Utilizing Call Detail Records for Travel Mode Discovery in Urban Areas

Martijn Roo

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Berlin 22.03.2017

Thesis supervisor:

Assoc. Prof. Keijo Heljanko

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Martijn Roo

Title: Utilizing Call Detail Records for Travel Mode Discovery in Urban Areas

Date: 22.03.2017 Language: English Number of pages: 4+79

Department of Computer Science

Professorship: Computer Science

Supervisor and advisor: Assoc. Prof. Keijo Heljanko

Mobile network operators often bill their customers based on their network usage. For this purpose, operators collect information about billable events, such as calls, text messages, and data usage. In recent years, operators have realized that they can monetize these billing records by selling insights extracted from them. In this thesis, a multi-stage data analysis algorithm is presented that uses these billing records for travel mode classification. This algorithm identifies whether a mobile phone user has traveled using a public transportation bus or using another transportation mode.

The billing records collected by a network operator contain the time at which a billable event happened, as well as the network cell from which the event originated. The coverage area of each network cell is known to the operator. Therefore, the billing records of a mobile phone user give an overview of that user's approximate location at different times. This data can be used to discover the sequence of network cells that the user has traveled through during a trip.

Travel mode classification algorithms in literature analyze long-distance or medium-distance trips. The data analysis algorithm presented in this thesis is novel for analyzing and classifying short-distance, intra-city trips. To classify mobility traces, it uses publicly available bus timetable data and road network infrastructure data. The accuracy of the classification algorithm is evaluated using a two-fold cross-validation analysis.

Keywords: Call Detail Record, Distributed, Travel Mode Classification

Contents

1 Introduction

Mobile network operators have always collected data on the usage of their network in order to bill their customers. In the last decade, researchers have shown that various insights about human mobility can be extracted from these billing records. Initiatives such as the Orange D4D Challenge[1] have seen researchers use billing data to answer societal or technical challenges. In recent years, operators have realized that insights gained from their billing data could form an additional source of income. Considering these developments, this thesis presents a multi-stage data analysis algorithm that uses billing data to discover the travel mode of mobile subscribers.

Network operators collect events in their network related to billable actions, such as calls, text messages, and data usage. Call and text events are registered in Call Detail Records (CDRs), whereas data usage events are recorded in Data Detail Records (DDRs). CDRs and DDRs record the device that generated the billable event, the time at which it was generated, and the network cell from which the event originated. The operator has data on how far away a subscriber is from a cell tower, which can be used to localize a user on a sub-cell level. However, this data remains in the access network and is not collected in CDRs or DDRs. Therefore, a CDR or DDR provides a mobile subscriber's location on the level of a network cell.

Mobile phone penetration is very high in many countries. As a result, human mobility insights based on CDR or DDR datasets are likely to be generalizable to the population as a whole. Another benefit of using billing records for research, is that the network operator collects this data as part of their core business operation. This makes the data reliable and possibly more affordable to acquire, since no additional network probes have to be installed. CDRs are generally less frequently generated than DDRs. Most mobile subscribers text or call less on their phones than that they use the Internet. Furthermore, smartphones often use a data connection in the background without a user's input, whereas text messages and calls require user interaction. As a result, CDR datasets often provide a less accurate overview of a subscriber's mobility than DDR datasets. A subscriber's location is not known in between records. The longer the period between two records, the more uncertainty there is about the subscriber's mobility. For instance, a subscriber could be traveling slowly from one point to another, or they could be stationary for a period of time and then travel very quickly from the first point to the second. Without intermediate records, these scenarios are indistinguishable. Having more frequent records reduces the uncertainty. In the field of mobility analysis, CDR datasets have been more common than DDR datasets. As a result, many researchers have opted for analyzing mobility patterns over medium to long distances instead of over short distances. Few CDRs are usually generated during short-distance trips, which makes it difficult to extract definite insights.

In this thesis, an extension on existing research is provided by a mobility analysis on short-distance, intra-city trips. A multi-staged data analysis algorithm is presented for discovering the travel mode of mobile subscribers in an urban area. The intra-city scale offers new use cases. For instance, a local business proprietor can discover how

[1] http://www.d4d.orange.com/en/

many people travel to their shop using different modes of transportation. They could then adjust their sales tactics to their customer base or improve the infrastructure used by customers to reach the shop. Similarly, a public transportation company could use the classification algorithm to discover how many people travel on particular buses. They could then adjust their timetable or capacity accordingly. In Berlin, there is no check-in/check-out system for public transportation. Consequently, the public transportation company, BVG, has to hire people to count travelers if they require detailed information about bus occupancy. This is an expensive process that could be replaced by data analysis, which would be less expensive and could be run more frequently.

The CDRs or DDRs of a subscriber can be used to discover the sequence of network cells that they have traveled through during a trip. Such a sequence of network cells is called a mobility trace. Extracting a mobility trace from billing data involves detecting when a subscriber is stationary in order to identify when a trip starts and ends. Extracting mobility traces is a separate area of research and outside of the scope of this thesis. The developed data analysis algorithm classifies mobility traces into one of two categories. It classifies traces either as traveled by public transportation bus, or as traveled by another mode of transportation. Although this is a simplification, it allows the discovery of valuable insights for the use cases previously mentioned, i.e., a local business proprietor and a public transportation company. One network operator can provide service to many millions of subscribers, all making multiple trips per day. To handle this data quantity, the classification algorithm is developed as a distributed program. The distributed data analysis algorithm uses proprietary network infrastructure data from a network operator, publicly available bus timetable data, and publicly available road network infrastructure data.

An evaluation of the algorithm's classification accuracy is presented at the end of this thesis. Specifically for the purpose of this evaluation, a ground truth dataset was collected in Berlin, Germany. It includes trips that were made by car and trips that were made by public transportation bus. For all trips, the transportation mode is recorded, so that the actual (ground truth) transportation mode can be compared to the results of the algorithm.

This thesis contains seven chapters. Chapter 2 covers related work in the field of human mobility analysis using mobile network data. The first two sections detail how mobile network data is captured and how mobility traces can be extracted from this data. The last four sections describe different areas of mobility research using mobile network data. Chapter 3 details the concept and design of the distributed classification algorithm presented in this work. In the chapter, an overview is given of the requirements for the algorithm and an overview of the algorithm steps is given. In Chapter 4, the implementation of the algorithm is covered. This includes an overview of the software architecture and detailed descriptions of how the steps of the algorithm are implemented. Chapter 5 presents the evaluation results. First, the setup of the evaluation is explained. Then, the data collected for evaluation is characterized. Lastly, a description of the two-fold cross-validation analysis and its results is presented. Finally, Chapter 6 presents the conclusions of this thesis.

2 Related Work

This chapter identifies related work in the field of mobility analysis using mobile traffic data. It starts with an introduction into the different types of mobile networks in Section 2.1, specifically how the cell sizes differ for each network type and which signaling events occur within those networks. However, the core of the related work focuses on mobility analysis using the data from a cellular network, for brevity hereafter referred to as *cellular data*.

Related work in mobility analysis is divided into categories. First, work on extracting trajectories from cellular data is presented in Section 2.2, followed by research on identifying laws in human mobility and predicting future mobility given past data in Section 2.3. Then, Section 2.4 presents research on using cellular data for discovering transportation system state and usage. Section 2.5 briefly addresses the accuracy and reliability of mobility analyses based on cellular data. Finally, Section 2.6 summarizes the presented research and places the contribution of this work within the field of mobility analyses using cellular data.

2.1 Cellular data collection

Mobile providers collect information on the usage of their networks for billing and maintenance purposes. In this section, the architecture of contemporary cellular networks is described; followed by the signaling events that occur within a cellular network; what probes can be used to gather those events; and how this data can be anonymized so that it can be analyzed without jeopardizing user privacy.

2.1.1 Cellular network architecture

Different generations of mobile networks are currently in widespread use, such as 2G (GSM, GPRS, and EDGE), 3G (UMTS and HSPA), and LTE (4G). Often multiple network types are operated simultaneously in the same geographical area by a single provider. The core network structure of these standards is similar, although different terminology is used for elements executing similar functions in the different standards [43]. A generalized architecture is presented below using the terminology for 2G networks, see Figure 1. However, terms used in 3rd and 4th generation networks are mentioned and significant differences between the different network generations are explained.

A mobile provider's network is divided into cells. A mobile subscriber connects to the Base Transceiver Station (BTS), known as NodeB in 3G networks, that is responsible for the cell they are in. One or multiple BTSs are managed by a Base Station Controller (BSC), known as a Radio Network Controller (RNC) in 3G. For LTE, the BTS and BSC functions are combined in the eNodeB component. The combination of a BSC with its connected BTSs is called a Base Station Subsystem (BSS), or Radio Network Subsystem (RNS) in 3G.

All BSSs together form the Radio Access Network (RAN), which is the part of the network responsible for providing wireless access to mobile subscribers. The RAN is

connected to the Core Network (CN), which is the part responsible for transferring calls and text messages between different BSSs as well as for connecting to external networks, such as the Internet.

The CN looks differently for 2G and 3G networks than for 4G. In 2G and 3G architectures, voice and text services are provided via the Circuit Switched (CS) core, whereas data services are provided via the Packet Switched (PS) core. The CS core consists of a Mobile Switching Center (MSC) for voice and text services within the mobile network and a Gateway Mobile Switching Center (GMSC) for voice and text services with other operators' networks. The PS core consists of Serving Gateway Support Nodes (SGSN) and Gateway GPRS Support Nodes (GGSN). These take care of packet-switched data transfers, where the former interfaces towards subscriber devices and the latter towards the Internet. The 4G architecture has no separate core for data and another for text and voice services, instead it has one Evolved Packet Core (EPC) that handles both functions. The EPC consists a Mobility Management Entity (MME) that handles the mobile subscriber, a Serving Gateway (SGW) that forwards data packets from the device, and a Packet Data Network Gateway (PGW) that provides access to other networks, such as the Internet.

The mobile network architecture needs to maintain an overview of a user's network usage in order to bill them properly. A few network components enable this. The Charger Trigger Function (CTF) observes network resource usage and creates charging events based on that. The CTF sends these events to the Charging Data function, which creates a Call Detail Record (CDR) containing information such as the number of the caller and of the called party, the cell identifier of the

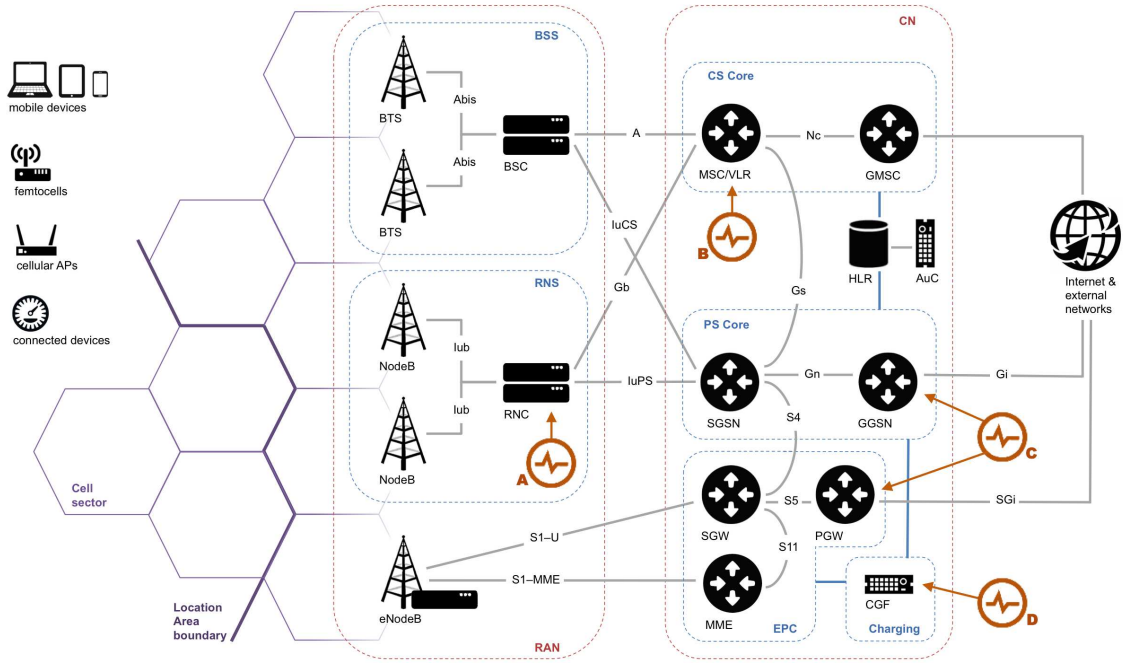


Figure 1: Cellular network architecture with mobile traffic probes [54]

originating cell, and the call duration [59]. Finally, the Charging Gateway Function (CGF) validates, reformats, and stores the CDR before forwarding it to the billing domain. Data Detail Records (DDRs) fulfill the same purpose as CDRs, but then for data usage events instead of call and text message events.

Cell sizes A cell is the smallest area in which a user can possibly be located based on a single signaling event in a mobile network. The range in cell sizes is therefore an important property to consider when analyzing users' location or mobility patterns. Providers use different types of cells depending on the required capacity and the number of range-limiting factors in the environment, such as tall buildings. Moreover, different network generations have different cell sizes as well due to changes in used technology.

Macro-cells are positioned higher on rooftops or cell towers and they usually cover tens of kilometers in radius. In denser areas, such as urban regions, macro-cells usually fail to provide enough capacity for all mobile subscribers. Therefore, coverage is supplemented with micro-cells that are elevated only to the level of street lamps and generally cover a few hundred meter in radius. In especially dense areas, such as airports, train stations, or offices, pico-cells can be employed. These cover a very small area up to tens of meters, comparable to a wireless access point. Lastly, a femto-cell has a similar range compared to a pico-cell, but is designed to have self-managing capabilities, which a pico-cell often is not. It can often be installed by users themselves to provide better indoor coverage at their offices or residencies.

The frequency bands and technology used for the different generations of mobile networks influences the theoretical maximum attainable range for a single cell [71]. However, the frequency bands used for a single network generation differ per country and per provider, making other developments more consequential for the change in average cell size over time. Third and especially fourth generation networks allow more hybrid networks where smaller (micro and pico) cells are used in conjunction with more traditional larger (macro) cells. This allows for a higher, more appropriate capacity in denser areas. This development results in cell sizes becoming smaller on

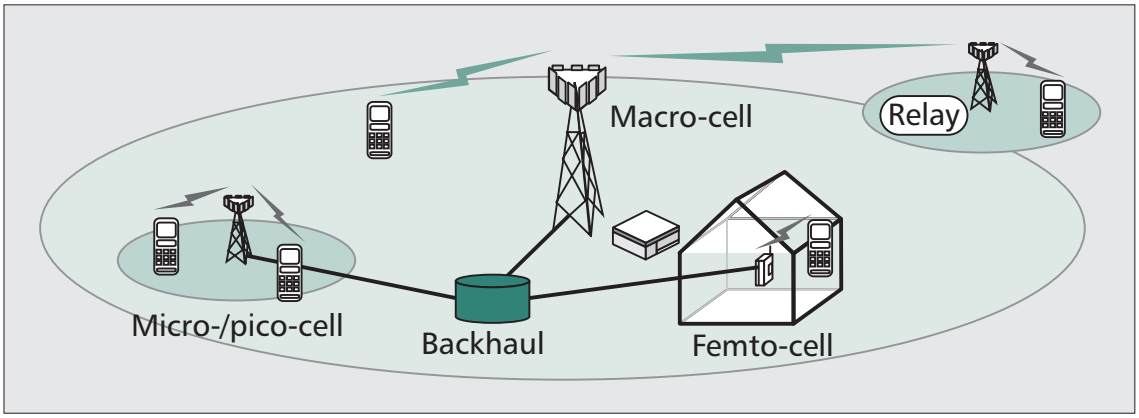


Figure 2: Different cell types [16]

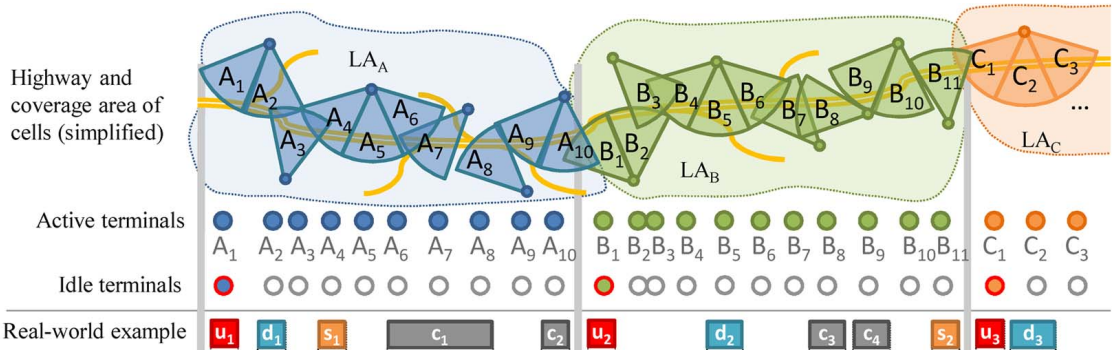


Figure 3: Signaling events on a highway section with c: phone calls, s: SMS messages, d: data connections, u: location updates [43]

average [39, 18].

2.1.2 Signaling events

Different events related to user usage and network status occur within a mobile network. Depending on the network probe being used, a subset of those events can be captured (see Section 2.1.3). Typical signaling events originating from a mobile device in a cellular network are text message events, call events, data usage events, cell handovers, and location updates. Text message, call, and data events occur when a mobile subscriber uses those respective services. A cell handover (HO) occurs when a user changes from the coverage area of one cell to another while calling. In this case, the device connects to the new cell in order to prevent interruptions in the user's call. Lastly, a Location Update (LU) occurs when a device enters a new Location Area (LA) (similar to Routing Area in 3G or Tracking Area in 4G) or after a network-determined timer expires. An LA is a group of cells and because a device sends an update when it enters a new LA, the network always knows the approximate location of the device, allowing the device to be contacted in a timely fashion if a request is made by another mobile subscriber [43].

Thus, for active devices, i.e., devices currently calling, texting, or using data, the provider knows the device's location up to the cell level. For currently inactive devices, the provider knows the device's location at least up to the LA level. Figure 3 shows an example of signaling events that could occur in a network covering a section of a highway.

2.1.3 Network probes

Depending on the type of network probe used in the network, some or all of the signaling events described in Section 2.1.2 can be captured. Figure 1 shows four places where probes can be attached in the network architecture, depicted by the letters A through D.

RNC probes, depicted by A in Figure 1, can capture all radio resource control

signals. This allows tracking of all HO, LU, call, text, and data transfer activities, providing an accurate, low-level view of a user's activity and mobility. Furthermore, it allows for tracking certain network performance indicators, such as upload and download speed to the device. RNC probes give the highest spatial accuracy, however, there are many RNC devices and they are geographically distributed over the coverage area. Therefore, RNC probes incur a non-negligible installation and maintenance cost. Lastly, some RNC devices are simply unsuitable attaching probes, which may lead to holes in the collected data.

MSC probes, depicted by B in Figure 1, are located in the Circuit Switched (CS) core. As such, MSC probes can only collect signaling events related to calls and texts, not data. Moreover, signals happening solely within one BSS (e.g., handovers between cells managed by the same BSS) are not sent to the MSC and cannot be tracked there.

GGSN/PGW probes, depicted by C in Figure 1, can be used to collect data on the data interfaces in the Packet Switching (PS) core. Most providers have measurement infrastructure in place to capture information on data usage, such as the user's IP address, session duration, and traffic volume. Providers often place GGSN and PGW in the same location, which allows collecting data on 3G and LTE data traffic simultaneously [15].

GGF probes, depicted by D in Figure 1, collect Call Detail Record (CDR) data. This information is used to bill the provider's customers based on network usage. They contain information on the kind of usage event, e.g. text or call, a timestamp for when the usage started, the duration, and the cell where the event originated. CDRs can also contain information on data usage, in which case it may be referred to as a Data Detail Record (DDR).

2.1.4 Datasets in literature

The authors of [54] give an extensive overview of literature regarding mobility analysis using cellular data. In table 2 of their work, the authors present an overview of datasets used in 56 research papers with various characteristics, such as the network events contained in each dataset. Figure 4 visualizes the relative frequency and co-occurrence of different types of network events in datasets used in the research reviewed in [54]. The occurrence of text events is not included in the figure in order to prevent the figure from becoming too cluttered. Moreover, all datasets in the survey containing text events also contain voice events and the co-occurrence of text events with data and signal events is almost identical to the co-occurrence of voice events with data and signal events.

The figure shows that most datasets in the surveyed research include voice events and that datasets including other network events most often include voice events as well. Signaling events, such as handovers and location updates are least commonly included. Moreover, in four of the datasets that included signaling events only handovers or location updates were included. Only very few datasets include voice, text, data, and signal events (2 out of 56).

2.1.5 Anonymization

Cellular data exposes significant amounts of information about a persons a person's private life, such as visited places of interest (and by extension potentially their religion or sexual orientation), their home and work places, schedules, movement patterns, and their relationships with others. This plethora of data makes cellular data valuable for analysis, but it also poses a risk to the user. Therefore a balance should be found that allows valuable results to be extracted from the data while ensuring a minimum level of privacy.

Different notions of privacy exist that are not necessarily subsets of one another, such as l-diversity [53], k-anonymity [69], t-closeness [48], and differential privacy [2]. The choice for one over the others is still an open question in research and may depend on the specific context.

Currently, mobile telecom providers often apply naive anonymization techniques. Most often, providers replace a user's unique identifier with a random string that is not associated with the user's actual identity. However, this technique may allow users to still be uniquely identifiable. The authors of [82] show that a dataset with cell-level accuracy can be used to identify 10 to 50% of users based on their top 2 visited locations (work and home) and more than 50% based on their top 3 locations. A way to increase the level of anonymity and to make identifying a specific user more difficult is by replacing the random strings that substitute a user's unique identifier more frequently.

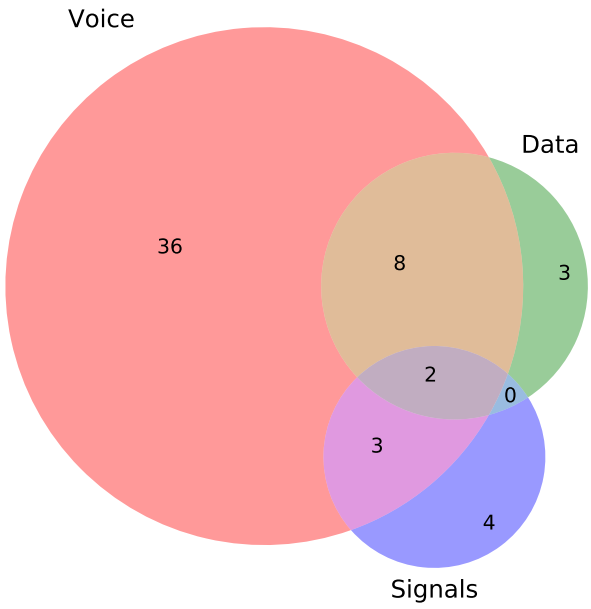


Figure 4: Occurrence of datasets containing different network event types in scientific work surveyed in [54]. Circle size is based on the number of datasets having that network event type.

2.2 Extracting trajectories from cellular data

Cellular data from a mobile provider, such as CDR, DDR, or network event data, provides a user's approximate location at the times at which an event was registered. Depending on the dataset, the spatial granularity can be on a sub-cell level, cell level, location area level, or more coarse if pre-processed by the provider. The temporal granularity depends on which network events are captured in the dataset. For example, data usage events are much more frequent than voice or text events for the increasing number of users that have a mobile data subscription on their device.

Knowing the location of mobile subscribers at different times can provide valuable insights, such as where hotspots are located in a city or how network usage is distributed temporally or over different users. However, a large portion of research focuses on user mobility and in order to analyze that, user trajectories are to be extracted from the cellular data. A user trajectory is the approximate path that a user traveled from one activity to another. Finding a user's trajectory requires to identify when a user is stationary. A simple trajectory can then be the sequence of cells the user traveled through from one stationary moment to the next [60]. More advanced trajectory extraction can use map-matching algorithms to match a user's path to a street or public transport map [55]. The authors of [25] apply this specifically to the problem of extracting vehicular trajectories based on circuit-switched and packet-switched data. Lastly, trajectories can also be matched to each other. This can be used to cluster trajectories, which limits the impact of outliers in the data and may provide more privacy in further analyses of those clustered trajectories [63].

2.3 Human mobility

Cellular data provides an enormous amount of information and, since mobile phone usage and penetration is high, it provides this information on a significant part of the population. This allows researchers to use cellular data to investigate human mobility on a larger scale and with lower cost than traditional methods, such as transportation surveys or census data.

Human mobility research using cellular data is grouped into three categories. The first is identifying fundamental insights and regularities in human mobility, such as common travel patterns among groups of users, distribution of travel distances, and externalities influencing traveling behavior. The second category is research that focuses on using these insights and other methods to form prediction models for user mobility. These prediction models can focus on individual mobility, or mobility of groups. The last category focuses on using these insights and models for better city planning.

2.3.1 Fundamental insights

Human mobility is often regular and systematic. Significant research has been done in capturing the characteristics of these laws of human mobility and their limits. Insights are presented in the locations users visit, the distance they travel, the regularity of their behavior, and how anomalous behavior can be detected.

Visited locations Analyzing cellular data shows that humans generally visit only a select number of locations [32]. Cellular data can also be used to classify users into groups of people with similar behavior, such as tourists, residents, and commuters [13, 29]. Combining this data with land use data allows the identification of user activities and important places in a user's life. It appears that most users have very regular travel patterns, going from home to work and that their home and work place locations can often be estimated based on their data [5, 41, 80]

Relate users based on visited locations Some research has sought to relate users to each other based on their co-visited locations. In [74], the authors use an extensive dataset to relate similar users to each other. They use data about users' app usage, such as when they use dating apps or listen to music, combined with information about users visiting the same locations to compare different users. A higher-level view is taken in [49], where zones in the city of Shanghai are indirectly related to each other through looking at when of residents from those zones are located in the same area.

Travel distance The authors of [36] find that users concentrate their activity in one cell and make frequent shorter trips to other locations. They also find that users have very low mobility in general. Users generally travel very small distances with a high probability of returning to a few highly frequented locations [32].

Spatial and temporal regularity As briefly stated previously, users show a high degree of regularity in their mobility patterns, both regarding the places they travel to and the times at which they travel. Looking at cellular data from Rome, [57] shows that mobility patterns were very regular as long as weekdays and weekend days are analyzed separately. The authors of [66] show that as much as 93% of user mobility in their user base is potentially predictable. They furthermore note that this figure does not depend on the distance a user travels, so users traveling larger distances generally show equally regular behavior as users traveling smaller distances.

Influences on user mobility Many external factors can influence human mobility. A factor is whether a user resides in an urban or countryside setting [14]. Another external influence is rainfall in the region a user is active [28]. And also literacy of a user can have an influence on that user's mobility patterns. In [46], the authors find that less literate residents of Dakar have fewer contacts and fewer weak ties and that they do not compensate for this by being more mobile.

Detecting irregularities Although human mobility is often very regular and predictable, irregular situations do occur. Detecting these deviating events and situations can further help understand human mobility and anticipate transportation demand. In [8], the authors use CDR data to detect regular movement patterns and users' deviations therefrom, such as large social events. The authors of [12] identify attendees of known social events and analyze their origins. They find that the closer someone lives to an event, the more likely it is they will attend that event. Moreover, the authors find that similar events have similar geographical distributions of the attendees' origins. In [72], the authors use cellular data to detect social events as opposed to analyzing known social events. They focus on probabilistic inference of events and also estimate the probability of specific users having attended the event.

2.3.2 Prediction models

The previous section has focused on general laws and insights in human mobility, noting that human mobility is often regular and predictable. This section introduces research on using these insights in developing mobility models that can be used to predict individual mobility and aggregated mobility.

Individual mobility prediction models Much research has been done in modeling individual mobility [65, 17]. Some work focuses on predicting user mobility on a small scale. In [26], a Gaussian mixture model is presented that probabilistically models user movement in between calls. As CDR data usually has a very low spatial granularity, better estimates for user movement in between data points can be valuable. On a similar scale, model have been presented to predict inter-cell movement from a user's historic movement pattern [81], or to predict dwell time of users at certain locations [19]. In [31], a Markov chain model is presented that can predict the next location of a user with an accuracy of 70-95% using the user's previous two locations. Since location history for a single user is often limitedly available, the

authors of [44] present an algorithm for clustering similar users in order to expand the historic data that can be used to predict a user's future location.

Other work has modeled user mobility on a larger scale. In [27], the authors predict the end-to-end route of a vehicle based on previous routes taken by the driver. This is possible due to drivers exhibiting very regular behavior in the routes they take to their regular destinations. The authors use GPS data, but a similar model might be created based on cellular data. In [20], a general location analytics framework is presented that can analyze data from sources with different spatiotemporal granularity. The framework can be used to give information on where users visiting specific areas come from, where they travel next, or how long they stay at the location.

Estimating the actual human trajectory from a spatiotemporally coarse-grained trajectory can be done using different interpolation techniques. The authors of [38] identify linear interpolation as the most accurate technique for sedentary (stationary) users and cubic interpolation as best for commuters. They also find interpolation methods to be most accurate within the user's radius of gyration.

Aggregated mobility prediction models User mobility can also be modeled for groups of users instead of individuals to give insights with coarser spatial granularity, such as on movements between city districts or between different cities. This can provide less privacy-sensitive, but still valuable information on users, such as expected travel demand for certain routes. For example, in [51], the travel demand of workers in Senegal is modeled using cellular data. They cluster users together and model their probabilistic distribution of activities and the travel patterns associated with them.

Aggregated mobility can be predicted with great accuracy due to the regularity of human mobility patterns. The authors of [52] show that a first-order Markovian model can predict daily human mobility in Cote d'Ivoire with 88% accuracy. This shows that human behavior is not only theoretically predictable with high accuracy, but that it is also practically feasible to get close to that.

2.3.3 City planning

The insights from mobility analysis can provide a direction in how cities could be planned better. This can be based on general analysis and insights, such as those presented in Section 2.3.2, or for very specific improvements on certain tracks or areas based on more specific analysis, such as those presented in Section 2.3.2.

In [33], the authors propose a methodology to find suitable locations for new amenities or new locations existing amenities. They look at amenities, such as home, shop, work, or leisure locations, and optimize for travel time and travel distance for residents. They use a multi-agent simulator to simulate to generate agent plans and apply their method on the Dakar region. In [21], instead of looking at where to locate amenities, the authors look at which regions to invest in. The idea is to test urban planning theories and improve government policies. Qualitative evaluations of the suggested regions for investments show promising results.

2.4 Transportation systems

A specific research focus of numerous papers is detecting characteristics of transportation systems through cellular data analysis. Subcategories in this research area are presented in the following sections. Section 2.4.1 summarizes research on detecting travel time and congestion; Section 2.4.2 on detecting mobility flows of people from origin to destination; Section 2.4.3 on improving transportation infrastructure planning; and Section 2.4.4 on travel mode discovery.

2.4.1 Travel time and congestion

Traditional methods of measuring traffic counts are surveys and sensors build into the roads. These methods are costly to implement, not real-time in case of surveys, and often only present on a small subset of road segments in case of road sensors. Cellular data provides an opportunity to make traffic count and congestion detection more affordable. Since providers already collect a lot of the needed data in order to bill their customers, using this data for traffic analysis comes with little additional costs. One challenge is to utilize and combine data from different types of network probes. Valerio *et al.* [75] set out a roadmap on how to do this for the purpose of road traffic estimation, which later work has built on. A final benefit of utilizing cellular data is that mobile penetration is very high and mobile (data) usage increasing. This makes it more likely that cellular data can provide a representative view of transportation systems.

Many papers, such as [50, 9, 84], focus on estimating traffic counts on certain roads or road segments. They compare these estimates with counts from road sensors to show that cellular data seems promising for approximating traffic counts.

Some work focuses on extracting other information from cellular data, such as generating high-resolution traffic maps with how information on how busy segments usually are [23], estimating previously unavailable city-to-city travel times in Senegal [47], or deriving public transportation timetables [37]. In this last work, the authors use public OpenStreetMap data to filter on travelers using the train in order to derive train timetables. They manage to derive departure times of popular train connections with an 89% recall rate and a maximum deviation of 5 minutes.

Another prominent research area is congestion or infrastructure load detection. Some research looks at public transport systems, such as [1], in which the authors look at the occupancy and origin–destination flows in the Paris underground transit system. Other research aims to monitor traffic in real-time, often with the aim of detecting congestion as quickly and reliably as possible. In [3, 4, 42], traffic monitoring on highways using cellular data is compared with traffic monitoring data gathered from road sensors. Congestion can be detected in a timely and reliable fashion using cellular data in the context of highway monitoring as shown in these cases. The same approach has also been applied to other areas other than highways. In [10], traffic is monitored in real-time in the city of Rome and combined with other data systems, such as bus movements and taxi trackers. The authors of [43] similarly monitor traffic in highway, urban, and semi-urban areas. They use cellular data from both active and passive devices to detect anomalies and then use events from active

devices only to improve the spatial accuracy of the estimate for the congestion event. Their system detects congestion 3 minutes faster on average than traditional road monitoring based on road sensors. Furthermore, their method detected all congestion events and reported no false positives.

2.4.2 Mobility flows

Cellular data can be used to extract user mobility flows, also known as origin–destination matrices. These are in turn widely used to analyze infrastructure usage and to allocate future infrastructure investments. Because of the coarse spatial granularity of cellular data, especially when aggregated over many users, mobility flows are usually estimated between larger geographical areas. In [35], for example, an origin–destination table is created for intercity movements between people in Israel. The model matches well with official traffic counts. Similarly, the authors of [7] construct an origin–destination matrix on a regional level within the Île-de-France region. Compared with traditional survey data, their estimates are off by no more than 9%.

Mobility flows can also be estimated on a somewhat smaller scale, such as areas within one city. A travel demand analysis is presented in [34] that takes measures to protect a user's privacy through site sequence clustering and frequent sequence extraction. The authors evaluate their method against official traffic count numbers and show a correlation of 89% between the predicted and actual values. On a similar scale, [11] infers origin–destination matrices between counties in eastern Massachusetts. They can distinguish between weekday and weekend day travel patterns, as well as work and non-work trips and whether special events are happening. On a county level, the authors find a coefficient of determination (R^2) of 0.76 for the origin–destination matrix based on cellular data compared to census data of commuters in the area. The authors of [30] determine mobility flows of visitors in Pisa, Italy. They find that visitors have a high mobility and are very likely to move across the city.

Lastly, in [73], the authors perform a mobility flow analysis on three different levels, the intra-city level, city level, and department level. They find origin–destination matrices between different regions in France at different granularities, but unfortunately lack a comparison ground truth information, such as census data.

2.4.3 Transportation infrastructure planning

Cellular data analysis can be used to detect infrastructure load or mobility flows of people between areas. This data can be useful in itself, e.g., knowing a road is congested allows other traffic to be routed via different roads. However, mobility analysis can also help prevent such congestion in the future by helping discover effective places to invest in transportation infrastructure.

For example, in [78], the authors look at origin–destination matrices and compare that to their expected travel demand between regions based on the number of calls between regions in Senegal. They use this information as input to an optimization algorithm to find recommendations for road infrastructure upgrades. Similarly, in

[64], a model is presented to find high traffic roads that are frequently congested in order to give insight into where infrastructure investments would make sense.

Conversely, in [24], the author does not present suggestions for future investments, but analyses the effect of a new toll road in Dakar opened in 2013. The paper finds that the average commuting behavior increased statistically significantly and that the average travel time was reduced by 7.8% for all origin–destination pairs traveling from an area on one side of the toll road to an area on the other side.

The authors of [76] present a model to predict and anticipate transportation demand. They test the model in a case study in Hong Kong. Their model expects a list of planned activities for each user as input, then identifies where these activities can be concluded and at what time to model their transport demand. The need for a list of planned activities makes this model an initial case study rather than something that can be applied on a large scale.

2.4.4 Travel mode discovery

Discovering the transportation mode a traveler uses, allows for better planning of infrastructure investments and public transport services. Information about certain bus routes being over- or under-utilized enables public transportation companies to make more informed decisions on how to improve their service. Similarly, local administrators can use this data to evaluate the impact of policies aimed to reduce car usage in their region. Cellular data analysis provides a valuable alternative to traditional methods, such as surveys, because it is affordable, covers a large part of the population, and can provide real-time data.

In [68], Stopher *et al.* present a model to deduce travel mode and the travel purpose of trips made by users. They combine trip data with GIS data and user-provided information on their work address, home address, and shops they frequent to identify the purpose of a trip. To identify the travel mode, the authors use street map data, including data on bus routes and bus stop locations. However, the presented model uses GPS data to make traces instead of cellular data and is not directly generally applicable due to the personal information needed from users.

Wang *et al.* [77] use CDR data to find trips taken by travelers in Boston. Travelers with similar origins and destinations are clustered into two groups based on their travel time using the k-means unsupervised clustering algorithm. The group with a shorter travel time is presumably using the car and the group with a longer travel time presumably using public transport to get to their destination. Figure 5, an example is given of the travel time distribution for a specific origin–destination pair. The figure shows the average travel times for the two clusters: 13.4 minutes for the car cluster and 42.4 minutes for the public transport cluster. The authors compare the average travel time per cluster with the travel time estimated by Google Maps for the car and public transport to find an average error of around 5–6 minutes. Finally, Wang *et al.* find that the resulting statistics on transport mode usage match well with external survey data, indicating that this method can provide accurate insights into aggregated travel mode usage in cities.

The most similar work to the research presented here, is introduced in [22]. Doyle

et al. define a Virtual Cell Path (VCP) for each route of interest, consisting of all cells a user may travel through while on the route of interest. Routes of interest are assumed to overlap at most partially. User paths predicted from CDR data can then be compared to the VCPs to find the best match. The authors present a case study wherein they discover whether travelers used the highway or railway between Dublin and Cork in the Republic of Ireland. They manage to classify around 90% of travelers on that route. Unclassifiable trips lack CDR data for areas where the highway and railway routes differ.

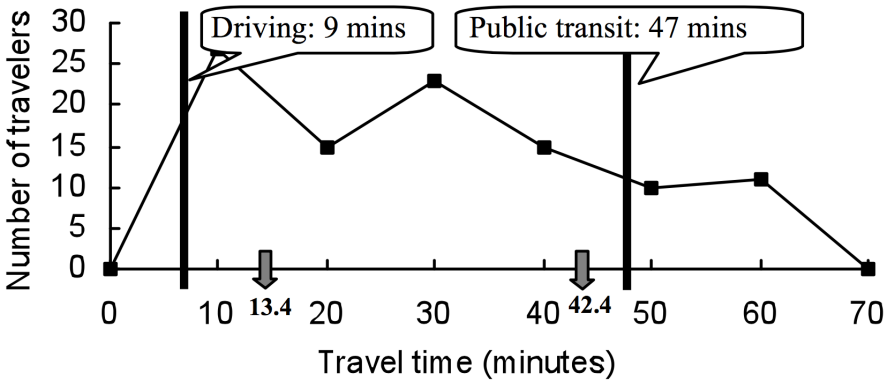


Figure 5: Travel time distribution and average travel time for two clusters [77]

2.5 Reliability and limitations of using cellular data for mobility analysis

Significant effort has been spent on validating the reliability of cellular data for mobility analysis. Cellular data, especially CDR data, is widely available and widely used in research, but depending on the research area, may introduce a bias in the conclusions drawn from it. Therefore, researchers have looked at where and to what extend this bias exists.

In [58], Rose provides one of the earliest research overviews of using cell phones as traffic probes. Rose recognizes the potential of using cellular data to estimate properties such as travel time, stating that limited evaluation results suggest this is feasible with an error rate of 5 to at most 20%. However, since mobile phone penetration is not equally distributed of different genders, socioeconomic backgrounds, vehicle type, or age, there is at least the possibility of bias. Jimenez [45] mentions similar challenges to Rose, such as the limited spatial granularity of cellular data, especially in sub- or extra-urban areas. Another challenge for road traffic analysis is the noise of non-road travelers, such as pedestrians, in the dataset as well as the possibility of having multiple cell phones in one car.

Naboulsi *et al.* [54] find from literature that the extend of bias introduced by CDR data depends on the analysis type of the study. There should be distinguished between the reliability of aggregated mobility analysis and individual mobility analysis studies.

2.5.1 Reliability of aggregated mobility analysis

Steenbruggen *et al.* [67] identify mobile phone data as a promising data source for smarter cities, but they recognize that real-life examples of using such data in urban management are currently sparse. CDR data is the most widely available cellular data source, yet it is also one of the data sources with the lowest spatial-temporal granularity. This means that CDR data allows, for example, to detect anomalies in data, without being able to infer the cause of the anomaly.

Schneider *et al.* [61] show that regular aggregated mobility patterns based on CDR data can capture up to 90% of the population in surveys. Bengtsson *et al.* [6] extend this by analyzing the irregular situation occurring after the earthquake in Haiti in 2010. They find that their outcomes for number of people fleeing the struck area after the earthquake are comparable to the results of a retrospective survey commissioned by the United Nations, indicating that mobility flows can also be estimated based on CDR data in anomalous situations. Commuting networks match well with empirical data in a study by Tizzoni *et al.* [70]. However, the authors find that commuting traffic is systematically overestimated using the CDR data.

Lastly, Wesolowski *et al.* [79] confirms that biases in CDR data can skew mobility analyses. Looking at user mobility in Kenya, they find that mobile phone ownership and usage is not distributed uniformly across the population. They conclude that people that use their phone more, tend to travel farther than other users. This ostensibly makes overestimating the mobility of a population likely. However, although

the authors find some overestimation regarding mobility, the effect was minimal.

2.5.2 Reliability of individual mobility analysis

Iovan *et al.* [40] observe that analyzing the mobility of users with few data points introduces significant spatial uncertainty, however, filtering on people with more data points during the day introduces a bias, since users that communicate more often, are more likely to be more more mobile. This means that users should be selected carefully. Data sets that include more network events than CDRs, may provide less spatial uncertainty while introducing a more minimal bias.

In [83], Zhang looks at the different characteristics of CDR data compared to DDR data. CDR data is largely based on calls, which are more concentrated in the afternoon or evening, whereas DDR data shows a more regular access from users over the day. Using DDR data, a lower radius of gyration is detected and more important locations are identified than using CDR data. This implies that the increased spatial granularity of DDR data leads to more accurate insights and that the dominant home/work-based mobility model may need revision.

Gonzalez *et al.* [32] state that mobility flows among a person's most significant locations can accurately be deduced from voice and text CDRs. However, Ranjan *et al.* [56] actually find that CDR data are biased towards a user's most important locations.

Comparing human mobility ranges with ground truth GPS traces from volunteers, Isaacman *et al.* [41] show a typical error between the estimated trace based on CDR data and the GPS trace to be in the order of 1 mile. However, Schulz *et al.* [62] thesis a bigger difference. They find that using GSM data underestimates human mobility by almost half when compared to GPS traces. Moreover, they also find that the radius of gyration of users is underestimated, although frequent stop locations can be detected accurately.

Another form of ground truth data, is information from traffic loops in roads. In [60], the authors find a good match between traffic flow measured by road loop detectors and their cellular data-based mobility analysis for people driving on a highway between Stuttgart and Walldorf, Germany. They note that their results are somewhat skewed since their model required at least location updates, which means a trip has to span at least three location areas unless a network timer would force intermediate location updates.

2.6 Summary

This work uses a dataset of Data Detail Records (DDRs) for the analysis. These records include more network event types than most works in this field, such as voice, text, data, and signaling events. As such, the dataset is comparable to 2 out of 56 datasets used in related papers reviewed in [54] and displayed in figure 4. A DDR dataset provides a finer temporal granularity than a CDR-based dataset, since more events are captured and data events tend to occur more often than call or text events. For both types of dataset, user locations are known at least at the LA level and at most on the cell level. However, since a DDR dataset generally has more frequent events, it is more likely to know the cell a user is in. Therefore, a DDR dataset has a finer spatial granularity as well.

The dataset used in this work is pre-analyzed by a mobile network provider in order to extract user trajectories. The data is also anonymized beforehand by replacing users' unique identifier with a random string that is not associated with the user. This identifier is replaced by a new identifier every 24 hours to prevent tracking users for longer periods. This work focuses on further analysis using these extracted and anonymized trajectories; it does not seek to find the limitations of the anonymization or trajectory extraction techniques, nor to improve upon them.

The primary aim of this work is travel mode discovery in an urban area. Therefore, related work presented in Section 2.4.4 is closest to the work presented here. Wang *et al.* [77] use CDR data to cluster travelers in a city, that have a similar origin and destination, based on their travel time. Assuming that the slower cluster used public transport, they find aggregated travel mode data that matches well with external survey data. This work differs from [77] by discovering the travel mode of individual users instead of groups of users. This allows a comparison with ground truth data from volunteer travel trajectories, as opposed to comparing with survey data. Doyle *et al.* [22] construct user paths between two Irish cities using CDR data. They compare these paths with virtual paths using different transport modes between the two cities to classify the transport mode used by individual users. The algorithm presented by Doyle *et al.* works on an inter-city scale and they distinguish between travelers using a car and travelers using a train using the assumption that both use different paths. This work presents an algorithm that works on an intra-city scale that distinguishes between travelers using a car and travelers using a bus where both do not necessarily use different paths.

The travel mode discovery algorithm presented here can be used to estimate travel times and especially to find the difference between travel times using different transportation modes. It is therefore also related to the work presented in Section 2.4.1. Similarly, although this work does not present recommendations for infrastructure planning, the presented algorithm can estimate the number of people on buses throughout the city at different times of day, therefore enabling such recommendations rather straightforwardly in the future. Lastly, whereas reliability analysis of cellular data for mobility analysis is not the core of this work, the functioning of the presented algorithm is evaluated against ground truth data gathered by volunteers, similar to [41, 62].

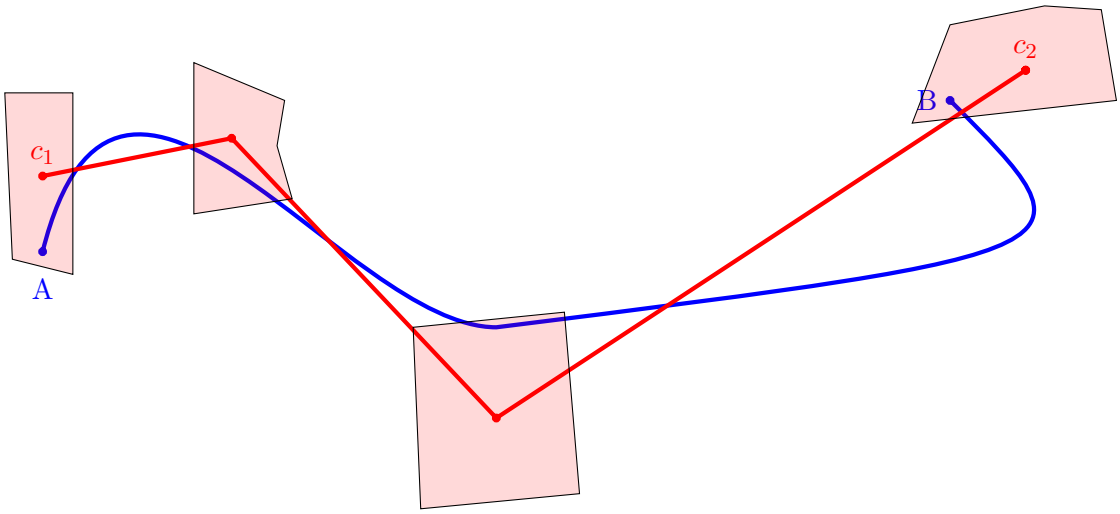


Figure 6: Visualization of an extracted mobility trace (red) and the actually traveled path (blue)

3 Concept and Design

Phones in a mobile network generate network events. This happens when a phone is used to make a call, send a text message, or to connect to the Internet. Furthermore, some network events occur to enable basic network operation, such as handover or location update events. Providers may capture and store these network events for billing or analysis purposes in call detail records (CDRs) or data detail records (DDRs). CDRs generally contain call and text events, and sometimes also handover or location update events. DDRs contain data events, however they often also contain the events usually contained in CDRs, thus DDRs usually provide more data than CDRs.

A CDR or DDR contains per network event the network cell from which the event originated. Therefore, CDRs or DDRs essentially store the phone's location at the level of a network cell at different points in time. This information can be used to extract a mobility trace for that phone. Figure 6 shows the path a mobile phone user took from place A to place B in blue. The phone generated network events in the cells shown as red polygons in the figure. A simple mobility trace can then be visualized by drawing a line from the center points of each consecutive cell in which network events were recorded. Such a trace is visualized by the red line.

A main challenge in extracting mobility traces from CDRs or DDRs is detecting when a trace starts and ends. Phones often switch cell towers if another cell tower provides a stronger signal than the current one. This may also happen when the phone is (almost) stationary. For example, if the signal from a cell tower is blocked in one room, but not in another, a phone may switch cell towers while remaining in the same building. Such cell tower switches have to be detected as not being related to actual movement, whereas other similar events do indicate the start of a trace. In

this project, mobility traces are analyzed, however, extracting these traces is outside of the scope of this project.

The aim of the developed software is to analyze CDR/DDR-based traces to discover the travel mode of mobile phone users, and specifically to distinguish between mobile phone users traveling on public transportation buses and mobile phone users traveling using other transportation modes. The requirements that arise from this main aim are presented in Section 3. Section 3.3 presents a high-level overview of the design. This includes the required data sources and logical processing steps in the algorithm. sectionFunctional requirements The main goal of the developed software is to accurately identify the transport mode corresponding with certain CDR/DDR-based traces, specifically traces that were covered by bus. This aim leads to the following functional requirements:

RQ 1 The system must be able to distinguish CDR/DDR-based traces traveled by public transportation buses from traces traveled by other modes of transport.

- A This classification needs to be sufficiently accurate.
- B The system must be tuneable in order to provide sufficiently accurate results for different datasets.

RQ 2 The system must be able to provide an estimate for the number of people traveling by bus through a specific area or ending their trip at a specific area, in a specified time-frame.

- A The system must be able to filter traces based on whether those traces go through a specific area or end at a specific area.
- B The system must be able to combine the area filters to filter on traces going through or ending at an area within a specific time-frame.
- C The system must be able to filter on traces corresponding to a specific bus line at a specific time, e.g., traces corresponding to bus line 100 leaving the central station at 10:04.

RQ 3 The system must not unnecessarily rely on private data sources.

RQ 4 The system must be able to run distributedly.

In addition to the functional requirements presented above, the following non-functional requirement has been identified:

RQ 5 The system's execution time should increase at most linearly with increased input data.

3.1 Roles

Different people interact with the developed system in different capacities. Figure 7 gives an overview of the roles from which people interact with the system. On the

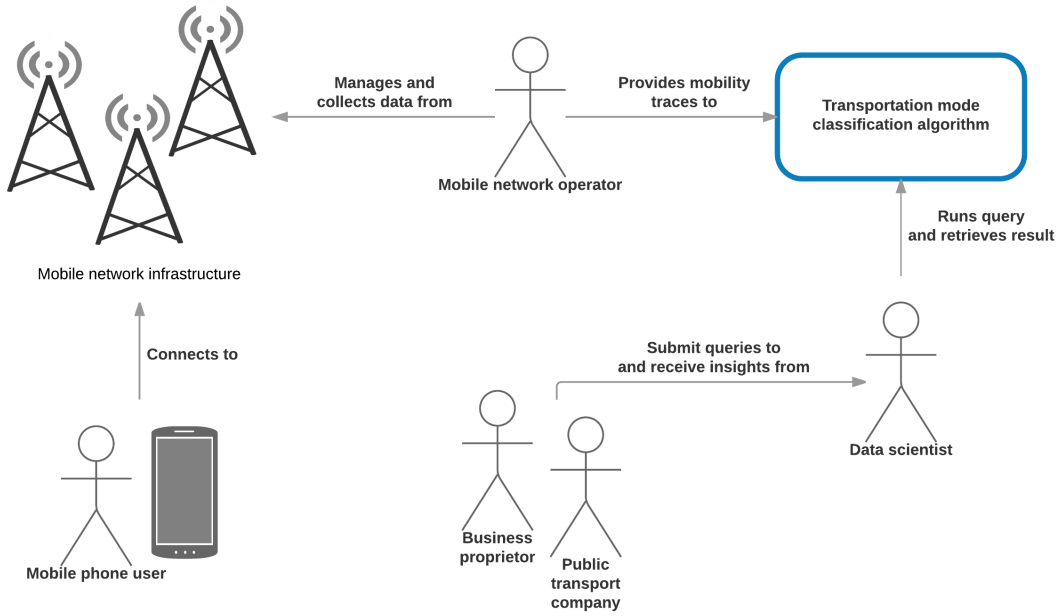


Figure 7: Roles of people interacting with the system

bottom left, a mobile user is depicted. They use their phone to connect to the mobile network infrastructure of their operator. The mobile user can then use the services provided by the network, such as calling, texting, or accessing the Internet. The mobile network infrastructure is managed by the mobile network operator. The operator collects data needed to bill the mobile phone user correctly and for technical reasons, such as load-balancing and infrastructure planning. The mobile network provider can extract mobility traces from the data they collect and provide that to the transportation mode classification algorithm presented in this document.

Local business proprietors and public transport companies, depicted in the bottom center of the figure, can submit queries about the number of users using certain transportation modes to a data scientist. A relevant query for a business owner might be "How many people travel to my store during opening hours using public transportation buses?", whereas a public transportation company could be interested a query such as "How many people travel by bus via a specific bus station on Tuesday morning between 8:00 and 10:00?". After the data scientist receives such queries, they can set the appropriate parameters and run the algorithm. Following the execution, they provide the resulting insights to the business proprietor or transportation company.

3.2 Architecture overview

The set of input mobility traces can be very large, especially when they concern a larger area, e.g., nationwide, or a longer time-frame. As a consequence, distributability is a key concern for the developed algorithm as stated in requirement RQ 4. This

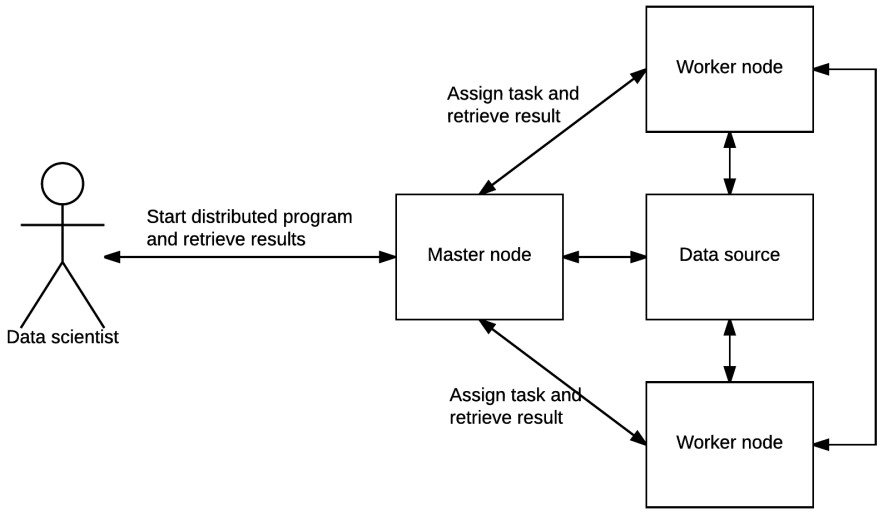


Figure 8: Conceptual distributed system architecture

has implications for the design of the software. In Figure 8, an overview of key components in a distributed system architecture is presented. The figure shows a master node, two worker nodes, and a data source. The master and worker nodes can each be run on separate machines. These machines together form a cluster. The data source can be a distributed file system, a database, or any other data source that is accessible from any machine in the cluster. When executing a distributed program, the master node is the entry point. The master node divides the main task of the program into sub-tasks and assigns these to the worker nodes. The data needed to execute the sub-tasks can be accessed in two ways. One option is that the master node assigns each worker node a partition of the dataset to work on and the worker nodes retrieve the data they need directly from the data source. Alternatively, the master node can retrieve all data from the data source, partition it, and send the appropriate subsets to the individual worker nodes.

A worker node can exchange data with another worker node. Such repartitioning may be necessary if a later step in the algorithm requires different data to be located on the same worker node than an earlier step in the algorithm. However, accessing data on another node is generally inefficient. Therefore, the algorithm should be designed to minimize the number of times data is partitioned and distributed over the worker nodes.

3.3 Algorithm overview

The algorithm classifies a mobility trace as either corresponding to a travel by bus or as corresponding to a travel by a non-bus transportation mode. Figure 9 depicts an overview of the high-level design of the algorithm. The algorithm overview shows he steps of the algorithm in order from top to bottom. Each trace is classified by the algorithm into one of three categories. A trace can be classified as irrelevant for the

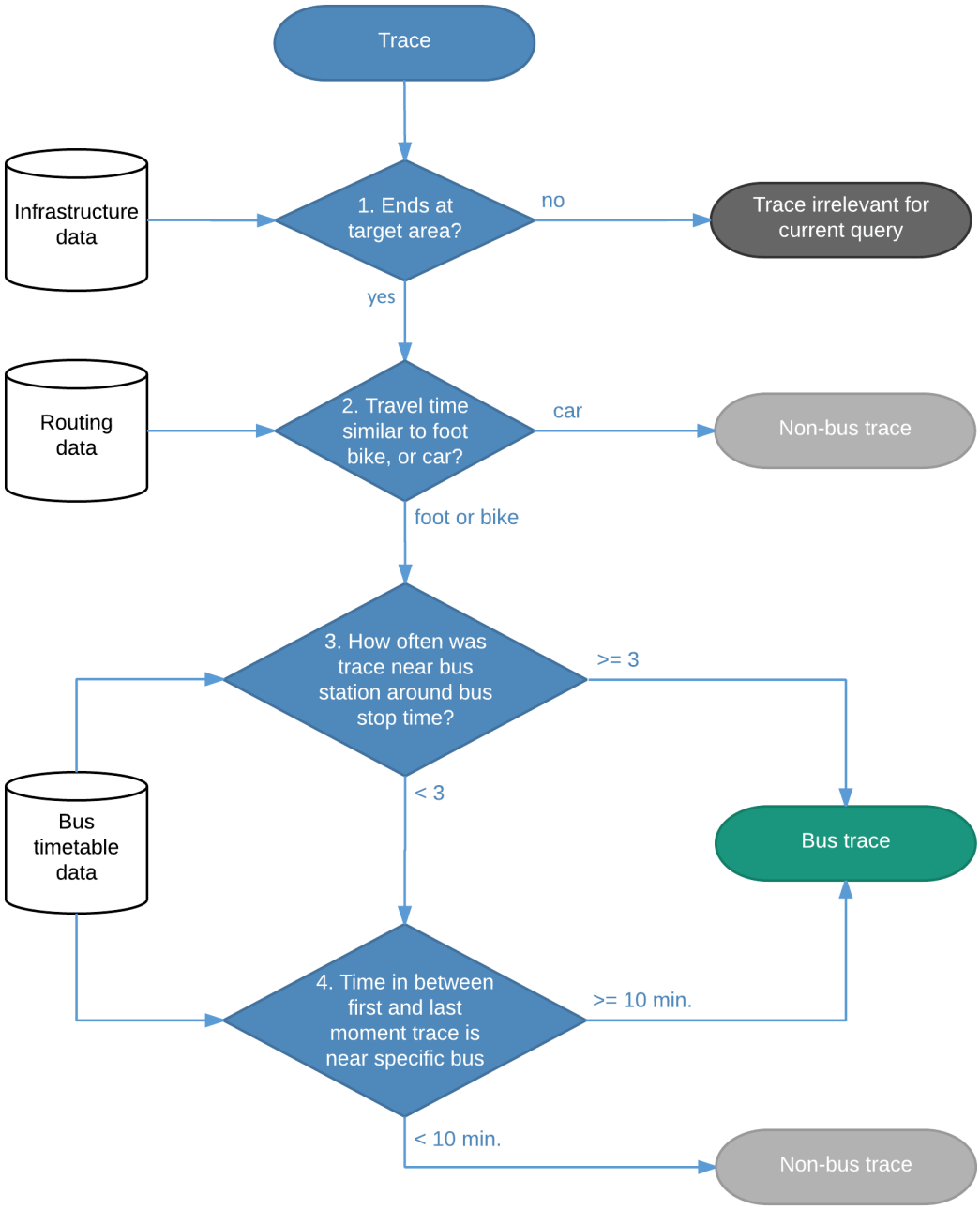


Figure 9: Overview of the high-level algorithm design

current query, as a bus trace, or as a non-bus trace. These categories are shown on the right side of the figure.

The main input for the algorithm are mobility traces. A mobility trace is a sequence of network cell identifiers with timestamps. It corresponds to a trip made by a mobile phone user from a point at which they were stationary until the next

time they are stationary. A trace does not include any geographical information in itself. However, the cell identifiers can be combined with network infrastructure data to find the coverage area of those cells. This network infrastructure data belongs to a network operator and it contains information on the cells in the operator's network. The infrastructure data contains per cell, the cell's unique identifier, location area code (LAC), estimated coverage area, and the network standard it operates under. The estimated coverage area represents the geographical area that is covered by the cell's signal. Thus, each trace can be represented as a sequence of areas with timestamps by finding the coverage area for all cell identifiers in that trace.

Traces can then be visualized as lines on a map by connecting the center points of these areas. Figure 10 shows four traces visualized using this method (in red). The cell coverage areas corresponding to the traces are represented by yellow polygons. The figure furthermore shows two bus lines in black and an area of interest as a green rectangle. The area of interest could be an area around a specific shop. The algorithm could then provide insights into the transportation modes of people traveling to that shop.

Apart from mobility traces, the algorithm uses three other types of input data, shown on the left in Figure 9. Firstly, the infrastructure data that matches network cell identifiers to geographical coverage areas; secondly, routing information that is used to estimate the travel time from trace origins to trace destinations; and lastly, bus timetable data, which is used to find all buses that stop in a certain network cell around a specific time.

In the following sections, the four steps of the algorithm are explained. For each step, the structure of the relevant data sources is mentioned, as well as how they are used in that step. The traces in Figure 10 are referenced in the steps to help visualize the results of each step. In Figure 9, specific values are shown for parameters in the algorithm, such as the travel modes that are further analyzed after step 2, the threshold number of 3 in step 3, and 10 as the threshold for step 4. For simplicity, the specific values shown in the figure are assumed in the following sections. However, in reality these values are configurable as per requirement RQ 1 B.

3.3.1 Filter traces on passing through or ending at a specified area of interest

The algorithm allows a data scientist to specify an area of interest. They can then specify if only traces should be analyzed that end at the area or also traces that pass through the area. This filtering happens in step 1 in Figure 9.

At the start of step 1, each worker node is assigned a partition of the traces dataset. For each of those traces, the worker node examines the coverage area corresponding to the last cell identifier of the trace. In order to find that coverage area, each worker obtains a copy of the infrastructure data from the master node. If the coverage area of the last cell in a trace overlaps with the specified area of interest, the trace is analyzed further in the second step. If it does not, the trace is excluded from further analysis. After this step, trace 1 in Figure 10 is filtered out, since it does not end at the area of interest. Traces 2, 3, and 4 are analyzed further. The functionality of

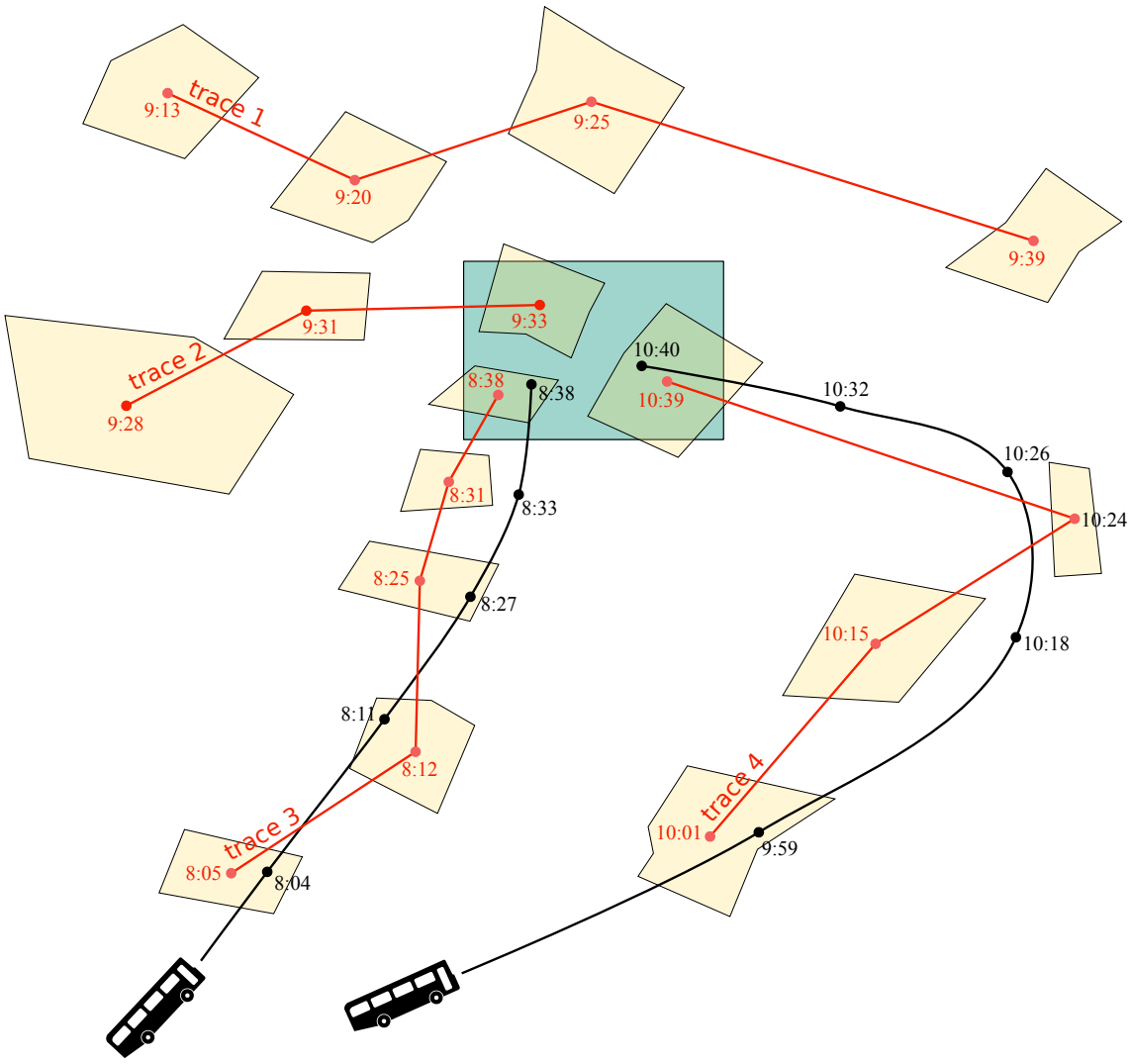


Figure 10: A visualization of mobility traces (red), network cells (yellow), bus lines (black), and an area of interest (green)

this step fulfills requirements RQ 2 A and RQ 2 B.

3.3.2 Filter traces on initial transportation mode classification

In step 2, the algorithm filters out traces that are deemed unlikely to correspond to a bus trip. The algorithm marks a trace as unlikely to be a bus trace if the travel time of the trace does not correspond to the fastest travel time by bus for the same origin and destination. The fastest route from the start of a trace to the end of a trace can be approximated by the fastest route from the center point of the first cell in a trace to the center point of the last cell in a trace. Using the center points of the cells' coverage areas introduces an error, since the mobile user could have been anywhere in the cell at the time of measurement. The center point was chosen, because it is straightforward to calculate and because the actual location of the mobile phone user is not known.

Multiple traces may start in the same cell and end in the same cell. Therefore, the fastest route is calculated once from all network cells to all other network cells. This information is then stored. The infrastructure data is partitioned by cell identifier and distributed over the worker nodes. Each worker node calculates the fastest route from the centers of the cells in their partition to the centers of all other network cells. Every worker node therefore also needs a copy of the whole network infrastructure dataset. The fastest route from the center of the cells to the center of all other cells is calculated for traveling with different transportation modes, namely by car, by bike, and by foot. The routing data used for this should preferably be open source, following requirement RQ 3. After this first stage, the travel times for different transportation modes from all cell centers to all other cell centers are stored.

For the second stage, the traces are distributed over the worker nodes and each worker node retrieves a copy of the data calculated in the first stage. For each trace, the worker node finds the earlier calculated fastest travel times for different transportation modes. It then compares the duration from the start of the trace to the end of the trace to the calculated fastest times. If the duration from the start to the end of the trace is most similar to the calculated travel time by car, it is classified as a non-bus trace. Cars generally travel significantly faster than buses. Therefore a trace with a travel time comparable to that of a car is unlikely to correspond to a bus. Trace 2 in Figure 10 has a very short travel time compared to the distance it covers. It would be classified as a probable car trace in this step and excluded from further analysis.

Filtering out traces at this point in the algorithm, lowers execution time in the remainder of the algorithm. If the data scientist has specified to filter traces on them ending at a certain area, the algorithm can calculate the fastest routes from all cell center points to that area once. In that case, filtering out traces based on travel time is especially efficient.

3.3.3 Classify traces based on the number of matches with a bus

After two filtering steps, the third step in the algorithm examines how often a trace was near a bus station around the time a bus stopped there. First, a data structure is

created that facilitates a quick lookup of all bus stations in a particular network cell based on the cell identifier. For this, the network infrastructure data is partitioned and distributed over the worker nodes. The worker nodes also retrieve a copy of the bus timetable data. Every worker then processes each network cell in their partition. Per network cell, all bus stations that are contained in that cell are listed. The resulting data structure is stored.

Afterwards, each worker node receives a partition of traces again. For each cell identifier in each trace, the worker nodes retrieve all bus stations. With a bus station identifier, the bus stop times at that bus station are retrieved from the bus timetable data. These bus stop times are compared to the timestamp in the trace. If a bus stop time is close to the timestamp, the algorithm marks that bus stop event as a match. Thus, a trace "matches" a specific bus if a trace has a cell identifier corresponding to a coverage area in which that bus stops at a time close to the timestamp in the trace. Per requirement RQ 2 C, the algorithm can be set to only match a specific bus and ignore all other matches.

Trace 3 in Figure 10 contains cell identifiers for four cells that contain a bus station. The timestamps in the trace are close to the bus stop times in all four cells (the time difference is ≤ 2 minutes in this case). Therefore, the trace matches the bus four times. Since it is the same bus all four times, this trace is classified as a bus trace and not analyzed further. In general, the algorithm classifies a trace that match the same bus at least three times as a bus trace. If that is not the case, the trace is further analyzed in step 4.

3.3.4 Classify traces based on the time between the first and last match with a bus

If a trace matches any one bus less than three times, the worker nodes examine buses that were matched only two times. In step 4, the time between the first and the last time the trace matches a bus is calculated. If the time between the first and last match is less than ten minutes, the trace is regarded as a non-bus trace. If the time between first and last match is at least ten minutes, it is considered a bus ride. After all, it would be very coincidental if a mobile user is around two bus stations at the same time as a specific bus if the stop times are very far apart. Trace 4 in Figure 10 matches one bus only two times, i.e., two cell coverage areas corresponding to cell identifiers in the trace contain bus stops at which a bus stops around the time of the trace timestamp. One match occurs at the start of the trace and one at the end of the trace. Since the trace has less than three matches, it is not classified as a bus trace in step 3. The time difference between the first and last match is 38 minutes (10:01–10:39). Therefore, the trace is classified as a bus trace in step 4.

4 Implementation

This chapter extends on the Concept and Design chapter by describing the implementation of the software. In this chapter, an overview of the software architecture and used technologies is presented. The steps of the algorithm are visualized using a diagram to provide an overview of the execution process. Technical implementation decisions are discussed and justified. At the end of this chapter, a brief explanation of the software's usage and output is given.

4.1 Software architecture

Mobile network operators often bill customers based on their usage of the operator's network. For this purpose, operators collect network events related to calls, text messages, and data usage in Call Detail Records (CDRs) and Data Detail Records (DDRs). Many national operators have millions of customers generating such events. As a result, datasets with CDR and DDR data are often very large. Analyzing such datasets requires a distributed program, running on multiple machines. Therefore, the travel mode classification algorithm is implemented as a distributed program. The algorithm processes data in batches. It can run at regular intervals, e.g., once per hour or once per day, to analyze the data collected in the last interval.

The classification algorithm has been built on top of the Apache Spark framework[2]. Apache Spark enables the creation of a distributed program using distributed data structures that are built into the framework. Spark was chosen as the base framework, because it is a mature technology with an active support community. Additionally, Spark is well-documented, relatively easy to setup, and Spark applications can be developed in Python, which the author was already familiar with.

The basic architecture of a Spark application is similar to Figure 8. It consists of a master node (driver program in Spark terminology) and multiple worker nodes. The master node is the entry point for an application. It assigns sub-tasks to the worker nodes and retrieves the processing results afterwards. The input data for the application can be read by the master node and distributed to the worker nodes or the master node can assign the worker nodes a part of the data to process, which they then retrieve themselves.

A Spark application can read data from and write data to various types of data storage, such as databases, the local file system, or a distributed file system. Internally Spark processes data using different distributed data structures. These data structures influence the types of operations that can be executed on the data, as well as the efficiency of those operations. The two main data structures in Spark are Resilient Distributed Dataset (RDDs) and DataFrames. RDDs can store any kind of data and they support various operations, such as mapping elements in the RDD to new values or reducing the elements in the RDD by combining them. DataFrames are similar in structure to tables in a relational database. In DataFrames, a collection of data is organized into named columns and operations similar to SQL queries can be performed. RDDs and DataFrames can be written out to a distributed file

[2] https://spark.apache.org/

system as Pickle files and Parquet files respectively. These file formats allow a Spark application to easily read in the data structures at a later time.

In Figure 11, a Spark driver node and a cluster of Spark worker nodes is shown in the bottom center. The Spark application running on these machines reads in CDR/DDR-based mobility traces from a local or distributed file system as shown on the bottom left of the figure. It then classifies the mobility traces using various data sources and writes the results of the classification to output files. The following sections describe the structure and content of the used data sources.

4.1.1 Configuration file

The configuration file is a JSON file read by the Spark driver node. It provides the possibility to set which steps of the algorithm should be executed and it contains parameters used during the execution. The file is small and can be stored on the local file system of the driver node. An example of a configuration file is shown in Appendix A.

4.1.2 Mobility traces

CDR and DDR records in our dataset contain the information shown in Table 1. The first column contains a deviceId. This number is used to identify which device generated an event. The operator uses an International Mobile Subscriber Identity (IMSI) number for this purpose. The IMSI number uniquely identifies a device

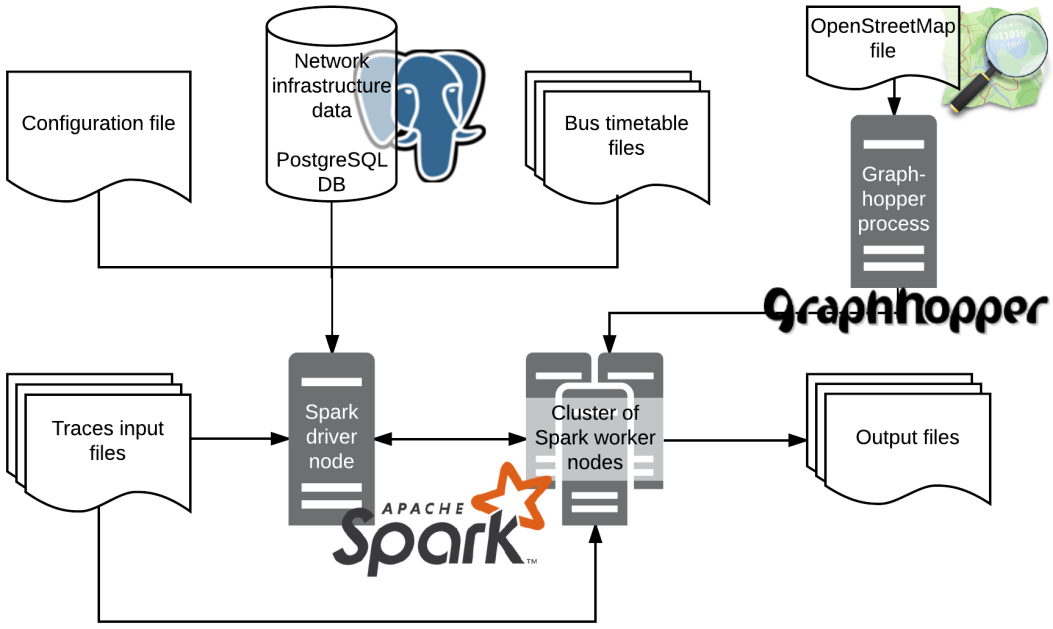


Figure 11: Software architecture overview

deviceId	cellId	areaId	ts	recordType
77631298440	320081	3266	1469421995	Location Update
77631298440	320081	3266	1469422998	Packet Gateway Record
77631298440	525102	3266	1469424798	Packet Gateway Record
77631298440	525102	3266	1469426598	Packet Gateway Record

Table 1: CDR/DDR records (dummy data)

globally and it is stored in a Subscriber identity module (SIM) card. IMSI numbers are privacy-sensitive, since they could be reduced to an individual SIM card and phone. Therefore, operators usually replace the IMSI number with a randomly generated number, the deviceId in this case. Operators periodically replace this random number with a new random number. This makes it harder to track a device for longer periods of time. However, as discussed in Section 2.1.5, researchers have shown that unique mobile phone users may still be identifiable in such a dataset. De-anonymized data could tell where a person was at different times and who they called or texted. Therefore, the machine on which the classification algorithm was developed was disconnected from the Internet. This is also the reason example data in this chapter is fabricated, although the structure is similar to that of the actual data.

The cellId column contains the identification number for cells in the operators network. The cellId is the identifier of the cell where a recorded event originated from. The areaId column contains the Local Area Code (LAC) for that cell. The cellId and areaId together uniquely identify a cell in the operator's network. The time of the event is recorded in the ts column as a Unix epoch time. Lastly, the recordType column shows the type of network event that was captured. A Location Update event is sent by a device when it enters a new LAC. This ensures that the network can quickly find the device if another device requests to connect with it. A Packet Gateway Record is captured when a new data usage session is started by the device, e.g., when it connects to the Internet.

From the CDR/DDR data, a data scientist extracts mobility traces. A mobility trace is an approximation of the route a mobile phone user traveled from one stationary point to another. Start and end points of traces could be detected by, e.g., checking if a user is stationary and whether the direction of the user's route changes drastically. For example, if a device is stationary for a certain amount of time, it can be seen as the end of a trace. Similarly, if a device drastically changes direction, this could be seen as the previous trace having ended and a new trace starting.

Table 2 shows the format of a mobility trace. Each mobility trace has a unique identifier stored in the id column. A trace is extracted from a sequence of CDRs/DDRs. Each of those CDRs/DDRs contains the identifier and LAC of the cell it was recorded in and the timestamp at which it was recorded. A network cell is uniquely identified by the combination of cell identifier and Local Area Code. The concatenation of the LAC and cell identifier gives a so-called cilac. Per trace, the cilacs column contains a sequence of cilacs corresponding to the consecutive cells a device generated CDR/DDR

id	cilacs	timestamps	device_id	start_cell	end_cell
0	['320081-3266', '119003-3325', '320081-8822', '119003-2188', '320081-5454', '119003-2188', '320081-5454', '320081-7645', '320081-7645', '320081-1128', '806342-1388', '806342-0092']	[1441605823, 1441606797, 1441606945, 1441606949, 1441607036, 1441607046, 1441607051, 1441607514, 1441607596, 1441607805, 1441608075, 1441609877]	77631298440	320081-3266	806342-0092

Table 2: Mobility traces table (dummy data)

data in. Similarly, the timestamps column contains a sequence of timestamps at which those records were recorded. The device_id column contains the random number generated by the operator that enable discovering which network events and mobility traces correspond to the same device. Lastly, the start_cell and end_cell columns contain the first and the last cilac from the cilacs column. With these columns, the first and last cell of a trace can be retrieved more quickly.

The mobility traces are stored in the column-separated values (CSV) file format. Spark can read CSV files and convert them to its internat DataFrame data structure using the Databricks spark-csv package[3]. The files with mobility traces are read from a file system accessible to the Spark driver and worker nodes. This could be a distributed file system supported by Spark. However, because of the earlier stated privacy concerns, the application was developed on a machine disconnected from the Internet. Therefore, the local (non-distributed) file system was used to store the CSV files with mobility traces.

4.1.3 Network infrastructure data

A German mobile network operator has provided cellular infrastructure data for the metropolitan area of Berlin, Germany. This data was stored in a table in a PostgreSQL[4] database. PostgreSQL is an open source object-relational database system that is well-supported and widely used. Furthermore, a PostgreSQL database can be extended with the PostGIS[5] spatial database extender. PostGIS adds support for querying geographical objects.

The infrastructure data includes information on mobile network cells, such as their unique identifier in the network, the broadcast technology and frequency it uses,

[3] https://github.com/databricks/spark-csv
[4] https://www.postgresql.org
[5] http://postgis.net

and the approximate coverage area of the cell. Table 3 shows a row corresponding to a single cell from the Berlin infrastructure table. The table shows dummy data, because the actual network infrastructure data is sensitive data for an operator. It can give a detailed view of an operator's network and as a result, harm their competitiveness.

The cell identifier $cell\_ci = 29070339$ and the location area code $cell\_lac = 1497$ together uniquely identify a cell within the network of a provider within one country. Combining this with the mobile country code (MCC) and the mobile network code (MNC) would give a globally unique identifier for this cell. Since our dataset only covers a single provider in one country, the combination of cell_ci with cell_lac is unique. The concatenation of these values is stored in the cilac column, so information corresponding to one cell can be quickly retrieved. The table contains a broadcast and frequency field to store the wireless communication standard and used frequency band of a cell respectively. Lastly, the table has a geom field, which contains the approximate coverage area for each cell. The coverage area is stored in the Extended Well-Known Text (EWKT) format. $SRID = 4236$ tells us that the coordinates should be interpreted in the spatial reference system corresponding to Spatial Reference System Identifier 4236. The multipolygon type can contain multiple polygons that together form the coverage area of a cell, although most cells cover an are that can be represented by a single polygon. Figure 12 provides a visual representation of the multi-polygon in the table.

cell_id	cell_lac	broadcast	frequency	cilac	geom
320081	3266	LTE	2800	320081-3266	SRID=4236; Multipolygon(((13.37262203430437602 52.50885965463277927, 13.37374791329762758 52.51134439579332991, 13.37831279976747467 52.51150567876850062, 13.37766767908402876 52.50729881524446085, 13.37262203430437602 52.50885965463277927)))

Table 3: Network infrastructure table (dummy data)

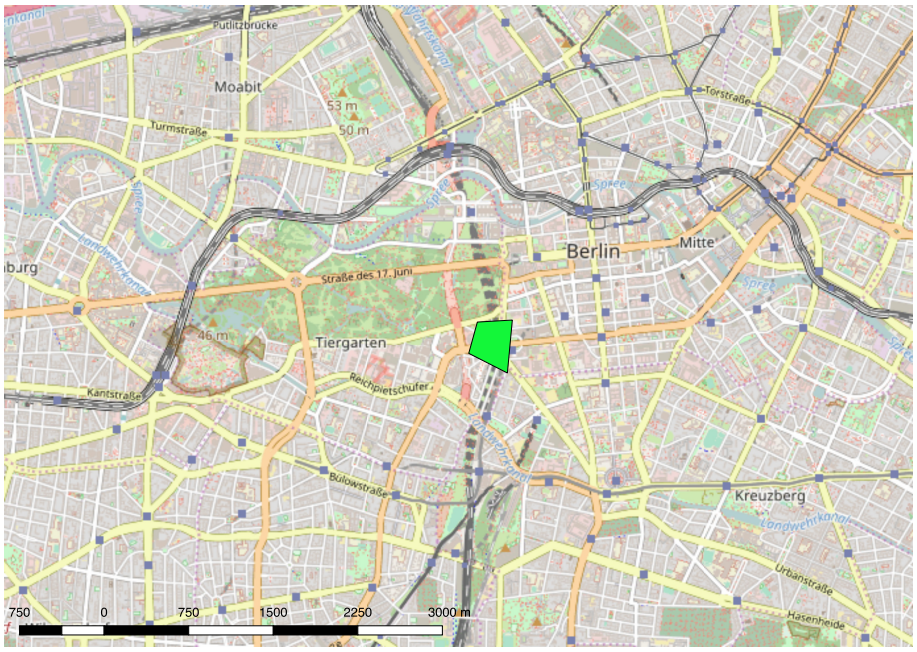


Figure 12: A multi-polygon corresponding to the dummy network cell in Table 3

4.1.4 Bus timetable data

The classification algorithm uses timetable data from public transportation companies. This timetable data is publicly available for Berlin[6]. The public transport information follows the General Transit Feed Specification (GTFS)[7]. This specification was developed by Google in 2006 to ease the integration of public transportation data in their Google Maps service. It has since then grown out to be the de facto standard for sharing transit data. GTFS specifies six files that are required, as well as optional files that may be present. The developed classification algorithm uses a subset of the required files and none of the optional files. In Table 4, the required files are described. The last column indicates whether the file is used by the algorithm. The contents of these files is discussed in later sections describing how they are used in the algorithm.

The bus timetable data concerns the scheduled bus arrival and departure times, not the actual times, which may differ due to delays. A dataset containing delay information would be preferred, but such a dataset is not available offline. Due to security constraints the algorithm was developed on a machine disconnected from the Internet.

In this document, the term bus station is used for the physical location where buses stop to let people enter or exit the bus. Stopover is used as the term for the moment a bus is scheduled to stop at a bus station. Lastly, for the sequence of bus stopovers a bus is scheduled to make, the term bus trip is used.

[6] http://www.europeandataportal.eu/data/dataset/vbb-fahrplandaten-april-bis-dezember-2016

[7] https://developers.google.com/transit/gtfs/reference/

Filename	File content description	Used
`agency.txt`	One or more transit agencies that provide the data in this feed.	No
`stops.txt`	Individual locations where vehicles pick up or drop off passengers.	Yes
`routes.txt`	Transit routes. A route is a group of trips that are displayed to riders as a single service.	No
`trips.txt`	Trips for each route. A trip is a sequence of two or more stops that occurs at a specific time.	Yes
`stop_times.txt`	Times that a vehicle arrives at and departs from individual stops for each trip.	Yes
`calendar.txt`	Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.	Yes

Table 4: Required files in the General Transit Feed Specification (file content description from [7])

4.1.5 Graphhopper and OpenStreetMap data

During the classification process, the Spark application requests routing information between certain points. It requests this information from a routing API. The development machine was not connected to the Internet for security reasons. Therefore, many online routing providers, e.g., Google Maps could not be used. Graphhopper[8] is an open source routing API under the Apache v2 License. It can therefore be run locally as required.

In order to calculate estimated travel times, Graphhopper makes use of locally downloaded OpenStreetMap[9] (OSM) data. OSM data provides a connected graph of roads and pathways. It includes information on the length of road segments, whether a path is one-way, or whether it is suitable for cars, cyclists, or pedestrians. Graphhopper can use this data to calculate the fastest route between two points for different transportation modes. The offline open-source version of the Graphhopper routing API supports routing for travel by car, bike, and foot.

4.2 Algorithm execution

The transportation mode classification algorithm has different steps that are executed sequentially. Although within one step, execution is distributed as much as possible. The steps of the algorithm are shown as a flow chart in Figure 13. The left side of the figure shows mobility trace files at the top and the eventual classification result files at the bottom. In between, the algorithm processes the traces in various steps. These steps use data from the data sources detailed in the previous section. However,

[8] https://graphhopper.com/
[9] https://www.openstreetmap.org

some processing has to happen on this data before it can be used to classify the traces.

The subsequent sections describe the steps of the classification algorithm. First, the initial data processing steps are described. Afterwards, the steps of the mobility trace filtering and classification are described. The steps are described in the order as indicated by the numbers in Figure 13.

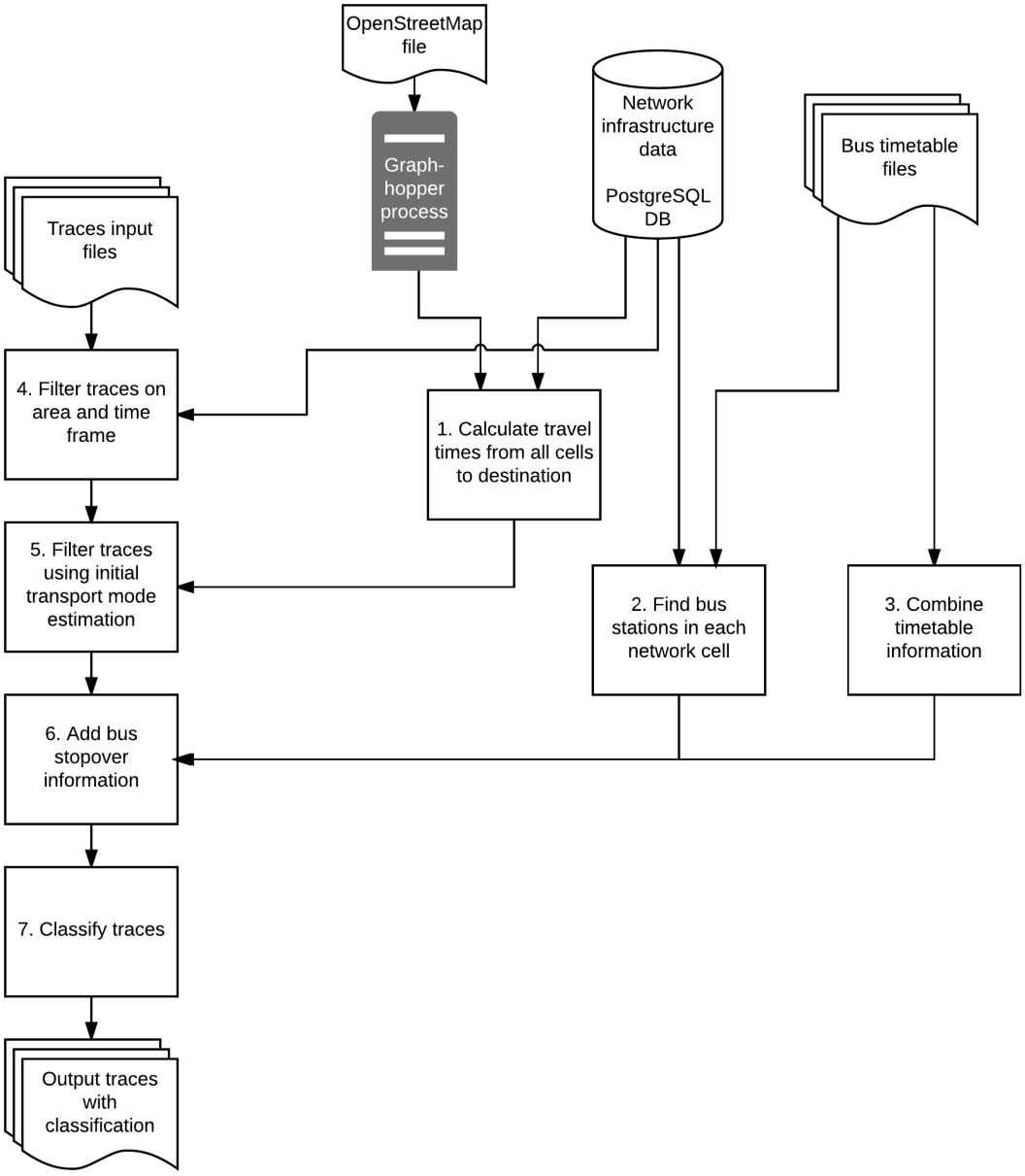


Figure 13: Algorithm execution flow

4.2.1 Calculate travel times

Mobility traces have a start cell and an end cell. Traveling from the start cell to the end cell using different modes of transportation usually results in different travel times. The expected travel times between the start cell and end cell of a trace can be calculated for different transportation modes. These expected travel times can then be compared to the duration of the mobility trace. This comparison yields the most likely transportation mode used by a mobile phone user while the trace data was being recorded. In step 5, mobility traces are filtered based on this principle.

There are many more input traces than there are network cells. Therefore, many traces will have the same origin and destination cells. Calculating the fastest travel times between the start and end cells of all traces individually would therefore lead to performing the same calculations many times. Instead, the expected travel time from all cells to all other cells should be calculated once for each mode of transportation. This information can then be stored in a way that facilitates quick look-ups for individual traces. This step of the algorithm prepares that data structure.

At the start of this step, the network infrastructure data is read in from a PostgreSQL database table. This data is partitioned based on cilac and distributed to worker nodes. For cell in the network infrastructure data, a worker node requests the travel time to all other network cells by car, bike, and foot. For the routing the centers of the cells are used. This introduces an error compared to the actual travel time, since a mobile phone user can be anywhere in a cell at the moment a CDR or DDR record is recorded. Figure 14 demonstrates this downside. If a mobile phone user travels from the border of a cell to the border of another cell, using cell centers may introduce a significant error. This is especially true for larger cells or in cases where cells have an oblong shape, such as the right cell in the figure. However, the cell center is straightforward to calculate and minimizes the average estimation error.

An example of the resulting DataFrame with the calculated travel times is shown in Table 5. The first column and first row contain cell identifiers. The values in row x, column y represent the travel times by car, bike, and foot from the cell in row x to the cell in column y. The results are written to the file system as a Parquet file.

Creating the DataFrame with estimated travel times from all cells to all other cells is a computationally expensive process. Fortunately, it only has to be recalculated if the network infrastructure changes or if significant changes in the road network occur. A data scientist can specify that only mobility traces ending at a specific area should be analyzed. In that case, this step will only calculate the travel times from all traces to that specified area. That reduces the computational complexity for this step from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ with n as the number of network cells.

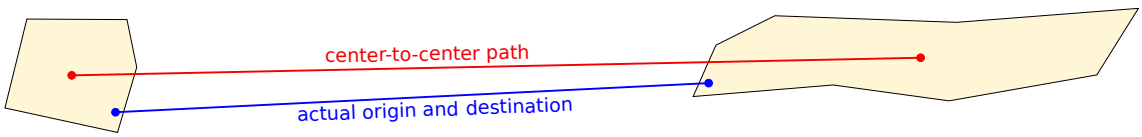


Figure 14: Estimation error due to using cell centers for travel time estimation

from cell \ to cell	320081-3266	320081-5454	806342-1388	...
320081-3266	–	(612,630,810)	(1439,3707,11391)	...
320081-5454	(566,1581,14932)	–	(1742,4012,11982)	...
806342-1388	(1230,2343,5622)	(820,246,707)	–	...
...	...	...	...	...

Table 5: Travel time look-up table traveling between cell centers by car, bike, or foot

stop_id	stop_name	stop_lat	stop_lon	...
9001103	Wiebestr./Huttenstr. (Berlin)	52.528318	13.32026	...
9001151	Reuchlinstr. (Berlin)	52.527903	13.323637	...
9001152	Neues Ufer (Berlin)	52.529103	13.315981	...
9001154	Ilsenburger Str. (Berlin)	52.525756	13.30984	...
9001155	Goslarer Platz (Berlin)	52.525797	13.314261	...

Table 6: Subset of columns from a GTFS `stops.txt` file for Berlin

4.2.2 Find bus stations per network cell

In step 6 of the classification algorithm, the application checks if CDRs or DDRs were recorded near a bus station, around the time that a bus stopped there. This step requires finding all bus stations contained in a network cell. Many network cell identifiers occur in multiple traces. Therefore, a data structure is created that enables quickly finding all bus stations in a certain network cell based on a cell identifier.

As discussed in Section 4.1.4, timetable data is stored in several text files according to the GTFS specification. The `stops.txt` file is formatted as a CSV file. Table 6 shows a subset of the columns contained in the file. The file contains three relevant columns: stop_id, stop_lat, and stop_lon. The former contains the unique identifier for each bus station and the latter two contain the coordinates of the bus station's location.

At the start of this step, the network infrastructure table is partitioned and distributed over the worker nodes. Every worker node also receives a copy of the `stops.txt` files from the driver node. Per network cell, the worker nodes process all bus stations individually. First, the multi-polygon representing the network cell's coverage area is read in using the following code:

```
cell = wkt.loads(cell_row.geom)
```

Here, cell_row corresponds to one row in the network infrastructure table. The Shapely[10] python library processes geometric objects. The wkt module of the Shapely library is used to load the coverage area geometry, which is in the Well-Known Text format (WKT). Afterwards, for each bus station, Shapely's geometry module is used to load the bus station's location as a point.

[10] https://pypi.python.org/pypi/Shapely

trip_id	arrival_time	departure_time	stop_id	stop_sequence	...
1	04:45:00	04:45:00	9230999	1	...
1	04:51:00	04:51:00	9230400	2	...
1	04:59:00	04:59:00	9220019	3	...
1	05:06:00	05:06:00	9220070	4	...
1	05:11:00	05:11:00	9220114	5	...
1	05:13:00	05:13:00	9220001	6	...
1	05:32:00	05:32:00	9260024	7	...
2	05:35:00	05:35:00	9260024	1	...
2	05:54:00	05:54:00	9220001	2	...

Table 7: Subset of columns from a GTFS `stop_times.txt` file for Berlin

service_id	monday	tuesday	wednesday	thursday	friday	saturday	sunday	...
000001	1	1	1	1	1	0	0	...
000002	1	1	1	1	1	0	0	...
000003	1	1	1	1	1	0	0	...
000004	1	1	1	1	1	0	0	...
000005	1	1	1	1	1	0	0	...
000006	0	0	0	0	0	1	1	...
000007	0	0	0	0	0	1	1	...

Table 8: Subset of columns from a GTFS `calendar.txt` file for Berlin

```
stop = Point(stop_row['stop_lon'], stop_row['stop_lat'])
```

Finally, Shapely tests whether the bus station is contained in the cell coverage area using the `intersects` method:

```
cell.intersects(stop)
```

If the bus station is contained in the cell coverage area, it is added to a list. The result from this data preparation step is a key-value RDD with cell identifiers as keys and lists of bus stations identifiers as values. This RDD is stored as a PickleFile on the file system. Finding all bus stations contained in all network cells is computationally expensive. The computational complexity is $\mathcal{O}(mn)$ where m is the number of network cells and n the number of bus stations. Fortunately, it has to be recomputed only when either the network infrastructure data changes or the bus timetable data changes.

4.2.3 Combine bus timetable data

Relevant bus timetable information is stored in different files following the GTFS format. The `stops.txt` file was used in the previous step to find the bus station identifiers corresponding to all bus stations in each network cell. The other relevant

route_id	service_id	trip_id	trip_headsign	...
1	000095	1	Flughafen Schönefeld Terminal (Airport)	...
1	000095	2	S Potsdam Hauptbahnhof	...
2	000114	3	Golzow (PM), Schule	...
2	000114	4	Golzow (PM), Schule	...
2	000114	5	Golzow (PM), Schule	...
2	000114	6	Lehnin, Busbahnhof	...
2	000114	7	Lehnin, Busbahnhof	...
2	000114	8	Lehnin, Busbahnhof	...
3	000096	9	Groß Kreutz, Bahnhof	...
3	000116	10	Groß Kreutz, Bahnhof	...

Table 9: Subset of columns from a GTFS `trips.txt` file for Berlin

bus timetable files are combined to more easily and efficiently retrieve all stopover events per bus station. Tables 7, 8, and 9 show parts of the `stop_times.txt`, `calendar.txt`, and `trips.txt` files respectively.

The trip_id in `stop_times.txt` is the unique identifier for a bus driving a specific route at a specific time, e.g., bus 200 leaving the central station at 12:03. A bus 200 driving at another time will have a different trip_id. Per trip_id, `stop_times.txt` contains the stopovers. Each stopover has a sequence number, an arrival time, and a departure time. There is also a stop_id uniquely identifying the bus station at which the stopover takes place. In `calendar.txt` the days of the week are stored on which a bus service is active. If bus 200 has a different schedule on weekends, it will have one service_id for the schedule during weekdays and a separate service_id for its schedule during the weekend. A stopover in the `stop_times.txt` file only happens on days that the corresponding trip_id is scheduled. To figure out if a bus has a stopover on a specific date at a specific time, the calendar and stop_times files have to be combined. The `trips.txt` file is used for that purpose. It shows which trip_ids correspond to which service_ids.

In the Spark application, the files are read in to Spark DataFrames using the spark-csv module. The following line reads in the stop_times file:

```
stop_times =
 ↪  sqlc.read.format('com.databricks.spark.csv').options(header='true',
 ↪  inferschema='true').load(bus_input_dir + 'stop_times.txt')
```

This tells Spark to load the `stop_times.txt` file using the `com.databricks.spark.csv` module. The directory where the file is stored can be configured and is stored in the `bus_input_dir` variable. The file has a header row with column names and spark should infer the data types for the columns, so the options `header` and `inferschema` are set to `true`. Next, the three DataFrames are joined together in two separate joins. First, the trips DataFrame is joined with the calendar DataFrame on the service_id column:

```
traces_with_days = bus_trips.join(calendar, bus_trips.service_id ==
↪  calendar.service_id)
```

Then the result of that join is combined with the stop_times DataFrame by a join on the trip_id column:

```
stop_times.join(traces_with_days, stop_times.trip_id ==
↪  traces_with_days.trip_id)
```

Eventually, arrival times and departure times of buses will be compared to timestamps. A timestamp depicts an exact time on a specific day, whereas the arrival and departure times are given as times relative to any day the trip is active on. Therefore, the time of day is compared separately from the day of the week. When the comparison is made, the day of the week is read from a timestamp and compared with the active days of the week for a stopover. The date of the timestamp is then changed to January 1 1970, leaving the time of day intact. Consecutively, this timestamp is compared to the arrival and departure time for a stopover. The arrival and departure times are therefore stored as timestamps relative to January 1 1970 as well.

The resulting DataFrame is written to the file system as a Parquet file. This step only has to be re-executed if the bus timetable data changes. Table 10 gives an overview of the resulting DataFrame with the combined bus timetable data.

4.2.4 Filter on geographical area and time frame

After the previous three data preparation steps, the algorithm starts processing the mobility traces. A data scientist can specify in the configuration file that mobility traces should be filtered based on a geographical area. They could specify that only traces *passing through* a specific area should be analyzed. Alternatively, they could specify to only analyze traces that *end at* a specific area. This filter can be combined with a time constraint for when a trace has to pass through —or end at— the specified area. This allows the user to answer questions, such as "How many

trip_id	stop_id	arrival_time	departure_time	mon	tue	...
31	U1071Z101	1970-01-01 07:07:10	1970-01-01 07:07:30	1	1	...
31	U953Z102	1970-01-01 07:09:25	1970-01-01 07:09:55	1	1	...
31	U713Z102	1970-01-01 07:12:00	1970-01-01 07:12:30	1	1	...
31	U921Z102	1970-01-01 07:14:00	1970-01-01 07:14:30	1	1	...
31	U118Z102	1970-01-01 07:15:45	1970-01-01 07:16:05	1	1	...
31	U209Z102	1970-01-01 07:17:15	1970-01-01 07:17:35	1	1	...
31	U476Z102	1970-01-01 07:18:45	1970-01-01 07:19:05	1	1	...
31	U400Z102	1970-01-01 07:20:15	1970-01-01 07:20:45	1	1	...
31	U1072Z102	1970-01-01 07:21:45	1970-01-01 07:22:15	0	0	...
31	U703Z102	1970-01-01 07:23:20	1970-01-01 07:23:40	0	0	...

Table 10: Selected columns from combined bus timetable data DataFrame

people traveled by bus to supermarket X between 8:00 and 19:00 today?". In the configuration file, the data scientist can specify the time frame within which traces need to be in the specified area. The filter area is specified in the widely-used Well Known Text (WKT) format, e.g.:

```
POLYGON ((14.42562 50.08981, 14.42901 50.08981, 14.42901 50.08826,
 ↪  14.42562 50.08826, 14.42562 50.08981))
```

Mobility traces have a sequence of cell identifiers that present the consecutive network cells in which a device generated a CDR or DDR. To discover if a trace overlaps with a geographical area, the geographical area is mapped to a list of cell identifiers. These cell identifiers correspond to the network cells that (partially) overlap with the target area. This is visualized in Figure 15. In the figure, five cells partially or completely overlap the target area and two cells do not. The cells corresponding to the target area are 1001, 1002, 1003, 1005, 1006. A mobile phone user could travel through cell 1001 in the example figure without actually passing through the target area. Including this cell in the filter, means that some traces area included that should not be. An alternative would be to only include cells that share at least half their area with the target area. The function used to find the cell identifiers of the cells overlapping the target area is as follows:

```
def get_overlapping_cells(cells_df, polygon):
filtered_df = cells_df.filter(udf(lambda shape_wkb:
 ↪  polygon.intersects(wkt.loads(shape_wkt)), BooleanType())('geom'))
return filtered_df.map(lambda row: row.cilac).collect()
```

The function `get_overlapping_cells` takes two arguments, a DataFrame containing the network infrastructure data and a polygon representing the target area. The cells DataFrame is filtered based on a user-defined function (UDF). This user-defined function is applied to the `geom` column of the cells DataFrame, which represents the cell coverage area. The function is used as a filter and thus it returns a Boolean. The `geom` data is called `shape_wkt` inside the UDF. It is read as a geometry object using the Shapely wkt module. Finally, Shapely is used to test if

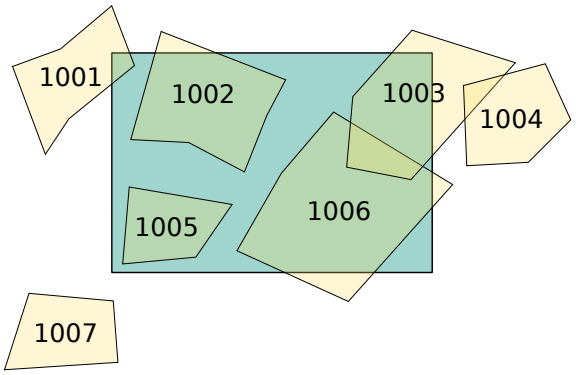


Figure 15: Target area (green) with network cell coverage areas (yellow)

the target area and the cell coverage area intersect. If they do, the UDF returns true. Otherwise, it returns false, and that row is not returned from the filter function. After the filter, `filtered_df` only contains information on cells that overlap with the target polygon. The last row returns from each row only the unique cell identifier (`cilac`) and collects these as a list.

After creating the list of target cell identifiers, the Spark application partitions and distributes the mobility traces over the worker nodes. This partitioning happens automatically when when an operation is executed on a dataset, although the number of partitions can be changed manually. If the configuration file specifies that all traces passing through the target area should be analyzed, the Spark worker nodes check if any of the cell identifiers in a mobility trace is contained in the set of target cell identifiers. If the configuration file specifies that only traces ending at the target area should be analyzed, the worker nodes only check if the last cell identifier of each trace is contained in the set of target cell identifiers. The difference in traces ending at a target area and traces passing through a target area is visualized in Figure 16. Trace 1 does not touch the target area and is therefore filtered out in both cases. Trace 2 ends at the target area and is therefore not filtered out in either case, since ending at the area is seen as an instance of "passing through" an area. After all, the mobility trace touches the area at some point. Trace 3 is passes through the target area, although it does not end there. Therefore it is included only if the filter is set to include traces passing through the area.

Lastly, if a filter time frame was specified, the classification algorithm filters on traces ending or passing through the target area *within the specified time frame*. For example, assume the configuration file is set to filter on traces passing through the

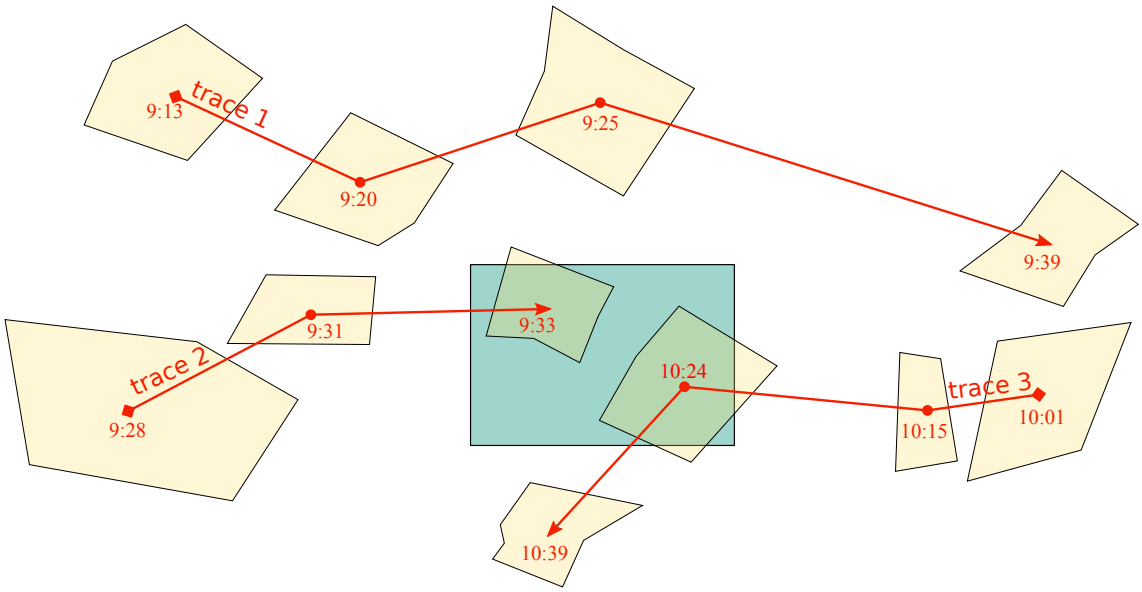


Figure 16: Traces (red) passing through, ending at, or not touching a target area (green)

target area between 9:00 and 10:00. Then, trace 2 in Figure 16 would pass the filter. However, trace 3 would be filtered out, because it passes through the target area at 10:24, well after the filter time.

4.2.5 Filter on initial transportation mode

In this step of the classification algorithm, mobility traces are excluded from further analysis if they same unlikely to correspond to bus traces based on their travel time. For each mobility trace, the first and the last cell identifier are retrieved. The expected travel times from the center of the first cell to the center of the last cell using different transport modes are then retrieved from the DataFrame created in step 1. The first timestamp for the trace is subtracted from the last timestamp to obtain the actual time the mobile phone user took to travel from the start cell to the end cell. However, as discussed previously, detecting the start and end of a trace is not completely accurate. Therefore, this travel time may not correspond to the actual trave time of the user.

Comparing the travel time based on the mobility trace to the expected travel times for different transportation modes, gives a transportation mode with a travel time closest to the trace duration. This transportation mode is most likely the one taken when the mobility trace was recorded. Figure 17 shows a mobility trace with the travel times for different travel modes. The travel times are for a travel from the center of the first cell to the center of the last cell of the trace. In this case, the actual travel time of 26 minutes is closest to the travel time by bike, 22 minutes. Therefore, this trace is initially classified as a bike trace.

In the configuration file, there is an option to set which initial travel mode classifications to filter out. A travel time similar to a car might indicate that a trace was not traveled by bus, since cars usually travel significantly faster, especially on longer travels. On shorter travels, the travel time by bus is very similar to the travel time by car. With this option set, those traces are filtered out. Thus, this filter makes the classification of shorter trips less accurate. However, a data scientist could alter this option to exclude traces initially classified as other travel modes if that leads to more accurate results.

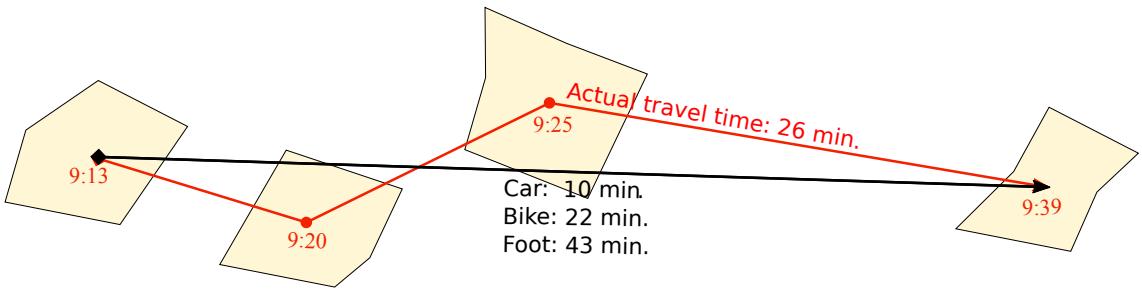


Figure 17: Travel times using different travel modes from the center of the first cell to the center of the last cell of a mobility trace

4.2.6 Add bus stopover information

Figure 18 shows a mobility trace in red and two bus lines in black, labeled 101 and 201. The trace has its first record in cell A at 8:05. Using the key–value RDD created in step 2 (Section 4.2.2), the classification algorithm retrieves all bus stations contained in cell A. In this case only one bus station is found. All stopover events for this bus station are retrieved from the DataFrame created in step 3 of the algorithm (Section 10). In this simplified case, there is only one stopover at 8:04 made by bus 201. Although, in reality, a bus line would stop at a bus station multiple times per day. The stopover happened in the cell where the mobility trace had a record, around the time that the record was registered. Therefore, the trace "matches" bus 201 in cell A. The trace additionally matches bus 201 in cell B and in cell C. However, it does not match bus 101 in cell C, because the stopover of bus 101 is more than two hours after the record for the trace in that cell. The algorithm stores information about all matches for each trace.

All Spark worker nodes need a copy of the key–value RDD mapping cell identifiers to identifiers for bus stations in that cell. They furthermore need a copy of the combined bus timetable DataFrame. These data structures are made available to all worker nodes using the following code:

```
cilac_stops = sc.broadcast(cilac_stops.collectAsMap())
stop_info_map = sc.broadcast(stop_info_map)
```

With the `broadcast` function of SparkContext, Spark can be told that a certain data structure should be available to all worker nodes. The key–value RDD cilac_stops is converted into a Python dictionary before being broadcast using the `collectAsMap` method.

The trace in Figure 18 has a cilacs column containing the sequence [A, B, C, D]. The `cilac_stops` dictionary created above, can be used to find the two stop_ids

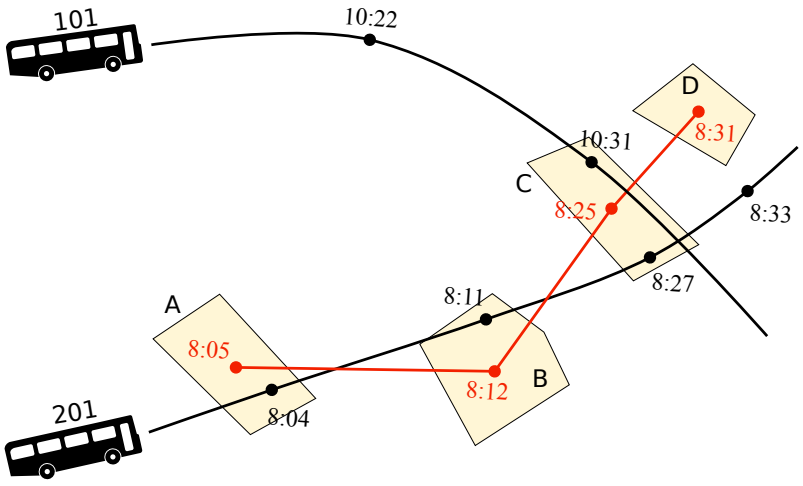


Figure 18: A mobility trace (red), two bus lines (black), and four cell coverage areas labeled A–D (yellow)

corresponding to the two bus stations in cell C. For each of those stop_ids, the stopover times are found in `stop_info_map`. At this point, the list of timestamps of the mobility trace becomes relevant. For this trace, the timestamps column would be [8:05, 8:12, 8:25, 8:31]. (In reality, these times would be timestamps; this is simplified for the example.)

If the timestamp from a mobility trace has to strictly be between a bus arrival and departure times, some bus traces would be misclassified as non-bus traces. Assume the following scenario: a mobile phone user is on a bus entering a cell and record is created in the operator's system at the moment the user enters the cell. However, the bus station the user is traveling to is at the other side of the cell. Therefore, the recorded timestamp for the mobility trace is earlier than the arrival time of the bus at that bus station, even though the user was on that bus. Similarly, a bus could be early or delayed compared to the timetable. To account for these situations, a time margin can be set to accommodate for this timestamp inaccuracy and for bus delays.

Per trace, information on the stopovers and buses that are matched by a trace are stored. For this purpose, a column is added to the Spark DataFrame with the mobility traces using the `withColumn` function.

4.2.7 Classification

The last step of the algorithm is the classification of traces as being (partially) traveled by bus or not. Two metrics are used to determine the likelihood of a trace being a bus trace. Firstly, a trace that matches the same bus more often is deemed more likely to be a bus trace. A mobile phone user being in different cells around the same time that the bus is in those cells, is a strong indication that the user is on that bus. The second metric for a trace being classified as a bus trace is the time between the first and last time that the trace matches a bus. If a user matches the same bus at least twice with considerable time in between the first and last match, it is likely that the user traveled on that bus. A mobile phone user could pass by a bus station around the time that a bus leaves from that station. The user could then later pass by a bus station on the route of that same bus. However, if this second station is far removed from the first, it is unlikely that the user arrives at that station at the same time as the bus, unless the user is on that bus.

These two metrics are tested independently. Therefore, a trace is considered to be traveled by bus if either a) the trace matches a bus trip at least a specified number of times, or b) the trace matches a bus trip at least twice with at least a certain amount of time in between the first and the last match. Both the minimum number of matches and the minimum amount of time between the first and the last match can be specified in the configuration file.

Two mobility traces are shown in Figure 19. If the acceptable time margin between a trace and a bus stopover event is set to 2 minutes or more, trace 1 matches bus 301 four times. In that case, trace 2 matches bus 401 two times. Assume that the minimum number of matches for a trace to be classified as a bus trace is 3 and that the minimum time between the first and last bus match is set to 30 minutes. Then, trace 1 is classified as a bus trace because it matches one bus more than 3

times. Trace 2 matches the same bus only twice. However, the time between matches is more than 30 minutes (from 10:01 to 10:39). Thus, both traces are classified as bus traces in this scenario.

Finally, all traces with their classification are written to the file system as a Parquet file. If the number of traces is low, the output is written to a CSV file as well to allow convenient further analysis. The software writes a brief analysis summary to the standard output containing the total number of trips after applying the filters and the number of those trips classified as bus trips.

4.3 Usage

The algorithm can be run on a machine with Ubuntu 16.04. On the machine, Python 3 should be installed as well as the Shapely python library. Java 1.8 and Apache Spark 1.6.2 are also required. The machine should be able to connect to a PostgreSQL 9.5 database with a network infrastructure table using the PostgreSQL JDBC driver (9.4.1208 JDBC 42). All parameters and options for the algorithm can be set in the configuration file. An example configuration is given in Appendix A. In Appendix B, the different options are explained.

After setting up the machine and adjusting the configuration file, the program can be run from the terminal in the project root folder using the following command:

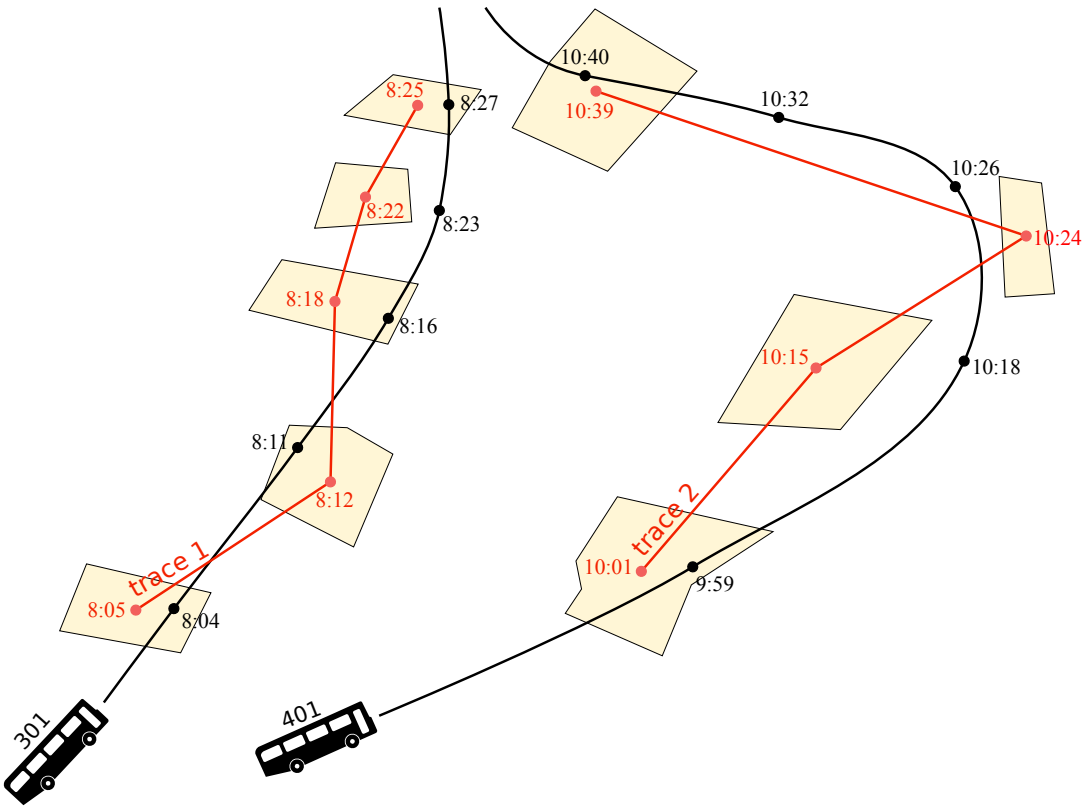


Figure 19: Two mobility traces (red), two bus lines (black), and network cells (yellow)

```
spark-submit --packages com.databricks:spark-csv_2.10:1.4.0
↪ --driver-class-path /home/user/posgresqlJDBC/driver-v.jar
↪ main.py
```

This submits a new job to Spark. It specifies that the Databricks spark-csv package is used. This package will be downloaded automatically if it does not exist on the system. The `-driver -class-path` option is used to tell Spark where the PostgreSQL JDBC driver file can be found. Lastly, `main.py` is the file containing the Spark application.

Some steps in the algorithm write intermediate files to the output directory, such the first three data processing steps. They create data structures containing estimated travel times, bus stations per network cell, and combined timetable information respectively. The algorithm automatically reuses these files if present in the output directory and if the configuration file states that those data preparation steps should not be executed. This speeds up consecutive runs on the same coverage area with different input traces.

Steps 4, 5, 6, and 7 all alter a Spark DataFrame containing the mobility traces. They filter traces from this data structure and add information to it. After step 8, this data structure is written to the file system as the algorithm output. However, it is possible to execute only a subset of the steps of the algorithm. In this case the result after those steps is written to the output file.

5 Evaluation

In this chapter, the evaluation of the developed transportation mode classification algorithm is presented. The aim is to give a realistic measure for the algorithm's accuracy in classifying the travel mode of mobility traces. This analysis requires a set of mobility traces for which the corresponding transportation mode is known. The term *evaluation trip* is used for the GPS trace recorded on an evaluation phone. For each evaluation trip, the mode of transportation is recorded. The term *mobility trace* is used for the trace that is extracted from Call Detail Records (CDRs) and Data Detail Records (DDRs). The distributed data analysis algorithm identifies the travel mode of mobility traces. That classification is then compared to the evaluation trips recorded on the phones to determine if the classification is correct.

The setup for collecting the ground truth data is described in Section 5.1. The setup describes the location, time and duration of the data collection, as well as the kind of evaluation trips that were made. Section 5.2 presents the data collected during the evaluation, as well as a characterization of the mobility traces extracted from the collected data. Lastly, Section 5.3 presents a two-fold cross-validation analysis based on the evaluation data.

5.1 Setup

The evaluation data was collected in Berlin, Germany during a one week time span in November 2016. The partnered mobile network operator made four Android phones available for the evaluation. These phones each had a white-listed SIM card, which means that the IMSI numbers of those SIM cards are tagged by the provider to exclude them from the anonymization process that normally follows data collection. This enables matching the mobility traces extracted from the operator's billing data to the actual evaluation trips that were recorded on the phones.

Two volunteers made evaluation trips in Berlin, each carrying two of the evaluation phones. The trips had Zoologischer Garten as either their origin or their destination. Zoologischer Garten was chosen as the focal point, because it is very central in west Berlin and has bus connections in all directions. Trips were made by car and by bus. From all available travel modes, the car is assumed to give a travel route and travel time most similar to traveling by bus. The difference in travel time and travel route between traveling by bus and traveling by foot, bike, metro, tram, or train is assumed to be larger. Therefore, if the algorithm can accurately differentiate between bus traces and car traces, the algorithm's accuracy is only expected to be higher when differentiating between bus traces and non-bus traces in general.

For bus trips starting at Zoologischer Garten, the end points were chosen to be bus stations that are reachable via a direct bus connection and for which the travel time is close to 10, 30, or 50 minutes according to the bus timetable. These bus stations would then be the origins for the trips in the opposite direction, ending at Zoologischer Garten. The car trips have the same origins and destinations as the bus trips, which means that their travel time is often shorter. If a car trip can start at an arbitrary moment, it is possible that the car trip does not overlap with a bus driving

in the same direction. This is especially true for destinations with less frequent bus connections. This makes classifying the car trip almost trivial, whereas the more challenging scenario would be to classify a car trip that occurs while a bus is driving in the same direction. Therefore, the decision was made to start car trips at the moment that a bus leaves for the same destination. Since car trips do not necessarily follow the same routes as buses and since their travel times can differ, differentiating between a bus trip and a non-bus trip could still be possible.

In order for the mobility trace extraction to accurately identify when a trip starts and ends, two considerations were taken into account during the evaluation trips. Firstly, there needs to be a period in which the phone is stationary both before and after the trip. If this period is too short, consecutive trips may be recognized as a single trace. Therefore, the volunteers were stationary at the origin and destination for at least 20 minutes before and after every trip. The second consideration is that an adequate number of network events need to occur for the extracted mobility trace to be sufficiently precise. The mobile provider collected call, text, data, and signaling data for the evaluation. Therefore, the phones were used for browsing the web and texting during bus trips and as navigation devices using Google Maps during car trips. Both web browsing and online navigation with real-time traffic information generate a sufficient number of network data events, while also being realistic real-world scenarios.

All evaluation trips were documented with pen and paper, and by using the Open GPS Tracker Android app[11]. For every trip the same unique identifier was used both in the Open GPS Tracker app and on the paper form. On paper, the start time, bus line, bus delay compared to timetable, and used phone were recorded. The Open GPS Tracker app was turned on at the start of every trip and off at the end of every trip. The GPS data recorded by the app was exported in the GPS exchange format (GPX). The steps for one evaluation trip are listed below for both a bus trip and a car trip. The steps assume the trip starts at Zoologischer Garten and after the trip, the same steps could be followed to travel back.

Procedure for a bus evaluation trip:

0. Be stationary for at least 20 minutes.
    - Browse web pages or use apps using the phone's Internet connection.
1. Board a bus at Zoologischer Garten towards a predetermined bus station.
    - Start tracking on the Open GPS Tracker app.
    - Write the trip information down on paper.
2. Browse web pages or use apps using the phone's Internet connection.
3. Exit the bus at the predetermined bus station, after 10, 30, or 50 minutes of traveling.

[11] https://play.google.com/store/apps/details?id=nl.sogeti.android.gpstracker

- Stop tracking on the Open GPS Tracker app.

4. Be stationary for at least 20 minutes.
    - Browse web pages or use apps using the phone's Internet connection.

Procedure for a car evaluation trip:

0. Be stationary for at least 20 minutes.
    - Browse web pages or use apps using the phone's Internet connection.
1. Leave by car from Zoologischer Garten towards a predetermined bus station at the time a bus leaves for the same destination.
    - Start tracking on the Open GPS Tracker app.
    - Write the trip information down on paper.
2. Use the phone's Internet for navigation.
3. Find a parking spot near the predetermined bus station.
    - Stop tracking on the Open GPS Tracker app.
4. Be stationary for at least 20 minutes.
    - Browse web pages or use apps using the phone's Internet connection.

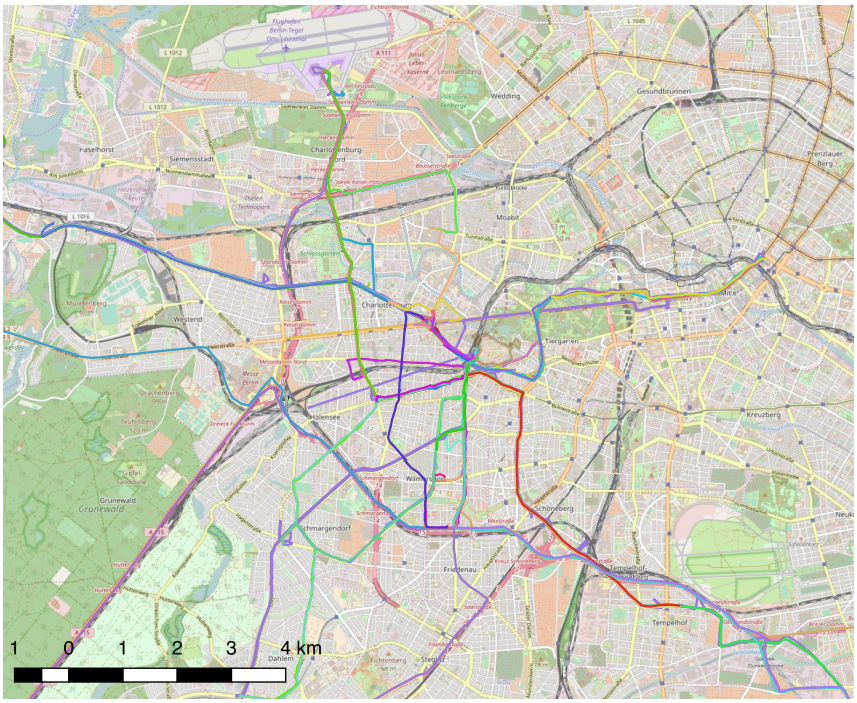


Figure 20: Evaluation trips in Berlin

5.2 Collected data

Two sets of data result from the evaluation trips made using the previously described setup. The first set is the data recorded by the volunteers on paper forms and using the Open GPS Tracker app on the phones. The GPS data recorded on those phones consists of frequent data points that contain an accurate location and the time at which the location was recorded. The associated notes register for each trip which transportation mode was used for that trip. During the evaluation, 196 evaluation trips were made. Information on these trips was recorded on paper and one GPX file was recorded per trip. For the evaluation, a trip is counted as one phone traveling on a specific path, so a volunteer traveling with two phones records two trips simultaneously.

An overview of all recorded trips is shown in Figure 20. Individual trips are colored differently as much as possible. Many trips in the image overlap. This has three causes. Firstly, the trip to a destination and the trip back to the origin often follow the same route. Secondly, in the evaluation setup it was decided to travel with the same bus line to a bus station at 10, 30, and 50 minutes from Zoologischer Garten. Since a bus line usually follows the same route, the first parts of these routes will overlap. Lastly, the car trips have the same origins and destinations as the bus routes. Although the car trips may travel via different routes in between origin and destination, these trips will partially overlap with the bus trips.

The second data set consists of the mobility traces that were extracted from the network events collected by the network operator. This second data set is the input for the transportation mode classification algorithm. 173 mobility traces were

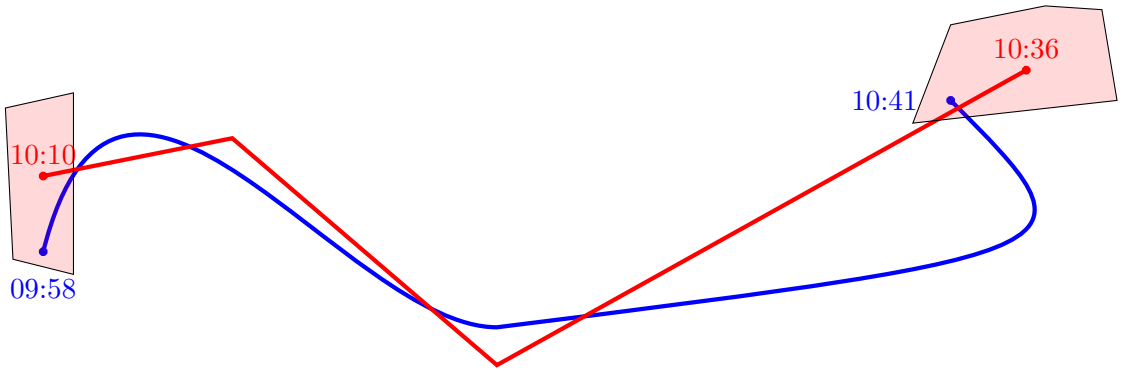


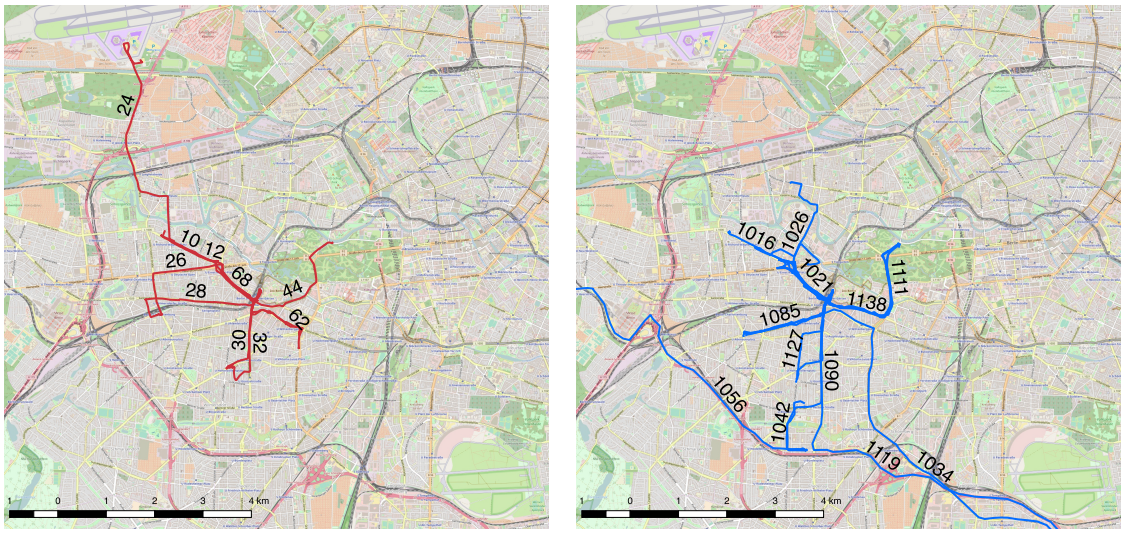
Figure 21: Visualization of an extracted mobility trace (red) with the corresponding GPS trip (blue)

extracted from the collected network events. The mobility traces extracted from the CDR and DDR data do not match the actual evaluation trips perfectly. Multiple evaluation trips may be extracted as a single mobility trace, an evaluation trip may be captured in multiple mobility traces, or a mobility trace may not match any evaluation trip. If no matching evaluation trip can be found for a mobility trace, the travel mode for that trace is unknown. In that case, it has to be excluded from the evaluation analysis.

A Spark application was written in Python to associate the mobility traces with the correct evaluation trips. This association happens based on three criteria. Firstly, the mobility trace should be extracted from the data corresponding to the same phone the trip data was recorded with. Secondly, the start and end location of the trace should be similar to the start and end location of the trip. This means that the GPS points representing the start and end points of the evaluation trip should be contained within the coverage area of the first and last network cell of the mobility trace respectively. Lastly, the start and end time of the trace should be close to the start and end time of the trip. In the evaluation data, having the trace start and end times match the trip start and end times with a 30 minutes margin results in an accurate matching.

This procedure for associating traces with trips is visualized in Figure 21. In this figure, the blue line represents the GPS data recorded directly on an evaluation phone. It gives an accurate representation of the actually traveled path. The red line represents the extracted mobility trace. The red line connect the center points of the recorded network cells. The coverage area for first and last cells in the trace are shown as red polygons. The trip and the trace match, because a) the trip starts in the coverage area of the first cell of the trace and it ends in the coverage area of the last cell, b) the trace start and end times are within half an hour of the trip start and end times respectively, and c) the trip and trace data belong to the same phone.

Using the procedure for associating trips with traces described above, 89 of the 196 evaluation trips could be matched with 89 of the 176 mobility traces. Figure 22 presents an overview of the trips for which a matching trace could *not* be found.



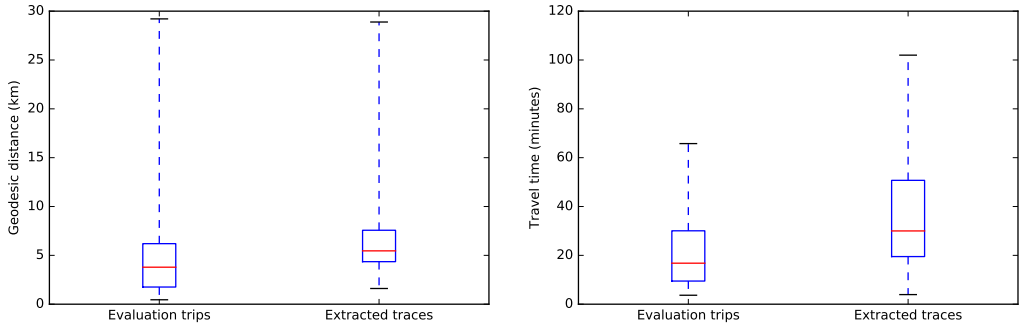
(a) Phone A (b) Phone B

Figure 22: Evaluation trips for which no associated trace was found

Comparing the figure with Figure 20, shows that it is mostly shorter trips that can not be matched with a trace. Notable exceptions to this observations are trip 24 to Tegel airport in sub-Figure 22(a) and trips 1034 and 1056 in sub-figure 22(b). Trip 24 could not be matched, since no trace was recorded around that time which ended at a cell near Tegel airport. Trips 1034 and 1056 were longer trips with long stationary periods following the trip. As a result, the trace extraction algorithm recorded the end times of the traces corresponding to those trips more than 30 minutes after the trips had ended. Therefore, the Spark script did not associate the trips with those traces. The general case for unassociated trips is that they cover a short distance, i.e., they are within a 2.5 kilometer radius from Zoologischer Garten. With such short distances and travel times, distinguishing between noise and signal in the trace extraction algorithm is less accurate.

Further insight in the differences between evaluation trips and the traces resulting therefrom is presented in Figure 23. It shows that 50% of the evaluation trips that were actually made, cover a geodesic distance between 1700 and 6200 meters, whereas for the mobility traces this is between 4300 and 7600 meters. Similarly, 50% of trips have a travel time between 9 and 30 minutes, whereas for traces this is in between 19 and 50 minutes. This agrees with the earlier analysis that longer trips are generally easier to associate with the corresponding mobility traces.

The last metric for the extracted mobility traces presented here, is the number of data points per trace. One data point corresponds to a cell identifier and a timestamp for when a CDR or DDR was generated. In the example trace visualized as a red line in Figure 21, each point at which the line changes direction, corresponds to a data point. The traveler's location is not known in between data points, thus the larger the time and distance between two points, the higher the uncertainty about the traveler's location in between. A trace with more data points generally provides a



(a) Distance between start and end point (b) Travel time between start and end point

Figure 23: Box plots showing the distribution of geodesic distance and travel time for evaluation trips and the extracted mobility traces

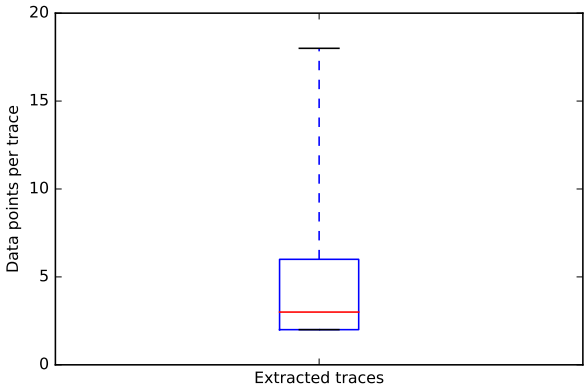


Figure 24: Box plot showing the distribution for the number of data points per mobility trace

more accurate approximation of the actual path traveled by a user. Figure 24 shows the distribution of this metric for the traces that could be matched to evaluation trips. 50% of traces have 2 to 6 data points and the halfway mark is at 3 data points per trace. This seems to be very low number, which could mean that the mobility trace extraction algorithm combines multiple records in some cases.

5.3 Two-fold cross-validation

The first section of this chapter has detailed how the ground truth data was collected. The second section described how this data was prepared for further analysis, as well as the data's relevant features. In this section, a two-fold cross-validation analysis of the classification algorithm is presented. The setup for the two-fold cross-validation analysis is presented in the following section. Successively, the results of that analysis are presented and the last section presents an interpretation of these results.

5.3.1 Validation setup

For the two-fold cross-validation method, the data set is split into two sets, d_0 and d_1. The algorithm is run on set d_0 using different parameter values to find the optimal performance. These parameter values are then used to run the algorithm on set d_1. Subsequently, the algorithm's parameters are optimized on d_1 and run on d_0 using those optimized parameters. The outcomes of the algorithm running on d_1 with the optimal parameters of d_0 and vice versa indicate the algorithms performance on a general data set. The advantage of the 2-fold cross-validation method is that all data is used both as training and as testing data, which is especially beneficial with small data sets.

The evaluation set consists of 89 traces that could be matched to evaluation trips. This set is randomly divided into two sets, d_0 with 44 and d_1 with 45 traces. The algorithm is executed on the first of these two sets with different values for the parameters. These parameters are the following:

Time margin For each record in a trace, the algorithm registers which buses were present in the network cell around the time that that record was created. This parameter sets the margin for how exactly the bus arrival and departure times have to coincide with the data point time for the match to be tracked. The tested parameter values are 2, 5, 10, and 15 minutes.

Minimum data points The more often a trace indicates that the mobile phone user was in the same cell as a specific bus, the more likely it is that the user was on that bus. This parameter sets the minimum number of data points that indicate the mobile phone user was in the same area as a bus for the trace to be considered a bus trace. The tested parameter values are 2, 3, and 4 times.

Minimum time difference The longer the time in between a trace first indicating a mobile phone user was in the same area as a specific bus and the last time that happens, the more likely it is that the user was on that bus. This parameter sets the minimum time difference between the first and the last time a mobile phone user was in the same area as a bus before the trace is considered a bus trace. The tested parameter values are 5, 10, 15 minutes.

Transportation mode pre-filter The algorithm uses Graphhopper to determine an initial travel mode classification based on travel time. Mobility traces are then excluded from further analysis if they seem unlikely to be bus traces. This parameter determines which initial transportation mode classification leads to traces being tagged as non-bus traces in this step. The tested parameter values are car, bike, foot, or none.

The number of algorithm executions increases in a multiplicative manner with each tested parameter. Therefore, the first three parameters introduced above are tested initially. The transportation mode pre-filter parameter is successively tested in combination with the best parameter values from the initial runs.

5.3.2 Results

Table 11 details the evaluation results from running the algorithm on set d_0. Four values were tested for the time margin parameter, three for minimum data points parameter, and three for minimum time difference parameter. Thus, a total of 36 parameter value combinations were tested. The first three columns in Table 11 state the parameter values used in the execution. The accuracy column shows the ratio of traces that were classified correctly as either bus or non-bus traces. A non-bus trace that is classified as a bus trace is called a false positive. The false-pos column shows the ratio of false positives to the total number of non-bus traces. This value represents the likelihood of a non-bus trace being misclassified as a bus trace. Similarly, a false negative is a bus trace that is classified as a non-bus trace. The ratio of false negatives to the total number of bus traces is shown in the false-neg column. Finally, the Δ(f-pos,f-neg) column shows the difference between the false-neg and false-pos columns. If this metric is close to zero, the algorithm is equally accurate in classifying bus traces as in classifying non-bus traces. If it is close to one, the algorithm's accuracy is very unbalanced and performs much better at classifying either bus traces *or* non-bus traces.

Since no single configuration of parameters results in the best values for all metrics, a prioritization has to be made. An intuitive approach would be to use overall accuracy as the defining metric. However, this would not enable differentiating between a good functioning algorithm and a bad one. Since the classification is binary —either a trace is classified as traveled by bus or it is classified as not traveled by bus— a trade-off can occur between the false positive and false negative ratio. After all, in the extreme scenario where the algorithm classifies all traces as non-bus traces, the false positive ratio would be zero, i.e., no non-bus trace is misclassified as a bus trace. At the same time, the false negative ratio would be one, since all bus traces would be misclassified as non-bus traces. Conversely, if the algorithm naively classified all traces as bus traces, the false positive ratio would be one and the false negative ratio zero. Neither of these scenarios would indicate a well-functioning algorithm, even though the overall accuracy could be quite high. The d_0 data set has more non-bus traces than bus traces. Therefore, the overall accuracy on classifying this set depends mostly on the false positive ratio. Table 11 shows that a 75% accuracy rate can be reached while as much as 62% of bus traces are misclassified.

time-margin	data-points	time-delta	accuracy	false-pos	false-neg	Δ(f-pos,f-neg)
15	3	10	0.68	0.32	0.31	0.01
15	4	10	0.68	0.32	0.31	0.01
10	2	5	0.57	0.43	0.44	0.01
10	3	5	0.57	0.43	0.44	0.01
10	4	5	0.57	0.43	0.44	0.01
15	2	10	0.66	0.36	0.31	0.04
10	2	10	0.64	0.32	0.44	0.12
15	2	15	0.59	0.36	0.50	0.14
10	3	10	0.66	0.29	0.44	0.15
10	4	10	0.66	0.29	0.44	0.15
15	3	15	0.61	0.32	0.50	0.18
15	4	15	0.61	0.32	0.50	0.18
15	2	5	0.57	0.50	0.31	0.19
15	3	5	0.57	0.50	0.31	0.19
15	4	5	0.57	0.50	0.31	0.19
10	2	15	0.59	0.32	0.56	0.24
...	...	...	...	...	...	...

Table 11: Evaluation results for training on set d_0. First 17 rows are shown, sorted by Δ(f-pos,f-neg) and accuracy.

time-margin	data-points	time-delta	accuracy	false-pos	false-neg	Δ(f-pos,f-neg)
15	2	5	0.62	0.37	0.39	0.02
15	3	5	0.62	0.37	0.39	0.02
15	4	5	0.60	0.37	0.44	0.07
15	2	10	0.62	0.33	0.44	0.11
15	3	10	0.64	0.30	0.44	0.15
10	2	5	0.60	0.33	0.50	0.17
10	3	5	0.60	0.33	0.50	0.17
15	2	15	0.60	0.33	0.50	0.17
15	4	10	0.62	0.30	0.50	0.20
10	4	5	0.58	0.33	0.56	0.22
15	3	15	0.64	0.26	0.50	0.24
10	2	10	0.60	0.30	0.56	0.26
5	2	5	0.62	0.26	0.56	0.30
5	3	5	0.62	0.26	0.56	0.30
10	3	10	0.62	0.26	0.56	0.30
15	4	15	0.62	0.26	0.56	0.30
...	...	...	...	...	...	...

Table 12: Evaluation results for training on set d_1. First 17 rows are shown, sorted by Δ(f-pos,f-neg) and accuracy.

excluded	time-margin	data-points	time-delta	accuracy	false-pos	false-neg	Δ(f-pos, f-neg)
none	15	3	10	0.68	0.32	0.31	0.01
car	15	3	10	0.68	0.29	0.38	0.09
bike	15	3	10	0.68	0.18	0.56	0.38
foot	15	3	10	0.64	0.18	0.69	0.51

Table 13: Evaluation results for training on set d_0 using the transportation mode pre-filter

excluded	time-margin	data-points	time-delta	accuracy	false-pos	false-neg	Δ(f-pos, f-neg)
none	15	2	5	0.62	0.37	0.39	0.02
car	15	2	5	0.67	0.30	0.39	0.09
bike	15	2	5	0.62	0.22	0.61	0.39
foot	15	2	5	0.53	0.26	0.78	0.52

Table 14: Evaluation results for training on set d_1 using the transportation mode pre-filter

Therefore, the difference between the false positive and the false negative ratio is used as the primary performance metric for the algorithm. If this metric is low, the algorithm classifies both bus and non-bus traces approximately equally accurately. If different parameter value combinations lead to similar Δ(f-pos,f-neg) values, overall accuracy is used as a second performance metric. Table 11 is sorted by ascending values for Δ(f-pos,f-neg) and descending accuracy values. The best values for the parameters for d_0 are a time margin of 15 minutes, minimum data points of 3 or 4, and a minimum time difference of 10 minutes.

The initial test was run without pre-filtering on an initial transportation mode classification based on travel time. The parameters for the transportation mode filter are tested using the best parameters from the first test. The results from this test are shown in Table 13. The table shows that not excluding any traces based on their initial transportation mode classification gives the best results, closely followed by excluding traces initially classified as car traces.

Using the best parameters from training on set d_0, the algorithm is executed on set d_1. For this execution, traces initially classified as car traces were excluded. The results of the execution are detailed in Table 15.

The second part of the two-fold cross-validation analysis, is to train on set d_1 and test on set d_0. This is executed in an identical manner to the first part where the training happened on d_0 and the testing on d_1. The results of the training phase are presented in Table 12 and Table 14. The optimal parameter values for d_1 are a time margin of 15 minutes, minimum data points of 2 or 3, and a minimum time difference of 5 minutes. No pre-filtering based on an initial transportation mode classification and pre-filtering traces initially classified as car traces outperform configurations

were bike or foot traces are excluded. Excluding no traces results in a slightly lower difference between the false positive and false negative ratios than when excluding car traces, however, the overall accuracy is slightly higher in the latter case. Therefore, excluding car traces was chosen as the optimal value. The optimal parameters values from training on d_1 are used to execute the algorithm on set d_0. The results of this test are presented in Table 16.

5.3.3 Interpretation

To examine the effects of the different algorithm parameters individually, the results are averaged per value for each parameter. The averages for the time margin, minimum data points, and minimum time difference parameters are detailed in Table 17, 18, and 19 respectively.

A higher value for the time margin parameter leads to more traces being classified as bus traces. Consequently, less actual bus traces are misclassified as car traces as evidenced by the lower false negative rate. However, more car traces are misclassified as bus traces as evidenced by the higher false positive rate. Conversely, higher values for the minimum data points and minimum time difference parameters lead to lower false positive rates and higher false negative rates. Comparing the tables with each other shows that the time margin parameter has the most significant effect on the results. The data points and time-delta parameters effect a change in the average overall accuracy and the Δ(f-pos,f-neg) metric of less than 10 percentage points. The time margin parameter shows a larger effect on the average overall accuracy and especially on the Δ(f-pos,f-neg) rate.

The best indication for the algorithm's performance on a general data set is given by the outcomes of the two-fold cross-validation analysis presented in Table 15 and 16. Averaging those outcomes results in an overall accuracy rate of 63% and a difference between the false positive and false negative rates of 12 percent point.

excluded	time-margin	data-points	time-delta	accuracy	false-pos	false-neg	Δ(f-pos, f-neg)
car	15	3	10	0.67	0.26	0.44	0.19

Table 15: Evaluation results for testing on set d_1 after training on set d_0

excluded	time-margin	data-points	time-delta	accuracy	false-pos	false-neg	Δ(f-pos, f-neg)
car	15	2	5	0.59	0.43	0.38	0.05

Table 16: Evaluation results for testing on set d_0 after training on set d_1

time-margin	avg. accuracy	avg. false-pos	avg. false-neg	avg. Δ(f-pos,f-neg)
2	0.72	0.04	0.71	0.67
5	0.64	0.23	0.60	0.38
10	0.61	0.34	0.48	0.14
15	0.62	0.39	0.38	0.12

Table 17: Evaluation results for training on set d_1, grouped and averaged per time-margin value

data-points	avg. accuracy	avg. false-pos	avg. false-neg	avg. Δ(f-pos,f-neg)
2	0.64	0.26	0.54	0.32
3	0.65	0.24	0.54	0.33
4	0.64	0.24	0.55	0.34

Table 18: Evaluation results for training on set d_1, grouped and averaged per data-points value

time-delta	avg. accuracy	avg. false-pos	avg. false-neg	avg. Δ(f-pos,f-neg)
5	0.63	0.30	0.49	0.28
10	0.67	0.22	0.52	0.31
15	0.64	0.22	0.61	0.39

Table 19: Evaluation results for training on set d_1, grouped and averaged per time-delta value

6 Conclusion

In this thesis, the design and implementation of a distributed travel mode discovery algorithm has been presented. The main input for the algorithm is a set of mobility traces extracted from Data Detail Records or Call Detail Records. The algorithm classifies these traces as either corresponding to a trip made by public transportation bus or as corresponding to a trip made using another mode of transportation. Identifying the use of public transportation buses is a unique feature of this program compared to existing solutions. Similarly, other travel mode classification algorithms either focus on larger, e.g., inter-city distances, or they use data with a finer granularity, such as GPS traces for their analysis.

Through configuring the algorithm in different ways, various insights about people's mode of transportation can be obtained. The algorithm can be configured to count how many people travel to a specific area by bus. This query can be further specified to only count travelers arriving at the specified area within a set time-frame. Alternatively, the algorithm can be set to estimate the number of people traveling on a specific bus on a given day. Theses features fulfill the sub-requirements of RQ 2. How well the other requirements for the algorithm are fulfilled is summarized in Table 20.

The classification algorithm has been designed to run distributedly using the Apache Spark framework. Therefore, it can handle large input datasets of mobility traces by distributing partitions of the data to different worker nodes in a cluster. Network infrastructure data from a mobile network operator is used to map the mobility traces to a sequence of geographical areas. These areas represent the coverage areas of the network cells that a mobile phone user was in. Public bus timetable data is used to discover how often a mobile phone user was near a bus station at the time that a bus stopped there. Based on this information, the algorithm classifies a mobility trace as a bus trace or as a non-bus trace.

An evaluation dataset using whitelisted SIM cards was created specifically for this thesis. A dataset of 89 traces was used for the evaluation. These traces were collected in the city of Berlin, Germany, using 4 phones. Since the algorithm can be tuned to more accurately classify different datasets, a two-fold cross-validation analysis was conducted. The evaluation dataset was randomly divided into two sets, d_0 and d_1. Consecutively, the algorithm was trained on d_0. Using the best parameter values from that training stage, the algorithm was executed on d_1. Similarly, the algorithm was trained on d_1 and executed on d_0, again using the best parameter values from the training stage.

Consolidating the results from both evaluation phases, the algorithm has an average accuracy rate of 63%. The average false positive rate, i.e., the likelihood of a non-bus trace to be misclassified as a bus trace, is 34%. The average false negative rate, i.e., the likelihood of a bus trace to be misclassified as a non-bus trace, is 41%. These accuracy rates are not satisfactory as such. However, this is the first implementation of a classification algorithm that uses CDR/DDR-based mobility traces for travel mode discovery on an intra-city scale. Therefore, it is likely that these results can be improved upon. Furthermore, the traces used for evaluation

resemble the worst-case scenario for classification. Evaluation trips made by car had the same origin and destination as public transportation buses and they left at the same time as a bus in the same direction. In reality, bus and non-bus trips are unlikely to be that similar, making distinguishing between them more accurate.

The presented distributed data analysis algorithm is currently rule-based with hard cutoffs. There is a threshold for whether the timestamp of a trace matches the scheduled stop time of a bus closely enough. Similarly, there is a threshold for how often a trace should match with the bus timetable before it is considered a bus trace. These hard thresholds could be replaced by probabilities to improve the classification. Different metrics could then influence the probability of a trace being classified as a bus trace.

Another improvement of the algorithm would be to use information from one mobility trace in the processing of another. Many modes of transportation can accommodate multiple travelers, especially public transportation modes. Therefore, the information from mobility traces of different travelers could be combined to determine if it is probable that the traces correspond to the same mode of transportation.

Lastly, the algorithm could be adapted to classify other modes of transportation in addition to public transportation buses. For most forms of public transportation, this would be straightforward. Timetable data and information on station locations is likely to be available for these public transportation methods.

Requirement	Explanation	Fulfilled
RQ 1 A	Traces are classified as bus traces or non-bus traces with 63% accuracy. This rate is not satisfactory in itself, although it is a promising start.	?
RQ 1 B	The algorithm can be tuned using various parameters to optimize it for different datasets.	✓
RQ 2 A	Traces can be filtered based on the trace ending at, or passing through a specified area of interest.	✓
RQ 2 B	Traces can be filtered on being in an area within specific time-frame.	✓
RQ 2 C	The algorithm can estimate the number of passengers on a specific bus.	✓
RQ 3	Proprietary network infrastructure data is used out of necessity. Otherwise than that, public bus timetable data and open source routing data is used.	✓
RQ 4	The algorithm is developed on top of the Apache Spark framework and designed to be distributable.	✓
RQ 5	The evaluation dataset contained too few mobility traces to analyze how the execution time would increase with the input data size.	?

Table 20: Fulfillment of the requirements presented in Chapter 3

References

[1] V. Aguiléra, S. Allio, V. Benezech, F. Combes, and C. Milion. Using cell phone data to measure quality of service and passenger flows of Paris transit system. *Transportation Research Part C: Emerging Technologies*, 43:198–211, 2014.

[2] M. Alaggan, S. Gambs, S. Matwin, E. Souza, and M. Tuhin. Sanitization of call detail records via differentially-private summaries. *arXiv preprint arXiv:1412.8412*, 2014.

[3] M. Alger, E. Wilson, T. Gould, R. Whittaker, and N. Radulovic. Real-time traffic monitoring using mobile phone data. *Online: https://www.maths-in-industry.org/past/ESGI/49/problems/vodafone/vodafone.pdf*, 2004.

[4] H. Bar-Gera. Evaluation of a cellular phone-based system for measurements of traffic speeds and travel times: A case study from Israel. *Transportation Research Part C: Emerging Technologies*, 15(6):380–391, 2007.

[5] R. A. Becker, R. Cáceres, K. Hanson, J. M. Loh, S. Urbanek, A. Varshavsky, and C. Volinsky. A tale of one city: Using cellular network data for urban planning. *IEEE Pervasive Computing*, 10(4):18–26, 2011.

[6] L. Bengtsson, X. Lu, A. Thorson, R. Garfield, and J. von Schreeb. Improved response to disasters and outbreaks by tracking population movements with mobile phone network data: A post-earthquake geospatial study in Haiti. *PLoS Medicine*, 8(8), 2011.

[7] P. Bonnel, E. Hombourger, A.-M. Olteanu-Raimond, and Z. Smoreda. Passive mobile phone dataset to construct origin-destination matrix: Potentials and limitations. *Transportation Research Procedia*, 11:381–398, 2015.

[8] G. J. Brouwer and F. Provost. Deviations from the norm: Detecting regularities and anomalies in human mobility patterns. *arXiv*, (1):343–351, 2014.

[9] N. Caceres, L. M. Romero, F. G. Benitez, and J. M. Del Castillo. Traffic flow estimation models using cellular phone data. *IEEE Transactions on Intelligent Transportation Systems*, 13(3):1430–1441, 2012.

[10] F. Calabrese, M. Colonna, P. Lovisolo, D. Parata, and C. Ratti. Real-time urban monitoring using cell phones: A case study in Rome. *IEEE Transactions on Intelligent Transportation Systems*, 12(1):141–151, 2011.

[11] F. Calabrese, G. D. Lorenzo, L. Liu, and C. Ratti. Estimating origin-destination flows using mobile phone location data. *IEEE Pervasive Computing*, 10(4):36–44, Apr 2011.

[12] F. Calabrese, F. C. Pereira, G. Di Lorenzo, L. Liu, and C. Ratti. The geography of taste: Analyzing cell-phone mobility and social events. In *International Conference on Pervasive Computing*, pages 22–37. Springer, 2010.

[13] F. Calabrese and C. Ratti. Real time rome. *Networks and Communication studies*, 20(3-4):247–258, 2006.

[14] J. Chenal, M. Bonriposi, G. Josse, P. de Oliveira, B. Grèzes-Besset, H. Chicault, and C. Thao. The municipality and the territory: Two scales of understanding the city. *arXiv*, (1):457–479, 2014.

[15] Cisco / Starent Networks. LTE: Simplifying the migration to 4G networks, 2010.

[16] L. M. Correia, D. Zeller, O. Blume, D. Ferling, Y. Jading, I. Gódor, G. Auer, and L. Van der Perre. Challenges and enabling technologies for energy aware mobile radio networks. *Communications Magazine, IEEE*, 48(11):66–72, 2010.

[17] M. Dash, H. L. Nguyen, C. Hong, J. Decraene, S. Antonatos, Y. Wang, D. T. Anh, A. Shi-nash, G. E. Yap, M. N. Nguyen, X. Li, S. P. Krishnaswamy, J. Decraene, S. Antonatos, Y. Wang, Others, D. T. Anh, and A. Shi-nash. Home and work place prediction for urban planning using mobile network data. In *Mobile Data Management (MDM), 2014 IEEE 15th International Conference on*, volume 2, pages 37–42. IEEE, 2014.

[18] P. Demestichas, A. Georgakopoulos, D. Karvounas, K. Tsagkaris, V. Stavroulaki, J. Lu, C. Xiong, and J. Yao. 5G on the horizon: Key challenges for the radio-access network. *IEEE Vehicular Technology Magazine*, 8(3):47–53, Sep 2013.

[19] T. Derrmann, R. Frank, F. Melakessou, G. Castignani, and T. Engel. A cell dwell time model for synthetic population generation from call detail records. *arXiv preprint arXiv:1407.4885*, (1):491–499, 2014.

[20] B. Deva and P. Ruppel. Location analytics as a service: Providing insights for heterogeneous spatiotemporal data. *2015 IEEE International Conference on Web Services*, pages 353–360, 2015.

[21] D. Doran, A. Fox, and V. Mendiratta. Where do we develop? Discovering regions for urban investment in Senegal. *CoRR*, abs/1407.4885:535–545, 2014.

[22] J. Doyle, P. Hung, D. Kelly, and R. Farrell. Utilising Mobile Phone Billing Records for Travel Mode Discovery. *ISSC 2011, Trinity College Dublin, June 23–24*, 2011.

[23] A. El-Mahdy, E. Algizawy, T. Ogawa, H. Shishiny, M. Baddar, and K. Kimura. High resolution traffic maps generation using cellular big data. *arXiv*, (1):552–593, 2014.

[24] T. Fetzer. Urban road construction and human mobility: Evidence from Dakar, Senegal. pages 399–412, 2014.

[25] P. Fiadino, D. Valerio, F. Ricciato, and K. A. Hummel. Steps towards the extraction of vehicular mobility patterns from 3G signaling data. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7189 LNCS:66–80, 2012.

[26] M. Ficek and L. Kencl. Inter-call mobility model: A spatio-temporal refinement of call data records using a Gaussian mixture model. *2012 Proceedings IEEE INFOCOM*, (July 2015):469–477, 2012.

[27] J. Froehlich and J. Krumm. Route prediction from trip observations. *Proceedings of SAE World Congress*, 2193(2008-01-0201):53, 2008.

[28] A.-H. Fujihara, D. Nogiwa, T. Kasai, and S. Maegawa. Exploring relationships between human mobility motifs and rainfall for development. *arXiv*, (1):599–612, 2014.

[29] B. Furletti, L. Gabrielli, S. Rinzivillo, and C. Renso. Identifying users profiles from mobile calls habits. *Proceedings of the ACM SIGKDD International Workshop on Urban Computing. ACM*, pages 17–24, 2012.

[30] L. Gabrielli, B. Furletti, F. Giannotti, and M. Nanni. Use of mobile phone data to estimate visitors mobility flows. *SEFM 2014 Workshops, LNCS*, 8938:214–226, 2015.

[31] S. Gambs, M.-O. Killijian, and M. N. Del Prado Cortez. Next place prediction using mobility Markov chains. *Proceedings of the First Workshop on Measurement Privacy and Mobility MPM 2012*, pages 1–6, 2012.

[32] M. C. González, C. A. Hidalgo, and A.-L. Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

[33] S. Gueye, B. M. Ndiaye, D. Josselin, M. Poss, R. M. Faye, P. Michelon, C. Genre-Grandpierre, and F. Ciari. Using mobile phone data for Spatial Planning simulation and Optimization Technologies (SPOT). *arXiv*, (1):516–534, 2014.

[34] D. Gundlegård, C. Rydergren, N. Breyer, and B. Rajna. Travel demand estimation and network assignment based on cellular network data. *Computer Communications*, 95:29–42, 2016.

[35] Y. J. Gur, S. Bekhor, C. Solomon, and L. Kheifits. Use of massive cell phone data to obtain inter-city person trip tables for nationwide transportation planning in israel use of massive cell phone data to obtain inter-city person trip tables for nationwide transportation planning in Israel. 2121(November 2008):145–151, 2009.

[36] E. Halepovic and C. Williamson. Characterizing and modeling user mobility in a cellular data network. *Proceedings of the 2nd ACM international workshop on*

Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks - PE-WASUN '05, page 71, 2005.

[37] C. Horn and R. Kern. Deriving public transportation timetables with large-scale cell phone data. *Procedia Computer Science*, 52(Ant):67–74, 2015.

[38] S. Hoteit, S. Secci, S. Sobolevsky, C. Ratti, and G. Pujolle. Estimating human trajectories and hotspots through mobile phone data. *Computer Networks*, 64:296–307, 2014.

[39] IHS Markit. Small cells to play a big part in network architecture of the future, 2011.

[40] C. Iovan, A.-M. Olteanu-Raimond, T. Couronné, and Z. Smoreda. Moving and calling: Mobile phone data quality measurements and spatiotemporal uncertainty in human mobility studies. In *Geographic information science at the heart of Europe*, volume 2013-Janua, pages 247–265. Springer, 2013.

[41] S. Isaacman, R. Becker, M. Martonosi, J. Rowland, and A. Varshavsky. Identifying important places in people's lives from cellular network data. *Pervasive Computing*, 6696:133–151, 2011.

[42] A. Janecek, D. Valerio, K. Hummel, F. Ricciato, and F. Hlavacs. Cellular data meet vehicular traffic theory: location area updates and cell transitions for travel time estimation. *In Proceedings of the 2012 ACM Conference on Ubiquitous Computing, ACM*, pages 361–370, 2012.

[43] A. Janecek, D. Valerio, K. A. Hummel, F. Ricciato, and H. Hlavacs. The cellular network as a sensor: From mobile phone data to real-time road traffic monitoring. *Intelligent Transportation Systems, IEEE Transactions on*, 16(5):2551–2572, 2015.

[44] J. Jeong, M. Leconte, and A. Proutiere. Improving mobility predictability based on user clustering on D4D dataset. *arXiv*, (1):500–505, 2014.

[45] F. Jiménez, F. Aparicio, and J. Páez. Evaluation of in-vehicle dynamic speed assistance in Spain: Algorithm and driver behaviour. *IET Intelligent Transport Systems*, 2(2):132, 2008.

[46] M. Kirchberger and C. Small. Text me - if you can: Literacy, networks, and mobility in Dakar. *arXiv*, (1):613–626, 2014.

[47] R. Kujala, T. Aledavood, and J. Saramäki. Estimation and monitoring of city-to-city travel times using call detail records. *EPJ Data Science*, 5(1):1, 2016.

[48] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, number 3, pages 106–115. Institute of Electrical and Electronics Engineers (IEEE), 2007.

[49] W. Li, X. Cheng, Z. Duan, D. Yang, and G. Guo. A framework for spatial interaction analysis based on large-scale mobile phone data. *Computational intelligence and neuroscience*, 2014:363502, 2014.

[50] T. Liang and V. Frias-Martinez. Cars and calls: Using CDR data to approximate official traffic counts. *arXiv*, (1):427–432, 2014.

[51] F. Liu, D. Janssens, J. Cui, and G. Wets. Building workers' travel demand models based on mobile phone data. *arXiv*, (1):379–398, 2014.

[52] X. Lu, E. Wetter, N. Bharti, A. J. Tatem, and L. Bengtsson. Approaching the limit of predictability in human mobility. *Nature Scientific Reports*, 3:2923, 2013.

[53] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. *L*-diversity. *ACM Transactions on Knowledge Discovery from Data*, 1(1):3–es, 2007.

[54] D. Naboulsi, M. Fiore, S. Ribot, and R. Stanica. Large-scale mobile traffic analysis: A survey. *Communications Surveys Tutorials, IEEE*, PP(99):1, 2015.

[55] M. a. Quddus, W. Y. Ochieng, and R. B. Noland. Current map matching algorithms for transport applications: State-of-the art and future research directions. *TRB Annual Meeting, Washington 2007*, 2007.

[56] G. Ranjan, H. Zang, Z.-L. Zhang, and J. Bolot. Are call detail records biased for sampling human mobility? *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(3):33–44, 2012.

[57] J. Reades, F. Calabrese, A. Sevtsuk, and C. Ratti. Cellular census: Explorations in urban data collection. *IEEE Pervasive Computing*, 6(3):30–38, 2007.

[58] G. Rose. Mobile phones as traffic probes: Practices, prospects and issues. *Transport Reviews*, 26(3):275–291, 2006.

[59] M. Sauter. *From GSM to LTE: An introduction to mobile networks and mobile broadband*. John Wiley & Sons, 2010.

[60] J. Schlaich, T. Otterstatter, and M. Friedrich. Generating trajectories from mobile phone data. *Transportation Research Board 89th Annual Meeting*, pages 1–18, 2010.

[61] C. M. Schneider, V. Belik, T. Couronné, Z. Smoreda, and M. C. González. Unravelling daily human mobility motifs. *Journal of the Royal Society, Interface / the Royal Society*, 10(84):20130246, 2013.

[62] D. Schulz, S. Bothe, and C. Korner. Human mobility from GSM data - A valid alternative to GPS? *Mobile data challenge 2012 workshop, June*, 2012:18–19, 2012.

[63] A. Schwering. Approaches to semantic similarity measurement for geo-spatial data: A survey. *Transactions in GIS*, 12(1):5–29, 2008.

[64] A. Singla, S. Agrawal, K. R. Aaatre, and P. S. Lyer. Operational Planning of Transport and Infrastructure using Mobile data Analysis (OPTIMA). *CoRR*, abs/1407.4885:453–456, 2014.

[65] Z. Smoreda, a.M. Olteanu-Raimond, and T. Couronné. Spatiotemporal data from mobile phones for personal mobility assessment. *Transport Survey Methods: Best Practice for Decision Making, Emerald, J. Zmud et al. (eds).*, (September 2015):1–20, 2013.

[66] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási. Limits of predictability in human mobility. *Science (New York, N.Y.)*, 327:1018–1021, 2010.

[67] J. Steenbruggen, E. Tranos, and P. Nijkamp. Data from mobile phone operators: A tool for smarter cities? *Telecommunications Policy*, 39(3-4):335–346, 2015.

[68] P. Stopher, C. FitzGerald, and J. Zhang. Deducing mode and purpose from GPS data. *87th Annual Meeting of the Transportation Research Board*, pages 1–11, 2008.

[69] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[70] M. Tizzoni, P. Bajardi, A. Decuyper, G. Kon Kam King, C. M. Schneider, V. Blondel, Z. Smoreda, M. C. González, and V. Colizza. On the use of human mobility proxies for modeling epidemics. *PLoS Computational Biology*, 10(7), 2014.

[71] S. M. Tondare, S. D. Panchal, and D. T. Kushnure. Evolutionary steps from 1G to 4.5G. *International Journal of Advanced Research in Computer and Communication Engineering*, 3(4):6163–6166, 2014.

[72] V. A. Traag, A. Browet, F. Calabrese, and F. Morlot. Social event detection in massive mobile phone data using probabilistic location inference. In *Privacy, security, risk and trust (PASSAT) and 2011 IEEE Third inernational conference on social computing (SocialCom), 2011 IEEE Third International Conference on*, pages 625–628. IEEE, 2011.

[73] R. Trasarti, A. M. Olteanu-Raimond, M. Nanni, T. Couronné, B. Furletti, F. Giannotti, Z. Smoreda, and C. Ziemlicki. Discovering urban and country dynamics from mobile phone data with spatial correlation patterns. *Telecommunications Policy*, 39(3-4):347–362, 2015.

[74] I. Trestian, A. Nucci, S. Ranjan, and A. Kuzmanovic. Measuring serendipity: Connecting people, locations and interests in a mobile 3G network. *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement - IMC '09*, page 267, 2009.

[75] D. Valerio, A. D'Alconzo, F. Ricciato, and W. Wiedermann. Exploiting cellular networks for road traffic estimation: A survey and a research roadmap. *IEEE Vehicular Technology Conference*, pages 1–5, 2009.

[76] D. Wang and T. Cheng. A spatio-temporal data model for activity-based transport demand modelling. *International Journal of Geographical Information Science*, 15(6):561–585, 2001.

[77] H. Wang, F. Calabrese, G. Di Lorenzo, and C. Ratti. Transportation mode inference from anonymized and aggregated mobile phone call detail records. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pages 318–323, 2010.

[78] Y. Wang, G. H. de Almeida Correia, and E. de Romph. National and regional road network optimization for Senegal using mobile phone data. *CoRR*, abs/1407.4885:352–378, 2014.

[79] A. Wesolowski, N. Eagle, A. M. Noor, R. W. Snow, and C. O. Buckee. The impact of biases in mobile phone ownership on estimates of human mobility. *Journal of the Royal Society, Interface / the Royal Society*, 10(81):20120986, 2013.

[80] P. Widhalm, Y. Yang, M. Ulm, S. Athavale, and M. C. González. Discovering urban activity patterns in cell phone data. *Transportation*, 42(4):597–623, 2015.

[81] G. Yavaş, D. Katsaros, Ö. Ulusoy, and Y. Manolopoulos. A data mining approach for location prediction in mobile environments. *Data and Knowledge Engineering*, 54(2):121–146, 2005.

[82] H. Zang and J. Bolot. Anonymization of location data does not work: A large-scale measurement study. *Proceedings of the 17th annual international conference on Mobile computing and networking, ACM*, pages 145–156, 2011.

[83] Y. Zhang. User mobility from the view of cellular data networks. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. Institute of Electrical and Electronics Engineers (IEEE), Apr 2014.

[84] M. Zilske and K. Nagel. A simulation-based approach for constructing all-day travel chains from mobile phone data. *Procedia Computer Science*, 52:468–475, 2015.

A Example configuration file

```
{
  "steps": {
    "read_input_traces": {
      "execute": true,
      "input_file": "/home/tu/ODM_results/raw_trips/trips_07.csv"
    },
    "shape_filter": {
      "execute": false,
      "shape": "POLYGON ((14.42562 50.08981, 14.42901 50.08981, 14.42901
       ↪  50.08826, 14.42562 50.08826, 14.42562 50.08981))",
      "only_end_at_shape": false,
      "min_touch_time": "2016-01-01 08:00:00",
      "max_touch_time": "2017-01-01 08:00:00"
    },
    "precompute_times": {
      "execute": false,
      "graphhopper_url": "http://localhost:8989"
    },
    "travel_times": {
      "execute": false
    },
    "filter_time": {
      "execute": false
    },
    "stop_info": {
      "execute": false,
      "bus_input_dir": "/home/tu/Documents/bus_lines/prague/"
    },
    "cilac_stops": {
      "execute": false
    },
    "add_stop_times": {
      "execute": true
    },
    "add_matches": {
      "execute": true,
      "timestamp_tolerance": 3,
      "bus_tolerance": 3,
      "filter_on_weekday": false
    },
    "add_match_info": {
      "execute": true
    },
    "classify": {
      "execute": true,
      "trip_id": null,
      "min_timedelta": 10,
      "min_cilacs": 3
    }
  },
  "options": {
    "telco_multiplier": 1,
```

```
52        "output_directory": "../output/",
53        "cells_query": "(SELECT id, cilac, (geom::bytea) AS geom,
          ↪  (geom_centroid::bytea) AS geom_centroid from locations_prague_50k) AS
          ↪  cells",
54        "trace_columns": ["coords", "cilac", "ts", "device_id", "start_cell"
          ↪  ,"end_cell"],
55        "timestamp_in_ms": true
56      }
57    }
```

B Software configuration documentation

B.1 Introduction

Project for analyzing traces based on Call Detail Records (CDRs) or Data Detail Records (DDRs) to identify the transportation mode used by each traveler, specifically whether a user traveled by public transport buses or by car. This document explains how to configure, set up, and run the project.

B.2 Configuration

The main directory has a config file called `config.js`. This file contains the settings for program execution. An example file is shown in Appendix A.

`steps` is a dictionary with the algorithm steps and the options specific to each step. Each step has a boolean `execute` that indicates whether the step should be executed. Below is an explanation of what each step does and the options for that step. Implicit input and output files are also stated. Some steps overwrite a file they use for input.
Note that output files are overwritten automatically and that a step may depend on a specific version of an input file.

- `read_input_traces` read the raw traces from a CSV file.
  - Options
    - `input_file` specifies the CSV file to read from.
- `shape_filter` filter raw traces on whether they touch a shape.
  - Input
    - *traces*
  - Output
    - *traces*
  - Options
    - `shape` is a shape string in the WKT format. The algorithm finds the cells that overlap with this shape and filters the traces that pass through at least one of those cells.
    - `only_end_at_shape` indicates whether only traces ending at the shape or traces touching the shape at any point should be filtered.
    - `min_touch_time` the time after which a trace can pass the `shape`. (Eg. if only buses after 10 am should be matched.) If `null`, traces are not filtered on this. Value in the form of "2016-01-01 08:00:00".
    - `max_touch_time` the time before which a trace can pass the `shape`. (Eg. if only buses before 12 pm should be considered.) If `null`, traces are not filtered on this. Value in the form of "2017-01-01 08:00:00".

- `precompute_times` computes travel times for different transport modes using Graphhopper. It uses settings from the `shape_filter` step. If `only_end_at_shape` is set, only travel times from all cells to the specified `shape` are calculated. Otherwise, travel times from all cells to all cells are calculated.
  Warning: takes a long time, especially if `only_end_at_shape == false`.
  - Input
    - Cells from Postgres database
  - Output
    - *travel_times*
  - Options
    - `graphhopper_url` is to the base URL of the Graphhopper instance.
- `travel_times` classifies all traces by transport mode (car, bike, or foot) only based on Graphhopper travel times.
  - Input
    - *traces*
    - *travel_times*
  - Output
    - *traces_initial_transport_mode*
- `filter_time` filter traces by travel time by excluding traces that seem to be taken by car or could not be defined, since these traces are most probably not corresponding to traces taken by bus.
  - Input
    - *traces*
    - *traces_initial_transport_mode*
  - Output
    - *traces*
- `stop_info` combines public transport information from different files into a convenient format.
  - Input
    - *calendar.txt*, *trips.txt*, and *stop_times.txt* from the `bus_input_dir` directory
  - Output
    - *stop_info*

  - Options
    * `bus_input_dir` is the directory with files in the GTFS format.

- `cilac_stops` maps cells — uniquely identified by cilac (cell id + LAC id) — to bus stops in that cell.
  Warning: takes a long time, since all cilacs are matched to all bus stops. Recommended to only execute this once and then reuse it.

  - Input
    * *stops.txt* from the `bus_input_dir` directory used in the `stop_info` step.
  - Output
    * *cilac_stops*

- `add_stop_times` add information to all traces on what buses stop at what times for each cell the trace touches.

  - Input
    * *traces*
    * *stop_info*
    * *cilac_stops*
  - Output
    * *traces*

- `add_matches` adds to traces which buses stop in a cell at a time the trace touches that cell. Buses that only match a trace in one cell are ignored.

  - Input
    * *traces* after `add_stop_times`
  - Output
    * *traces*
  - Options
    * `timestamp_tolerance` the number of minutes a timestamp for a data point is allowed to mismatch with a bus for it to still be a match. This is important since a data point can be taken at any time while a user is in a certain cell.
    * `bus_tolerance` the number of minutes a bus can be late or early compared to the time schedule. A data point taken this long before or after a bus arrives according to the schedule, is still considered to match that bus. Increasing `bus_tolerance` has the exact same effect as increasing `timestamp_tolerance`.

        * `filter_on_weekday` boolean to set whether matches should take the day of the week into account. Should be `true` normally, however Berlin bus data does not include weekday information. If `false`, all bus services are assumed to operate on every day of the week.

- `add_match_info` adds more information on buses matching a trace, such as how often each bus is matched, in how many different cells a bus is matched, and how much time is in between the first and the last match.
  - Input
    * *traces* after `add_matches`
  - Output
    * *traces*
- `classify` classify traces as being (partially) taken by bus or not. Traces are seen as bus traces if they *either* match the same bus multiple times with a lot of time in between *or* if they match the same bus in many different cells.
  - Input
    * *traces* after `add_match_info`
  - Output
    * *traces*
  - Options
    * `trip_id` is the id of a specific trip that should be matched on. If set, passengers of a specific bus can be counted. If set to `null`, all traces matching any bus are matched.
    * `min_timedelta` the minimum amount of time between the first and last time a trace matches a specific bus for the trace to be classified as a bus trace. (The matches need to be in different cells.)
    * `min_cilacs` the minimum number of cells a specific bus should match a trace in for the trace to be classified as a bus trace.

`options` includes other options that are not related to specific steps:

- `telco_multiplier` To find the number of passengers taking buses (or taking a specific bus), the number of traces matched by the algorithm is multiplied by this number to account for the ratio of customers of the telecom operator to the overall population. (Eg. if one in four people are customers of telco X, the `telco_mutliplier` would be 4.)
- `output_directory` the directory that (intermediate) result files are written to. Existing result files will be overwritten. This is also the directory existing intermediate results will be read from if applicable (if a preceding step is not executed this time).

B.3 Setup

Other than the config file, some things need to be set up.

B.3.1 Spark

Download Apache Spark 1.6.2 from https://spark.apache.org/downloads.html (for any version of Hadoop, since Hadoop is not currently used).

B.3.2 Graphhopper

1. Download Graphhopper, see the quickstart guide specifically. It does not require building Graphhopper, but supplies a JAR file.

2. Download the appropriate OSM data (in osm.pbf format), for example from this website for Berlin.

3. Alter the config file (`config-example.properties` is included with Graphhopper as an example) to include a flag `graph.flag_encoders=car,foot,bike`. This ensures that Graphhopper prepares the data in the next step for routing with those transport modes.

4. Run

   ```
   java -jar *.jar jetty.resourcebase=webapp
   ↪ config=config-example.properties
   ↪ datareader.file=berlin-latest.osm.pbf
   ```

   with the appropriate names for the config and OSM files. The first start will take longer (~30 seconds for Berlin), further starts will be very fast. The Graphhopper server should now by running. By default, http://localhost:8989 should give access to a routing in

B.3.3 Python

Set up Python 3 with Shapely library installed.

B.3.4 Compute infrastructure cell centroids

Run the following SQL query on the PostGIS infrastructure table to compute and store the centroids of all cells:

```
ALTER TABLE infrastructure_cells ADD COLUMN geom_centroid bytea;
UPDATE infrastructure_cells SET geom_centroid =
↪ ST_AsEWKB(ST_Centroid(geom));
```

Here, `infrastructure_cells` is the table storing the infrastructure information and `geom` is the column storing the geometry of the cells.

B.4 Execution

To execute the algorithm in a Spark job, execute:

```
spark-submit --packages com.databricks:spark-csv_2.10:1.4.0
↪ --driver-class-path /home/user/posgresqlJDBC/driver-v.jar
↪ main.py
```

Where `/home/user/posgresqlJDBC/driver-v.jar` is the appropriate JDBC Postgres driver JAR. See https://jdbc.postgresql.org/download.html to download this.