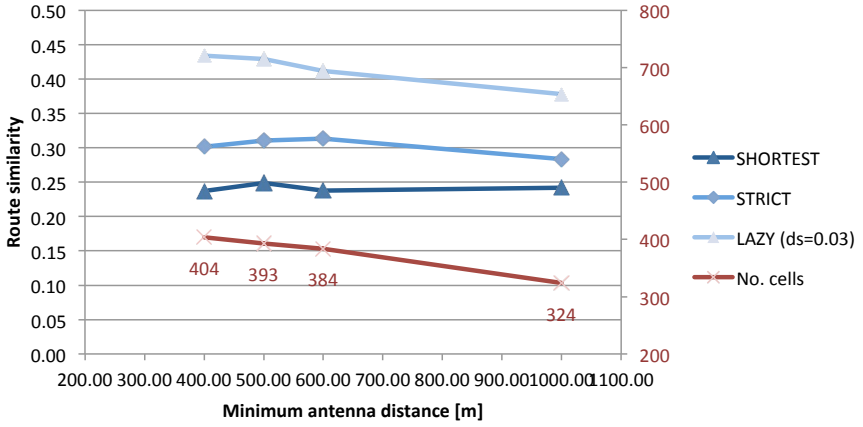


Figure 9.5: Impact of the minimum antenna distance threshold in antenna clustering on the route similarity for the different algorithms. No. cells is the number of cells after clustering.

more waypoints are generated to make the user follow the cellpath. A value of $d_s = 0.0$ would generate a waypoint in every single cell of the cellpath yielding routes similar to Strict Voronoi Routing. The higher the parameter is set, the fewer waypoints/segments are generated per cellpath (see figure 9.6).

As the simulated data always contains the right cells, it is not unexpected that the the similarity increases when more segments/waypoints are used to make the routes follow the cellpath more closely. The reason that Strict Voronoi Routing, even though it is using many waypoints performs worse than Lazy Voronoi Routing, can be explained by how waypoints are selected: In Strict Voronoi Routing three subsequent cells from the cellpath are used, while for Lazy Voronoi Routing three subsequent cells from the simplified cellpath are used. Selecting a waypoint if the three cells used for waypoint calculation are direct neighbors is difficult (which is often the case with the SSTEM data for Strict Voronoi Routing) as the optimal waypoint depends much on the exact link that the user is coming. If the cells used for the waypoint search are far away from each other, which is by nature the case in the simplified cellpath for Lazy Voronoi Routing, the exact start- and end positions used for the waypoint calculation inside the cells have less influence. Choosing the simplification tolerance is also a trade-off between the quality of the result and the computation time. The higher the distance is set, the fewer waypoints and route segments have to be calculated whereas more waypoints possible make the routes follow the cellpaths more precisely.

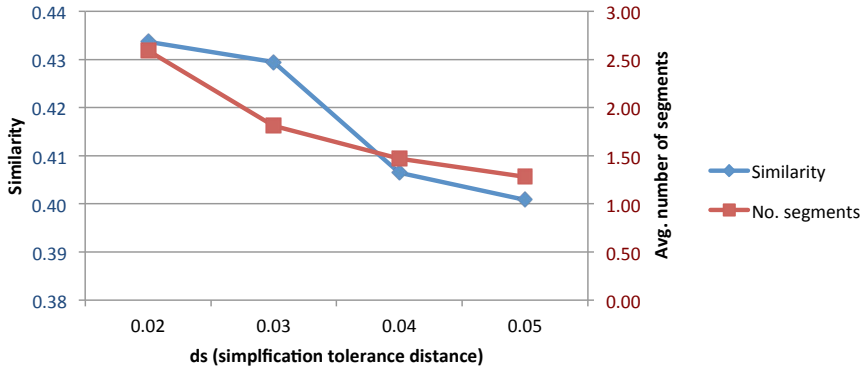


Figure 9.6: Impact of the simplification tolerance distance parameter when using the Lazy Voronoi Routing algorithm

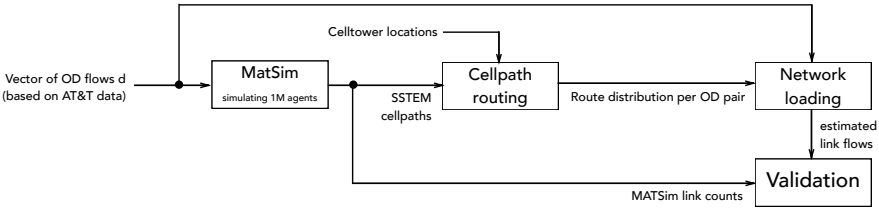


Figure 9.7: Network loading experiment based on SSTEM data

9.4 Validation of the network loading

After the route validation, the next experiment is to validate the link flows resulting from a network loading. For that we use the following setup (see figure 9.7): The inputs are a given OD-matrix (that was calcualated according to the procedure described in chapter 6.2 by researchers at UC Berkeley) and the SSTEM cellpaths (see chapter 9.1). For the cellpath routing, the Lazy Voronoi Routing algorithm is used with the default parameters listed in chapter 9.2). To validate the network loading result, we compare the estimated link flows with link counts that were obtained by aggregating the routes of all simulated agents in MATSim.

As the the simulated data already only contains movements and the OD-matrix is given, only the cellpath routing and network loading components of the whole method are used and validated in this experiment. Due to the fact that the OD-matrix was given for pairs of TAZs, while the model expects it to be between pairs of Voronoi cells, it was necessary to convert the matrix. This was done as follows: Given a pair of origin and destination TAZs all Voronoi cells that intersect with the origin respectively destination TAZ are queried. Then the travel demand from the TAZ based OD-matrix is assigned to the possible combinations of cell OD-pairs based on the share

of the area that the cells cover of the original TAZ areas. To verify that no major flows were lost in this process, the total demand in the original TAZ based matrix can be compared to the calculated cell based matrix. For the L.A. dataset it turned out that a minor flow of less than 2% was lost due to some TAZs outside of the border of the considered area having flows that did not intersect with any cell as they where outside of the considered area. Considering the small volume affected by this, no more effort was put into a workaround fixing this problem.

Using these inputs the result of the network loading is shown in figure 9.8. Similar to the network loading for Dakar are the main freeways having the highest flows. Comparing these estimated flows with the ground truth flows from the simulation gives the absolute differences[1] shown in figure 9.9. The comparison shows quite significant differences for the main roads. Problems occur especially where there are two parallel routes that are very close together. There is a high chance that these routes generate te same cellpath which makes them indistinguishable for the cellpath routing algorithm.

However, it also has to be noted that the freeway with the biggest errors is a very special case. It is divided into a normal freeway and a separate expressway which is only allowed to be used for buses and cars with a certain number of passengers. Therefore the freeway is divided into 4 links (2 per direction) in the road network. Neither the MATSim simulation nor the cellpath routing takes really account to these special regulations. Both treat the links as normal links. As with every link, the simulation takes also the traffic that already is on the link into account, while the cellpath routing algorithms select the link based on the free-flow travel time. This results in a slightly different share of the flow assigned to either the expressway or the normal freeway. A workaround could be to merge the expressway and the normal links into one for the sake of comparison. However as the links have different entries and exits and span through the whole corridor, this would require a lot of manual work and was therefore not done for this thesis.

To validate the network loading the GEH statistics (see chapter 2.2.3) is more useful than comparing the absolute counts. A GEH value was calculated for each link based on the estimated flow and the MATSim link flow (see figure 9.10). Only about 16% of the links have an acceptable (< 10) and only 8% (< 5) have good GEH values, which is far from the 85% commonly accepted as good practice. This indicates that the network loading is not precise enough for real traffic planning. On the other hand, it is unclear how the routes generated by MATSim relate to real user's routes. Even if the routing algorithms used by MATSim are simulating the human behaviour realistically, they are using assumptions about the capacity of each link which might not be correct in every case. If users follow the shortest-path more in reality than in the simulation, the results might actually be better with real data than with the simulated data.

[1] Note that some links are missing in the comparison due to difficulties to reidentify the links in both the estimated network loading and the MATSim link counts.

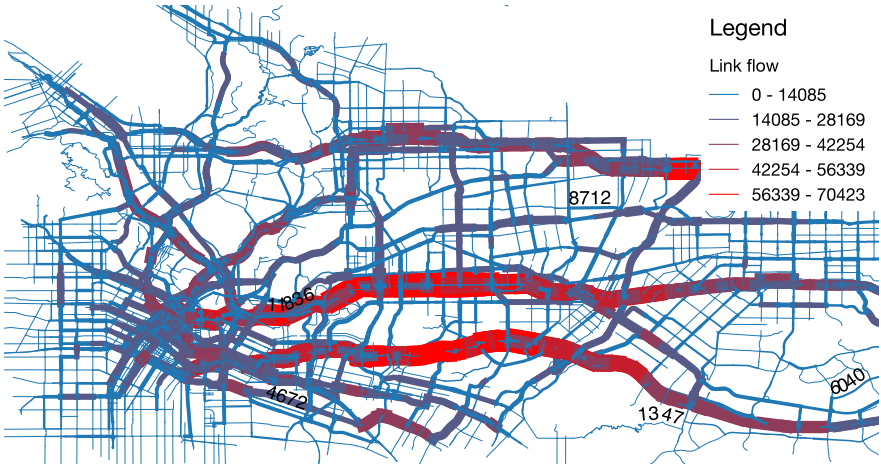


Figure 9.8: Network loading for the Los Angeles corridor using Lazy Voronoi Routing based on SSTEM data

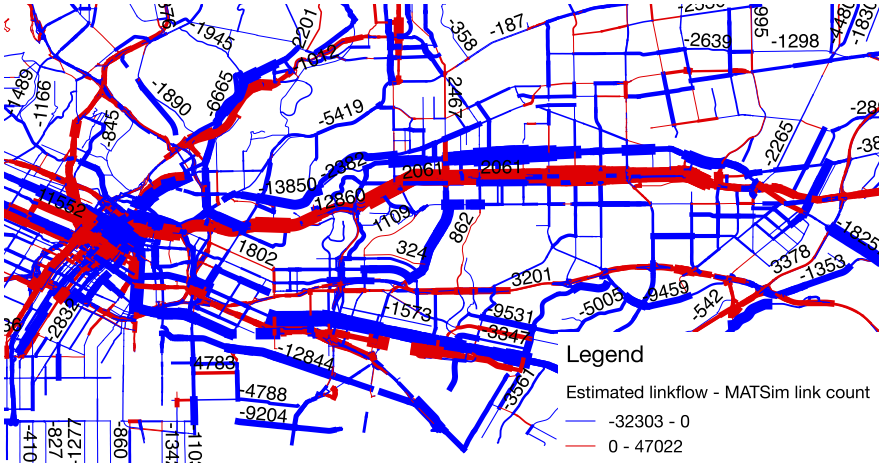


Figure 9.9: Absolute difference between the estimated linkflows using Lazy Voronoi Routing compared to MATSim link counts

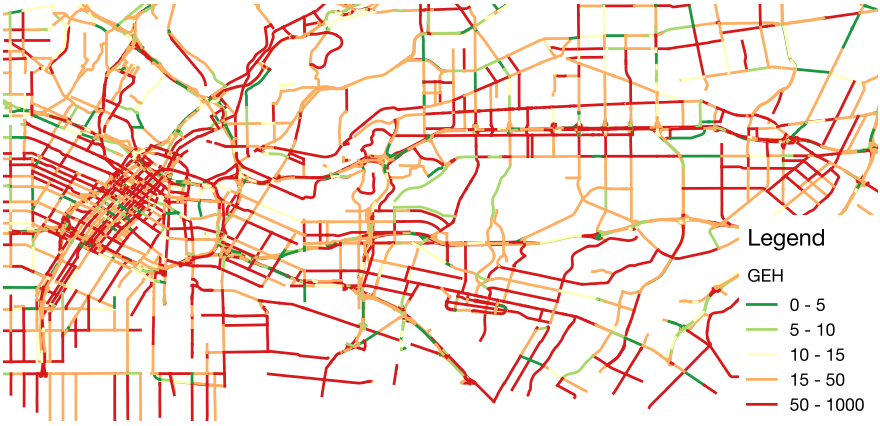
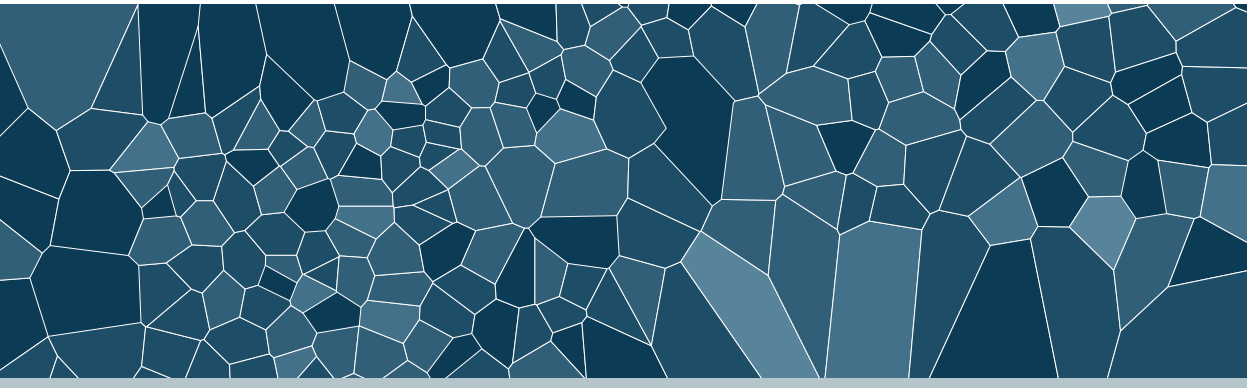


Figure 9.10: GEH values for all links in the Los Angeles corridor comparing the Lazy Voronoi Routing result and link counts from MATSim

Chapter 10

Discussion and future work

The two case studies presented in chapter 8 and 9 showed that cellular data helps to get a more detailed picture of traffic. However, there are still several weaknesses that prevent the presented method from being a precise traffic flow estimation method. This chapter will discuss these weaknesses and name possible future improvements that could help to make the method applicable for traffic planning in real-world scenarios.



10.1 Limitations of the travel demand estimation

The travel demand estimation using only CDR data as presented in chapter 6.1 has clear weaknesses. Upscaling the trips extracted from CDRs using a single population number for the whole country might give a first estimation, but is probably far from the real travel demand. For example could the sample consisting of users of one carrier be biased (Ranjan et al., 2012).

A characteristic of the travel demand estimation is that it only estimates the number of travellers. This is both an advantage, as for example link counts only measures only vehicles on the road which only covers certain modes. A link count does also not give any information about how many people are travelling (there could be only one person per vehicle or 6 per vehicle with a high share of public transport). From cellular network data we get an estimation of the total number of people travelling, which can be useful for integrated traffic models covering all modes of travel. On the other hand for some applications like investigating the occupancy of the links require an estimation of the number of vehicles. Using only cellular network data it is very difficult to estimate how travellers split up between different modes (although this could be worth investigating more in detail). To estimate the number of vehicles, it is also unclear how the number of travellers relates to the number of vehicles on the road (e.g. how many passengers each car has). To add even more complication, this number might even be different for each OD-pair. Estimating the number of vehicles from cellular network data only is difficult, if not impossible. More promising are methods that combine cellular network data and census data like the method presented in 6.2. However, this method only estimates commute travel between home and work and in contrast to the trip scaling method it does not generate a time-sliced matrix containing the demand for each hour of the day, leaving room for further improvements.

10.2 Possible improvements to the waypoint selection

The route validation (see chapter 9.2) showed waypoints especially for the Strict Voronoi Routing are sometimes adding extra detours to the route. There are at least two reasons why this can happen:

1. For the waypoint selection OSRM is used as the routing engine, but for the final cellpath routing, shortest paths are calculated in the database directly based on the road network imported through osm2po. As both tools might use slightly different costs for each roadtype, it is not guaranteed that a waypoint selecting with OSRM is also optimal if the costs calculated by osm2po would have been used.

2. The waypoint search is only based on three subsequent cells at each

time. This works fine if they are far away from each other (as in CDR data), but if they are direct neighbors, which is often the case in STEM data, sometimes suboptimal waypoints are selected. This is due to the fact that which waypoint is optimal in a cell depends on where the user exactly comes from in the previous cell and where he/she goes to in the next cell.

The reason for using OSRM for the waypoint selection was simply a performance reason. Due to the use of the a contraction hierarchy and a custom wrapper built for this thesis that accesses the direct OSRM C++ API allowed to calculate routes almost 100 times faster compared to calculating routes using a standard Dijkstra library on the database[1]. However, especially Lazy Voronoi Routing requires to modify the link cost before each route calculation, which is not possible with OSRM as this would require to rebuild the contraction hierarchy every single time loosing all of the performance gains. For this reason both tools had to be used in order to get a full network loading result in reasonable time. A solution could be to modify the cost profiles of both tools in order to make them to generate routes that are as similar as possible.

The second problem actually makes the Strict Voronoi Routing perform worse when the data is very detailed. It is difficult to select exactly one optimal waypoint for a cellpath part of 3 cells that are neighbors as there might be many ways to cross the cell depending on where the user comes from and goes to. One simple workaround could be to not use 3 subsequent cells from the cellpath to select a waypoint. It could already help to select the waypoint using a cell one or two further back respectively forth instead of the previous and next cell from the cellpath. For example if the cellpath is 1,2,3,4,5,6 select the waypoint in 4 using cell 2 and 6 instead of 3 and 5.

Another idea to improve the routing/network loading performance significantly is to use an abstract road network by advancing the road network simplification (see chapter 7.2). The road network simplification already calculates all shortest paths between border points of a cell anyway so the idea would be to save these routes directly storing only one link each shortest path between border points. Routing on this network that would only contain the border points, should be a lot faster than on the original one. The original links can be annotated to the virtual links making it possible to get the same routes as before. It would also be possible to annotate the virtual links with the cell that they belong to to allow to modify the costs for Lazy Voronoi Routing.

10.3 Multiple routes per cellpath

All presented cellpath routing algorithms only calculate exactly one route per cellpath. If there are two routes that pass through the same cells with

[1] using the pgrouting extension for Postgres/PostGIS

almost the same travel time, only the one with the lowest free flow travel time will get all travellers assigned. This could be one of the reasons why the estimated link flows are far from the real values (see chapter 9.4).

An algorithm that finds a number of different routes per cellpath could possibly improve the result. One approach could be to split the travel demand estimated for the cellpath among several routes based on the travel time of the routes. For example the route with the lowest travel time would still get the biggest part of the flow but the alternative routes would also get a portion based on how much higher the travel time is compared ot the fastest route. To calculate how to split the flow among the different routes a logit model (Wen et al., 2001) could be used. Alternatively a more sophisticated method could involve some kind of equilibrium model in order to model that travel times change on the links depending on how many travellers are using certain links.

10.4 Data fusion with other data sources

Because of the imprecise location information from cellular data, any method only based on this data will always have certain inaccuracies. This is especially a problem when there are multiple parallel roads with similar travel times in the same cell. A solution to this problem could be to complement the cellular network data with link count data from classical link count sensors. The method by Wu et al. (2015) is very promising, but has not yet been tested with real data. The authors propose a convex optimization model in order to fuse the two datasources. So far the model assumes that cellpaths are complete (meaning that all consecutive cells in the cellpath are direct neighbors) which is not the case with real data. An idea is to use the Lazy Voronoi Routing algorithm presented in chapter 7.4.3 in order to patch the raw cellpaths using the data fusion method. This would work by calculating a route based on the raw cellpath. Then a new patched cellpath would be created by compiling all cells that the estimated route passes through. While the Lazy Voronoi Routing algorithm might not always select the exact correct links as chapter 9.2 showed, routes often just use very close parallel links to the correct one. Therefore it is likely that if the links are not always matching, the algorithm would still recover the correct cells that the original route passed through.

10.5 Real time applications

An interesting area worth further investigation would be to modify the traffic flow estimation model for a more real-time scenario. While the model presented in this thesis is designed to analyze historic traffic for a typical day, estimating traffic flows in real time is interesting for example for traffic control. To be able to run the model continuously in real-time, the design

needs to be changed in order to process a continous stream of new data. As the amount of data available in real-time would be much smaller than the historic aggregated data, it would probably make sense to combine historic data and real-time data in order to get a reliable continous flow estimation. An idea is to use a Kalman filter (Lint et al., 2012) in order to fuse the real-time data and the historic data.

Chapter 11

Conclusions

The model presented in this thesis shows how cellular data could be used in order to estimate the traffic flows in a transportation network. The process can be divided into 4 components (trip extraction, travel demand estimation, cellpath routing and network loading). The extraction of trips from the raw cellular network data requires a method that matches the type of the data used. While for sparse CDR data, trips must be extracted from very few available positions, other data types like STEM data allow to get a more continous indication of the movement of a user.

Estimating travel demand only from cellular network data has the risk of bias due to how the customer base of the operator is composed. Home- and work-locations for the available user base can however be approximated easily and relatively accurate. For a reliable travel demand calculation it would be best to combine cellular network data with other statistics like official census data and link counts, if available. It is also important to keep in mind that using mobile phones as traffic probes implies that the travel demand is estimated in terms of the number of people travelling. For many traffic planning applications it is however necessary to for example estimate the travel demand in terms of vehicles. This requires more information about the modal split (could possibly be obtained from cellular data at least approximately) and the average number of passengers per vehicle (can most likeley not directly be obtained from cellular network data), which makes more data sources necessary.

Comparing single routes estimated from cellular network data to simulated routes showed that especially the Lazy Voronoi cellpath routing algorithm helps to recover the links that a user used (route similarity > 0.4 instead of only 0.25 when just using the direct shortest path between start and destination). Through the use of waypoints only for few cells in the cellpath and using modified link costs it is not only giving a better route estimation, but also much faster than the Strict Voronoi Routing algorithms that places a waypoint in every cell of the cellpath. On the other hand

the validation of the network loading indicates that this value is still not sufficient to get a reliable network loading using this method only based on cellular network data. The cell areas are too big to exactly recover the correct links used, especially when there are many parallel roads within the same Voronoi cell with. It is also unclear how realistic the simulated routes are. Real data might give a better result if people follow the shortest-path more often as in the simulation. An improvement could be to calculate multiple routes per cellpath and split the flow among the routes depending on their travel time or even use a model that takes account to the travel time increase perceived on links with high flows.

Future interesting research areas are data-fusing cellular network data with link count data, as this would enable a much more precise network loading. Using cellular network data for traffic analysis is definitely useful not only to estimate link flows. It could also be used in order to get information on the route flows, which is something that typically only can be obtained relatively unreliable from surveys. Route flows are for example interesting for public transport planning as it could give insights on between which bus-lines users change typically. If there are many people changing at some station and all continuing with a different line, the line network could be improved by adding a direct line for example. Using cellular network data in real-time could allow to react quicker to incidents or traffic jams. The natural characteristic of mobile phone data is that the movement of people is revealed (assuming that most people carry exactly one phone). This is contrary to for example link count sensors where instead the movement of vehicles is measured. This opens up for a whole new category of integrated mobility analysis for all travel modes.

References

Aggarwal, Alok, Leonidas J Guibas, James Saxe, and Peter W Shor (1989). "A linear-time algorithm for computing the Voronoi diagram of a convex polygon". In: *Discrete & Computational Geometry* 4.1, pp. 591–604.

Andrienko, Gennady, Aris Gkoulalas-Divanis, Marco Gruteser, Christine Kopp, Thomas Liebig, and Klaus Rechert (2013). "Report from Dagstuhl: the liberation of mobile location data and its implications for privacy research". In: *ACM SIGMOBILE Mobile Computing and Communications Review* 17.2, pp. 7–18.

Baert, A.-E. and D. Seme (2004). "Voronoi mobile cellular networks: topological properties". In: *Parallel and Distributed Computing, 2004. Third International Symposium on/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004. Third International Workshop on*, pp. 29–35. DOI: 10.1109/ISPDC.2004.58.

Balakrishna, Ramachandran, Constantinos Antoniou, Moshe Ben-Akiva, Haris Koutsopoulos, and Yang Wen (2015). "Calibration of microscopic traffic simulation models: Methods and application". In: *Transportation Research Record: Journal of the Transportation Research Board*.

Balmer, Michael, Marcel Rieser, Konrad Meister, David Charypar, Nicolas Lefebvre, Kai Nagel, and K Axhausen (2009). "MATSim-T: Architecture and simulation times". In: *Multi-agent systems for traffic and transportation engineering*, pp. 57–78.

Becker, Richard A, Ramon Caceres, Karrie Hanson, Ji Meng Loh, Simon Urbanek, Alexander Varshavsky, and Chris Volinsky (2011). "A tale of one city: Using cellular network data for urban planning". In: *IEEE Pervasive Computing* 10.4, pp. 0018–26.

Caceres, N., J.P. Wideberg, and F.G. Benitez (2007). "Deriving origin-destination data from a mobile phone network." In: *IET Intelligent Transport Systems* 1.1, pp. 15–26.

Calabrese, Francesco, Giusy Di Lorenzo, Liang Liu, and Carlo Ratti (2011). "Estimating Origin-Destination Flows Using Mobile Phone Location Data." In: *IEEE Pervasive Computing* 10.4, p. 36. ISSN: 15361268.

Cormen, Thomas H (2009). *Introduction to algorithms*. MIT press.

Dean, Jeffrey and Sanjay Ghemawat (2008). "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1, pp. 107–113.

Deming, W Edwards and Frederick F Stephan (1940). "On a least squares adjustment of a sampled frequency table when the expected marginal totals are known". In: *The Annals of Mathematical Statistics* 11.4, pp. 427–444.

Douglas, David H and Thomas K Peucker (1973). "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2, pp. 112–122.

Estivill-Castro, Vladimir (2002). "Why so many clustering algorithms: a position paper". In: *ACM SIGKDD explorations newsletter* 4.1, pp. 65–75.

Evans, Suzanne P. (1973). "A relationship between the gravity model for trip distribution and the transportation problem in linear programming". In: *Transportation Research* 7.1, pp. 39–61. ISSN: 0041-1647. DOI: 10.1016/0041-1647(73)90005-1.

Fienberg, Stephen E (1970). "An iterative procedure for estimation in contingency tables". In: *The Annals of Mathematical Statistics*, pp. 907–917.

Fillekes, Michelle (2014). "Reconstructing Trajectories from Sparse Call Detail Records". PhD thesis. University of Tartu.

Fotheringham, Stewart and Peter Rogerson (2013). *Spatial analysis and GIS*. CRC Press.

Goldberg, Andrew V and Chris Harrelson (2005). "Computing the shortest path: A search meets graph theory". In: *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, pp. 156–165.

Gundlegård, David and Johan M Karlsson (2009). "Route classification in travel time estimation based on cellular network signaling". In: *Intelligent Transportation Systems, 2009. ITSC'09. 12th International IEEE Conference on*. IEEE, pp. 1–6.

Gundlegård, David, Clas Rydergren, Jaume Barcelo, Nima Dokoohaki, Olof Görnerup, and Andrea Hess (2015). "Travel Demand Analysis with Differentially Private Releases". In: *NetMob*.

Hoteit, Sahar, Stefano Secci, Stanislav Sobolevsky, Carlo Ratti, and Guy Pujolle (2014). "Estimating human trajectories and hotspots through mobile phone data". In: *Computer Networks* 64, pp. 296–307. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2014.02.011.

Isaacman, Sibren, Richard Becker, Ramón Cáceres, Stephen Kobourov, Margaret Martonosi, James Rowland, and Alexander Varshavsky (2011). "Identifying Important Places in People's Lives from Cellular Network Data". English. In: *Pervasive Computing*. Ed. by Kent Lyons, Jeffrey Hightower, and ElaineM. Huang. Vol. 6696. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 133–151. isbn: 978-3-642-21725-8. doi: 10.1007/978-3-642-21726-5_9.

Leontiadis, Ilias, Antonio Lima, Haewoon Kwak, Rade Stanojevic, David Wetherall, and Konstantina Papagiannaki (2014). "From Cells to Streets: Estimating Mobile Paths with Cellular-Side Data". In: *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, pp. 121–132.

Lint, Hans van and Tamara Djukic (2012). "Applications of Kalman Filtering in Traffic Management and Control". In: *New Directions in Informatics, Optimization, Logistics, and Production*. Vol. 9. INFORMS, Hanover, MD, pp. 59–91.

Luxen, Dennis and Christian Vetter (2011). "Real-time routing with OpenStreetMap data". In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '11. Chicago, Illinois: ACM, pp. 513–516. isbn: 978-1-4503-1031-4. doi: 10.1145/2093973.2094062.

McNally, Michael G (2008). "The four step model". In: *Center for Activity Systems Analysis*.

Ming-Heng, Wang, S.D. Schrock, N.V. Broek, and T. Mulinazzi (2013). "Estimating dynamic origin-destination data and travel demand using cell phone network data." In: *International Journal of Intelligent Transportation Systems Research* 11.2, pp. 76–86.

Montjoye, Yves-Alexandre de, Zbigniew Smoreda, Romain Trinquart, Cezary Ziemlicki, and Vincent D. Blondel (2014). "D4D-Senegal: The Second Mobile Phone Data for Development Challenge". In: *CoRR* abs/1407.4885. url: http://arxiv.org/abs/1407.4885.

Murtagh, Fionn and Pedro Contreras (2012). "Algorithms for hierarchical clustering: an overview". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.1, pp. 86–97. issn: 1942-4795. doi: 10.1002/widm.53.

Pang-Ning, Tan, Michael Steinbach, Vipin Kumar, et al. (2006). "Introduction to data mining". In: *Library of Congress*, p. 74.

Patriksson, Michael (2015). *The Traffic Assignment Problem: Models and Methods*. Courier Dover Publications.

Rahnema, Moe (1993). "Overview of the GSM system and protocol architecture". In: *Communications Magazine, IEEE* 31.4, pp. 92–100.

Ramer, Urs (1972). "An iterative procedure for the polygonal approximation of plane curves". In: *Computer Graphics and Image Processing* 1.3, pp. 244–256. ISSN: 0146-664X. DOI: 10.1016/S0146-664X(72)80017-0.

Ranjan, Gyan, Hui Zang, Zhi-Li Zhang, and Jean Bolot (2012). "Are Call Detail Records Biased for Sampling Human Mobility?" In: *SIGMOBILE Mob. Comput. Commun. Rev.* 16.3, pp. 33–44. ISSN: 1559-1662. DOI: 10.1145/2412096.2412101.

Rose, Geoff (2006). "Mobile Phones as Traffic Probes: Practices, Prospects and Issues." In: *Transport Reviews* 26.3, pp. 275–291. ISSN: 01441647.

Steenbruggen, John, Maria Borzacchiello, Peter Nijkamp, and Henk Scholten (2013). "Mobile phone data from GSM networks for traffic parameter and urban spatial pattern assessment: a review of applications and opportunities." In: *GeoJournal* 78.2, p. 223. ISSN: 03432521.

Tekinay, S. and B. Jabbari (1991). "Handover and channel assignment in mobile cellular networks". In: *Communications Magazine, IEEE* 29.11, pp. 42–46. ISSN: 0163-6804. DOI: 10.1109/35.109664.

Tettamanti, Tamás, Hunor Demeter, and István Varga (2012). "Route choice estimation based on cellular signaling data". In: *Acta Polytechnica Hungarica* 9.4, pp. 207–220.

Walke, Bernhard (2002). *Mobile Radio Networks : Networking, Protocols, and Traffic Performance.* John Wiley and Sons, Inc. ISBN: 9780471499022.

Wen, Chieh-Hua and Frank S Koppelman (2001). "The generalized nested logit model". In: *Transportation Research Part B: Methodological* 35.7, pp. 627–641. ISSN: 0191-2615. DOI: 10.1016/S0191-2615(00)00045-X.

White, J. and I. Wells (2002). "Extracting origin destination information from mobile phone data." In: *Road Transport Information and Control, 2002. Eleventh International Conference on (Conf. Publ. No. 486).*

Wu, Cathy, Jérôme Thai, Steve Yadlowsky, Alexei Pozdnoukhov, and Alexandre Bayen (2015). "Cellpath: Fusion of cellular and traffic sensor data for route flow estimation via convex optimization". In: *Transportation Research Part C: Emerging Technologies.*

Zang, Hui and Jean Bolot (2011). "Anonymization of location data does not work: A large-scale measurement study". In: *Proceedings of the 17th annual international conference on Mobile computing and networking.* ACM, pp. 145–156.

Glossary

3G third generation of mobile telecommunications technology 71

4G fourth generation of mobile telecommunications technology 71

4-stage model A common traffic assignment model (see chapter 4.1) 16, 29, 30, 31

C++ Low level object-oriented programming language 34

cellpath A sequence of cells or their ids (see definition 2.1) 15, 26, 31, 32, 33, 46, 51, 55, 56, 57, 74, 76, 77, 83, 85, 87

connected component A subgraph of a graph in which a path exists between all vertices 47

free flow speed the travel speed on a certain link that is reached when there are no other vehicles on the road 16, 50

GEH Metric to compare linkflows named after Geoffrey E. Havers 17, 18, 79

Hausdorff distance Distance measure between two geometries that gets small when all points in one geometry are close to a point in the other 72

Jaccard index Set comparison metric 72

Lazy Voronoi Routing Cellpath routing algorithm combining strict Voronoi routing and modified cost routing 8, 27, 55, 56, 57, 67, 74, 75, 76, 77, 84, 85

link flow The number of travellers or vehicles using a specific link in the road network 10, 11, 15, 17, 31, 33, 57, 66, 77, 84, 88

linkpath A sequence of of connected links (or their ids) in a transportation network 32, 33

MATSim Multi-Agent Transport Simulation (see matsim.org) 71, 77, 79

NumPy Mathematical library for Python (see numpy.org) 33

OD-matrix A matrix containing a traffic demand between pairs of origin and destination zones 11, 15, 16, 17, 31, 32, 33, 41, 42, 64, 71, 77, 78

Open Source Software whose source code is publically available 19

osm2po Tool that converts OpenStreetMap data into a routable network for PostGIS (see osm2po.de) 62, 83

OSRM Open Source Routing Machine, see project-osrm.org 19, 34, 51, 83, 84

Postgis Extension to Postgres providing GIS operations 33

Postgres SQL-Database 33, 34

Python Programming language 33, 34

QGIS GIS application (see qgis.org) 34

SciPy Scientific library for Python (see scipy.org) 33

Shortest-path Routing Cellpath routing algorithm that only calculates the shortest path between the first and last cell in the cellpath 52, 53, 57, 67, 74

STEM cellular network data generated from handovers 13, 15, 32, 37, 39, 44, 71, 83, 87

Strict Voronoi Routing Cellpath routing algorithm that forces the route to enter every cell in the cellpath 53, 55, 56, 57, 67, 74, 75, 76, 77, 83, 84, 87

volume-delay function A function that describes the travel time on a link using it's free-flow travel time and the flow on the link 16

Voronoi cell A polygon describing the area where no other site (here antenna) is closer than the one inside the cell (see chapter 2.3.4) 20, 26, 47, 48, 49, 50, 51, 53, 54, 55, 56, 61

Voronoi diagram Partition of the plane into regions based on a set of points and their distance (see chapter 2.3.4) 14, 20, 24, 27, 61

Chapter cover pages:

2 Voronoi cells for a part of Dakar, points show the antenna locations
3 Number of homelocations in each Voronoi cell for a part of Dakar, lighter means more homelocations
4 Estimated flows for a part of Los Angeles using Lazy Voronoi routing
5 Sample cellpath from the Los Angeles SSTEM dataset visualized by line connecting the antennas in the cellpath
6 Travel demand from one cell to all other cells in Los Angeles (map flipped horizontally), a thicker line means more travel demand
7 Estimated flows for a part of Los Angeles using Lazy Voronoi routing
8 Road network for Dakar (only half of the residential links shown)
9 Road network for Los Angeles (no residential links)
10 Number of people estimated in each Voronoi cell for Dakar between 8:00-8:59 from CDRs, lighter means more people

På svenska

In English