Transparent and versatile traffic simulation: Supply data, demand data, and software architecture

vorgelegt von
Dipl.-Inform. Michael Zilske
geboren in Hamburg

von der Fakultät V – Verkehrs- und Maschinensysteme
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr.-Ing. Roland Baar
Gutachter: Prof. Dr. rer. nat. Kai Nagel
Gutachter: Prof. Dr.-Ing. Peter Vortisch

Tag der wissenschaftlichen Aussprache: 3. August 2017

Berlin 2018

Abstract

Traffic simulations, considered as software systems, are complex and data-intesive. Deploying a model instance for a planning scenario requires gathering data and developing software, often as a team effort. Both tasks are demanding in time and resources. Both basic research in traffic simulations and their practical use would benefit from simplifying these tasks.

Two major strains of development in the software industry are open source and open data. Open source refers to transparent software development, often as a team project, with source code shared on public platforms. Open data refers to providing data, often from government, public infrastructure and scientific research, for public use.

This dissertation examines three aspects of traffic simulations along these strains of development:

- Input data on traffic supply
- Input data on traffic demand
- A modular software architecture

In the first part, tools and processes are developed to exploit established sources of open data on traffic networks, particularly OpenStreetMap and transit schedules, for traffic simulations. The second part explores data sources on traffic. In reaction to the increasingly frequent availability of mobile phone datasets - a data source which is markedly not open, and whose prospects of openness are fundamentally limited by the requirement of the protection of personal data - the use of anonymized mobile phone datasets is examined. In the third part, a software architecture for traffic simulations is developed, whose main features are extensibility and transparency, two aspects which are critical in its successful use in research conducted in large teams.

Zusammenfassung

Verkehrssimulationen, als Softwaresysteme betrachtet, sind komplex und datenhungrig. Das Aufsetzen einer praktisch anwendbaren Modellinstanz für ein konkretes verkehrswissenschaftliches Untersuchungsszenario umfasst Datenbeschaffung und arbeitsteilige Softwareentwicklung. Beides sind langwierige und kostenintensive Teilaufgaben. Sowohl die Grundlagenforschung an Verkehrssimulationen als auch ihre praktische Anwendung würden davon profitieren, diese Aufgaben zu erleichtern.

Zwei große Entwicklungslinien in der Softwareindustrie sind Open-Source und Open-Data: Die zumeist arbeitsteilige, transparente Softwareentwicklung mit offenem Quellcode und öffentlich verfügbaren Werkzeugen, sowie das freie Verfügbarmachen von Daten, oft aus Infrastruktur, Wissenschaft und öffentlich finanzierten Projekten.

Die Dissertation untersucht drei Aspekte von Verkehrs-Mikrosimulationen anhand dieser Entwicklungslinien:

- Eingabedaten zum Verkehrsangebot
- Eingabedaten zur Verkehrsnachfrage
- Eine modulare Softwarearchitektur

Im ersten Teil werden Verfahren und Werkzeuge entwickelt, etablierte offene Datenquellen zur Verkehrsnetzen, speziell OpenStreetMap und Fahrplandaten, für die Verkehrssimulation nutzbar zu machen. Im zweiten Teil wird zunächst untersucht, welche tatsächlich offenen Datenquellen für die Verkehrsnachfrage zur Verfügung stehen oder verfügbar gemacht werden könnten. In Reaktion auf das häufiger werdende Auftauchen von Mobilfunkdatensätzen, einer Datenquelle, die gerade nicht offen ist und deren Offenheit besonders Datenschutzerwägungen fundamental entgegenzustehen scheinen, wird dann die Nutzung von anonymisierten Mobilfunkdaten untersucht. Im dritten Teil wird eine Softwarearchitektur für Verkehrssimulationen entwickelt, die sich hauptsächlich durch Erweiterbarkeit und Transparenz auszeichnet, zwei Aspekte, die entscheidend für die erfolgreiche Verwendung in arbeitsteiligen Forschungsvorhaben sind.

Contents

0. Introduction

> The disadvantages involved in pulling lots of black sticky slime from out of the ground where it had been safely hidden out of harm's way, turning it into tar to cover the land with, smoke to fill the air with and pouring the rest into the sea, all seemed to outweigh the advantages of being able to get more quickly from one place to another — particularly when the place you arrived at had probably become, as a result of this, very similar to the place you had left, i.e. covered with tar, full of smoke and short of fish.
>
> Douglas Adams, The Restaurant at the End of the Universe

A model of a traffic system consists of a model of the supply side and a model of the demand side, the thing being in supply and demand being transportation. The supply side models the means of transport a population has access to, with their static and dynamic characteristics: Where are the roads? How long does it take for people to travel on them? Are there tolls? What busses and trains go at which times? How much does a taxi ride cost? The demand side models how people use the traffic system: How many people commute, and where do they live and work? At what times, and to which additional destinations, do they need transportation? How do they decide whether to drive or to take a bus? How much are they willing to pay for reduced time spent in traffic? These sub-models and their interaction are, of course, implemented as a software system. Besides the issues of correctness of the implementation and its computational performance, if it is more than a one-off experiment, aiming to be a platform on which different researchers can conduct their research, issues of software engineering arise: How should the system be designed so that it will support use-cases of which the developer is not currently thinking? Will it be reasonably easy to learn? Does it support workflows consistent with how scientists work, or should work?

The main source of motivation for this work, and what ties this dissertation together, is my enthusiasm for transparent, open-source, software. I take little notice of software that is not on GitHub. When I hear about a piece of software, perhaps in a conference talk, I want to try it *right away*, give it some input, see what it does, change some parameter, and try again. I am most likely to build trust in it when what it does can essentially be explained by executable examples, with movable parts, of how to program its outermost layer, much like every new web library or application programming interface (API) does it, in the distinct current style of a one-page site featuring short code-examples in large green-on-black text.

Research software is often very much not like this. The agent-oriented modelling sub-community with their downloadable and executable NetLogo models comes close, but

these are typically designed to argue a single point, not to be extensible by programming, or to be run on input more complex than a few numerical parameters. A typical data-driven simulation software will still more likely resemble a custom business information system rather than an API: grown over the years, hundreds of parameters, non-standard way of compiling, installing and running the software, sometimes requiring additional systems such as databases to be set up, and, most importantly, not be designed to be adapted to new use cases other than by understanding and then changing the code, in place, presuming that the code is, in fact, available. MATSim was already different (and open-source) when I started, but my ambition was to push it even more towards the ultimate goal of full library-hood.

To try out a data-driven simulation, one needs a data set. MATSim includes toy data sets to demonstrate what it can do, and example code which uses them. But the data behind actual studies in transportation has traditionally been proprietary. Supply-side data would be procured from the local administration and transport service providers. Demand-side data would come from pen-and-paper household surveys.

Contrast this with the world of API and open data. When I started to work with the MATSim team, volunteered geographic information (VGI) was starting to get traction, most prominently in the form of OpenStreetMap, but also in the form of location sharing services such as Google Latitude, a web API for accessing one's own movement data recorded with a smart phone, which has since been discontinued, and the then-novel Foursquare/Swarm service, which combines a user-editable database of points of interest (POI) with a game where users were encouraged to report their presence at POI whenever they would visit them. Public transport providers outside of Germany were slowly beginning to publish their schedule in a common file format, without restrictions, to the web.

I already knew OpenStreetMap, and the MATSim community was already using it, so I quickly took an interest and started to co-develop the software tools and processes for OpenStreetMap, as well as for the General Transit Feed Specification (GTFS). At the same time, I started experimenting with my own movement data on Google Latitude and felt that at least the way of accessing this data was a lot more like what I had in mind than processing custom-formatted household surveys procured under non-disclosure agreements, and also the mode of acquiring the data seemed promising: Informal, crowd-sourced, volunteered, web-based movement data with the potential to become available more frequently and inexpensively than government-run pen-and-paper surveys.

Then, it happened that a mobile phone dataset was made available, with comparatively few restrictions, by a mobile phone operator in a developing country, in the context of a data challange called Data for Development (D4D). In contrast to VGI, mobile phone data is opportunistically collected. The users do not know they are participating. Instead, the spatial and temporal resolution of the data is restricted as to be acceptable from a data protection point of view. This mode of data acquisition does not provide information about the user, such as age or employment status, but it presumably has lower selection bias, only on account of having a contract with the particular mobile phone provider, instead of selecting for the kind of person who owns a smart phone and would enlist in

web-based collection of movement data. This was the moment our focus shifted to this kind of data set, mainly because we had access to one, but also because it looked like such data would, in the near future, be available more frequently and at higher resolutions.

The dissertation is organized in three parts. Part I is about data on the supply side. First, I describe core OpenStreetMap concepts and how they map to concepts in MATSim, and also some software tooling for converting OpenStreetMap data to MATSim which I already found implemented when I started. Then, I make a case for using OpenSteetMap as the primary data source for MATSim studies, for which I outline a workflow. I observe that an interactive software tool for editing or augmenting OpenStreetMap data for MATSim studies is missing, and I describe the design and practical implementation of such a tool.

Part II is about data on the demand side. Chapter 3, a position statement, picks up the situation before mobile phone data appeared accessible to me. I propose to use volunteered user location data as an augmentation of, or substitute for, pen-and-paper surveys. The rest of Part II turns to mobile phone data. Chapter 4 describes our initial effort at using mobile phone data in MATSim, using the D4D data set. We study a simple and direct way of constructing travelling agents from mobile phone traces. In Chapter 5, I take a step back and analyze what it means specifically for MATSim to have its traditional input of activity diaries replaced by mobile phone traces, with an emphasis on accuracy as exemplified by the total travelled distance which results from running the model. Lacking ground truth data for the D4D case, I start using synthetic mobile phone data as a framework to evaluate methods for constructing demand. In Chapter 6, I assume a set-up where demand is directly constructed from mobile phone data which is biased and incomplete. This initial, low-confidence demand model is then updated by using a set of link counts as assumed additional observations, using the degrees of freedom which are available due to the spatio-temporal uncertainty of the movement data, as well as introducing individual weights for the constructed agents. First, we show with a toy scenario how the spatio-temporal information contained in the mobile phone traces, even when sparse, can inform the simulation model better than a set of origin-destination flows or path flows. Second, we apply the method to a scenario with a more realistic network and population. In Chapter 7, we aim at a more accurate construction of travel plans from mobile phone traces. We acknowledge that simply constructing trip chains from consecutive mobile phone records will underestimate travel on the population level, and when this is corrected by scaling up weights, it will still underestimate travel on the individual level. We then augment our approach by adding trips for which there is no direct evidence, using the most accurately observerved individuals as templates. We consider a simple method to distinguish activity locations from observations made while travelling. Finally, we conjecture that within the MATSim framework, where individual and aggregate data are fused, even simple methods of constructing trajectories from traces can recover much of the information contained in the data.

In Part III, I turn to the software engineering aspect. In Chapter 8, as a case study of what it can mean to extend our simulation software, I describe the integration of freight traffic, a project from the early time of my involvement with MATSim. Chapter

9 then describes a major refactoring which was undertaken to meet the challenges of maintaining a high-quality, transparent, versatile software package as the common core of several evolving research projects.

The three parts mostly stand for themselves and have their separate conclusions.

Part I.

Supply

1. OpenStreetMap for traffic simulation

Parts of the material in this chapter were previously published in Zilske et al. (2011).

1.1. Introduction

Simulating a traffic system requires a formal specification of its supply side, which means the road network and the transit operation.

Authoritative road network models come from government agencies. Authoritative transit schedules come from transit operators. Sometimes authoritative datasets do not exist, especially in developing countries. When the datasets do exist, their maintainers are often not inclined to give them away. Getting them to hand over the data for purposes of formal research may requires a formal joint project, or, at the very least, personal contact and negotiation. Sometimes they treat it like a trade secret. Sometimes they treat it like a trade secret even though it is theoretically already in public knowledge under its intended use: Digital transit schedules are meant for consumers so they can plan their trips, and by means of repeated elementary queries, the full schedule can be retrieved from a trip planning service. Still, the German railway obfuscated the data in their CD-ROM based trip planning product to make it somewhat (Kreil, 2012) more difficult to be used for anything else.

Also, such datasets are maintained locally, by local authorities. Covering a territory with several local datasets requires merging them. Merging graph-based datasets is difficult, even when they come in the same format, which they often do not.

The road network and the transit operation are linked: Busses use road infrastructure. But the road network model and the transit schedule model are typically not linked, especially since they come from different sources. Sometimes, even the road network model or the transit operation model on their own consist of different bits from different sources, as when different sub-authorities are responsible for roads and for signal systems, or different departments of a transit company are responsible for planning and operations, and they use different database identifiers for the same entities, so there is no linked model of which traffic light is on which junction, which transit vehicle serves which line run, or which way a bus goes.

All of this is not satisfying.

Providers of car navigation systems have solved the problem of spacially isolated road network data for themselves. They have datasets which are integrated enough to allow automated door-to-door navigation for trips spanning whole countries or continents. Their business model is based on these datasets, and they have little incentive to provide them for free. Unlike, arguably, public transit operators, they do not not have a public

obligation to do so.

Google has since put forward an open data format for transit schedule data, the General Transit Feed Specification (GTFS)[1]. They ask transit authorities to deliver their schedule in this format to have it included in their web application for multi-modal routing, integrated with their popular Maps application.

The number of local transit operators in Germany who have since gone one step further and released their authoritative transit schedule to the general public for general use is three.[2]

Meanwhile, the OpenStreetMap (OSM) online mapping community (Haklay and Weber, 2008) has refined their collaborative product. The phenomenon has gained academic and industrial attention and is now more generally known as Volunteered Geographic Information (VGI, see Goodchild (2007)). The data in OSM can be used free of charge and is instantly available.

One of the most salient features of OSM is that its thematic scope is vast. Besides roads, it can include anything which mappers deem worth mapping, which means that anything from postboxes and fire hydrants to topologically consistent transit line runs can be mapped, but completeness with respect to any feature varies wildly between geographical areas. The quality of the road network data itself has been favourably compared to proprietary routing data for urban areas (Zielstra and Zipf, 2010), and it has been used for commercial routing application.

In this chapter, a process is described where OpenStreetMap (OSM) is used as the primary data source for the supply side of traffic simulations. It consists of a transformation of the schema-less data model used by OSM (a design consequence of its thematic openness) to a MATSim road network specification. This tooling allows for a workflow where an operational simulation scenario can be updated with changed or additional input data without starting over. In the subsequent chapter, we describe the implementation of this transformation as an interactive software tool.

1.2. OpenStreetMap data model and tools

OpenStreetMap uses what can be called a semi-schemaless data model, where basic abstract entities and relationships needed to represent a road network are defined like in a traditional data model, but all higher level features of both entities and relationships are described in an unconstrained way (Haklay and Weber, 2008), similar to what is now known as a NoSQL database (Strauch et al., 2011). It has three types of *elements*, which are *nodes*, *ways* and *relations*. Each element has a unique identifier (database identity). The element types are defined as follows:

Node A node is a coordinate pair. It represents a point on the map.

Way A way is an ordered list of nodes. A node can be in zero or more ways, and it can be in a way more than once. It can represent a road, or a part of a road, or

[1] `https://developers.google.com/transit/`, accessed October 2016

[2] `https://transitfeeds.com/l/168-germany`, accessed October 2016

anything which is rendered as a line on a map. Since it can be *closed* (when the first node is the same as the last node), it can also represent a polygon.

Relation A relation is an ordered list of elements (nodes, ways and relations). Relations represent compound entities, such as a network of bicycle paths, or abstract entities, such as a railway route (as opposed to the track on which it runs). Hierarchies of relations can represent arbitrarily complex entities, such as an entire public transport system.

Every element can have an arbitrary number of *tags*, which are $(key, value)$-pairs of strings. They describe the tagged entity in a free, schemaless form. In addition, every member of a relation optionally has an arbitrary *role* string.

The basic form of distribution of OpenStreetMap data is as an XML file, commonly called `planet.osm`, currently (2016) over 720 GB in uncompressed size, and released weekly.

Overpass[3] is a server and query language for the OpenStreetMap database. Since OpenStreetMap by now contains large amounts of data not directly related to streets, a simple range query for a larger area of interest such as a city will return too much data for most applications to process. Overpass allows an application to specify a query for OSM entities in much the same way as one would use the Structured Query Language (SQL, Date and Darwen (1997)) for relational databases. The following example retrieves the complete road network of Berlin, including footpaths:

```
[timeout:120];

// Fetch area named "Berlin" to search in
area["name"="Berlin"]["admin_level"="4"] -> .searchArea;

// Fetch ways representing streets
way["highway"~"motorway|motorway_link|trunk|trunk_link|primary|
    primary_link|secondary|tertiary|minor|living_street|residential|
    unclassified"] (area.searchArea);

// Also fetch nodes in these ways
(._; >;);

// Output result
out meta;
```

The query processes within a few minutes.

1.3. Road network

Road segments are represented as ways with a `highway` tag. By convention, this tag takes one of a catalogue of values, the most common of which are listed in Table 1.1. The value denotes the significance of the road for traffic. It determines the style with

[3] `http://overpass-api.de/`, accessed October 2016

which the road is rendered on maps. There are additional values for roads which are not open for access by the public. These are generally not relevant for the purpose of traffic simulation and are not considered here.

A road is considered usable in both directions, unless it is tagged with `oneway=true`. The direction in which a oneway road segment can be used is defined by the order of the nodes within the way. Since a way can have an arbitrary number of segments, a long road can be represented by a single way, even if it is intersected many times. A way must be split if one part needs different tags than the other.

Speed limits are tagged like `maxspeed=30`, which denotes a legal speed limit of $30km/h$. Tagging of public roads with speed limits in Berlin is near universal. Roads missing this tag are mostly private service roads.

The number of lanes is tagged like `lanes=2`, which denotes two lanes overall, counting lanes for all directions, whether the road is tagged as oneway or not. There is ambiguity here. Traditionally, this tag has been used to refer to car lanes only, but this is not codified. There are extended tagging schemes which allow specifying road attributes by lane. Different speed limits on two lanes of a two-laned road would, for example, be specified as `maxspeed:lanes=50|30`.

MATSim uses a graph-based model of the road network, where nodes are intersections and edges are road segments. When such a network is constructed from OpenStreetMap, the length of a road segment is taken directly from the geometry. This is the only geometric information required for the traffic flow model. After taking note of its geometric length, any road segment between two junctions without another junction in between can be generalised to a straight line, discarding intermediary nodes.

In the queuing model of traffic flow (Gawron, 1998), the characteristics of each road segment are modeled as three parameters: free speed, flow capacity, and storage capacity.

Free speed Free speed is not identical to the legal speed limit, but the two are related. For instance, while large roads within city limits in Germany typically have a legal speed limit of $50km/h$, the average free speed is decreased by traffic signals. An average of $25km/h$ was determined for the primary road network of Berlin. During a green wave or when driving faster than the legal limit, this can at times rise to $44km/h$, but this is countered by poorly coordinated signals on other parts of the route. On the freeways (Autobahnen), on the other hand, the legal limit is often not honored. Here, we determined the empirical average free speed to be 1.2 the legal limit. In areas where speed limit zones are extensively used (mostly to $30km/h$), time which is lost at unsignaled nodes governed by right-before-left priority is often countered by driving in excess of the speed limit, leading to an overall factor of 0.8 for tertiary roads. On streets tagged as `living_street`, the prescriptive legal speed limit is walking speed, which in reality is almost never honored. In lack of better data, we assume $15km/h$. The free speed sets a lower bound for the time each car has to spend on a link.

Storage capacity Storage capacity is the maximum number of vehicles which a link can contain at any given time. When the link is full, the simulation does not add any

Highway tag	Lanes	Free speed $[km/h]$	Capacity per lane $[veh/h]$
`motorway`	2	120	2000
`motorway_link`	1	80	1500
`trunk`	1	80	1500
`trunk_link`	1	50	1500
`primary`	1	80	1500
`secondary`	1	60	1000
`tertiary`	1	45	600
`minor`	1	45	600
`unclassified`	1	45	600
`residential`	1	30	600
`living_street`	1	15	300

Table 1.1.: Link attributes used for the values of the OpenStreetMap highway tag.

more vehicles to a link. The congestion is propagated upstream. Storage capacity is calculated from the link length and the number of lanes.

Flow capacity Flow capacity limits the number of vehicles which can leave a link in each time step.

Free speed and flow capacity are characteristics of road types (Baier et al., 2009) which we identified with the values of the OSM highway tag. The parameters per `highway` tag which we used in our studies are summarized in Table 1.1. The queuing model uses directed edges, so for each road segment except for those which are tagged as `oneway`, two links are created.

The region of interest for our studies has typically been an urban area. The road network of the region of interest is extracted with an Overpass query (see Section 1.2) or by any other method of acquiring raw OSM data, such as downloading the complete `planet.osm` database dump and filtering it by means of open-source tools such as Osmosis, which permits extracting the road network of the entire region of interest by specifying a bounding box for the city, and a tag filter selecting all highway types accessible by car.

When traffic is later assigned to the network, it is helpful not to cut the network at the city limits, but to embed it into its environment by adding the primary roads of the commuting range of the city. It can then be determined by routing if and on which way a commuter passes through the city.

Certain links and nodes can be blocked from being removed during simplification, which is important if they are linked to features relevant to the simulation. For example, to calibrate simulation parameters and to validate results, traffic counts may be used. Local authorities obtain them by means of induction loops or traffic cameras. These data are typically referenced to a proprietary network used by the respective agency, or they only carry coordinates but no explicit reference to a road segment. For one of our studies, we manually referenced count stations to OSM nodes and saved the OSM node identifier

to the counts database. Where no suitable node was available, a node can be inserted into OSM specifically as a reference point. It turned out that nodes are stable enough to enable this workflow. Still, with every update from OSM it has to be verified that the reference nodes still exist, and any changes to their location have to be monitored.

1.4. Junction layout

Apart from a higher tagging coverage of attributes from which flow capacity, free speed and number of lanes can be inferred, traffic simulation applications would benefit from a detailed representation of junction layouts (Krajzewicz et al., 2015). In the queue model as described above, the flow capacity controls the rate at which cars flow out of a link. In reality, different turns at junctions have different capacities, which depend on the number of turn lanes available. If this is modeled by decreasing the capacity of certain turn relations, this raises the question of how to handle the traffic going in other directions: If cars are removed from a link strictly on a first-in, first-out basis, a car going straight will be forced to wait behind a car going left which is kept by a capacity restriction on the turn, while in reality it may well be that turning traffic has its own lane and can be passed. On the other hand, if cars going in either direction actually share the same lane, this would again be inaccurate.

A straight-forward solution is to split the link into several ones, one for each lane (Grether and Thunig, 2016). For this to work, it needs to be known how many lanes are available per turn relation, and also their length, since once a turn lane runs full, turning vehicles will spill back to the main lane. This kind of data was not yet available in OSM when Benjamin Schulz proposed a tagging scheme for this information as part of his Bachelor thesis (Schulz, 2011), where he also developed a graphical editor as a plug-in for JOSM, the Java OpenStreetMap editor[4]. It allows adding extra turn lanes to the links leading up to a node, specifying into which link each turn leads, and moving the beginning of the lane with a slider (see Figure 1.1). The turn relation can cover complex junctions consisting of multiple nodes and permits special cases like U-turn lanes. Lanes in OpenStreetMap have since taken a different development, but this work marked the beginning of our work with JOSM and sparked the idea of developing the plug-in described in Chapter 2.

1.5. Visualization

An agent-oriented traffic simulation produces trajectories of individuals. The simulation environment MATSim features an interactive visualization tool called OTFVis (Strippgen, 2009), which can display the current location of each agent during a simulation. This is an important tool for the user in order to spot regions with implausible traffic patterns caused by errors in link attributes or in the population structure. Agents are rendered as discs moving over a simple schematic view of the network graph, where links

[4] `https://josm.openstreetmap.de/`

Figure 1.1.: The turn lane editing pane in JOSM.

are lines with a width corresponding to the number of lanes. A user who is familiar with the simulated scenario will be able to identify locations from the road layout alone. However, for a detailed analysis, it may be preferable to use a properly rendered street map as a background image. As part of this work, OTFVis was extended with a background layer of OSM tiles using the `JMapViewer` component. In principle, this would also be usable if the network data for the simulation were taken from a different source. However, as with any map overlay, this would lead to inaccuracies. When the cars move on the very same map which is rendered as a background, the visual impression is much clearer.

1.6. Generating the synthetic population

The synthetic population which models the transport demand is typically generated from survey data. The available data may be as fine grained as complete mobility diaries of a population sample, containing georeferenced activity locations, durations and the means of transport taken in between (Follmer et al., 2010). However, this information is often not available. The minimal amount of data required for a meaningful simulation is home and work locations for the synthetic population. These can be drawn from a commuter matrix, which for each pair over a set of polygons contains the number of people commuting from a home location in polygon O to a work location in polygon D on a typical work day. These polygons are typically administrative areas.

Depending on the scale and the desired accuracy of the simulation scenario in question, it may be sufficient to randomly draw points from the entire administrative area and designate them as home and work locations for a person. In our simulation, activity locations are on links, so each randomly drawn point is matched to its nearest link. This approach is sufficient if the scale of the given polygons matches the simulated

network level. For example, if only commuter traffic on the primary network is simulated, locations may be drawn from cities or counties without much consideration for different land use, since the locations are snapped to the next primary road. On the other hand, if commuter data and polygons are available on the level of city quarters or even building blocks, the entire road network may be used without any ill effects, such as small roads becoming unrealistically congested because the simulation places homes where, in reality, nobody lives. In this process, OSM can be used in two different respects: First, the administrative areas themselves may be obtained from OSM. In real-life data obtained from local governments, it is interestingly not always the case that the polygons referenced by the commuter matrix are delivered in a common data format or even with a common coordinate reference system. If, however, the reference is made using a common identifier as a key (for example German administrative areas, which are multipolygon relations tagged like `de:amtlicher_gemeindeschluessel=12067137`), the shape may be taken directly from OSM, which saves writing a converter for the specific data. Secondly, OSM may help to bridge different levels of aggregation. For instance, if commuter data is only available at the level of cities and counties, but a simulation of the entire road network is desired, drawing locations uniformly is problematic, for the reasons stated above. This could be solved by employing a distribution which takes land use or even buildings into account, as available in OSM. A step in this direction is to restrict the area from which activity locations are drawn to certain values of the `landuse` tag, or to assign weights to different values, so that home locations are concentrated in areas tagged as `landuse=residential`. How to refine this, and in particular how to deal with the problem that the availability of this information may vary widely within the area of one simulation scenario, will be the subject of future work.

1.7. Summary

In this chapter, we described a workflow for generating multi-agent traffic simulation scenarios based on OpenStreetMap. For detailed scenarios, additional demand data will be necessary, but in principle, it is possible to set up a basic simulation based only on OSM and a commuter matrix. Finally, using a network for the simulation which at the same time can be rendered as a high-quality background map has substantial benefits for the visualization of results. We think that Volunteered Geographic Information will play a substantial part in making it easier to set up and maintain large-scale traffic simulation scenarios.

2. Interactive editing of integrated network models

This chapter describes an ongoing software project which originated as a Bachelor's thesis (Kühnel, 2014) co-advised by the author. Parts of the material in this chapter were previously presented by the author at the State of the Map (2016) conference.

2.1. The JOSM plug-in

The workflows described in the preceding chapter are non-interactive data conversion tasks, where serialized OpenStreetMap data are converted to MATSim input files. As part of this work, a plug-in for JOSM, the Java OpenStreetMap editor, was developed[1]. It allows previewing the MATSim network model resulting from the loaded OpenStreetMap data while editing. It uses the software engineering concept of data binding, which is ubiquitous in recent frameworks for graphical user interface development such as JavaFX (Weaver et al., 2012) for Java desktop applications, and AngularJS for web applications (Kozlowski, 2013). The OpenStreetmap (source) data model loaded into the editor is monitored for local changes resulting from interactive editing steps. The MATSim (target) data model is updated with relatively inexpensive local operations. This allows, for instance, dragging a node with the mouse while the length values of the MATSim links incident to the node is updated continuously in a table view of all links (Figure 2.1).

OpenStreetMap data for the region of interest can be directly fetched from an Overpass server (see Section 1.2). The bounding box is selected by mouse-dragging over a slippy map. The plug-in knows which OpenStreetMap entity types it can process, so a relevant query filter is pre-set without any required action by the user. Filtering is necessary because the topical domain of OpenStreetMap has expanded beyond actual streets to a point where an unfiltered bounding box query of any urban area has become impractical. With a filter for `highway` and public transit entities, downloading the entire Berlin network from Overpass happens in tens of seconds.

Using an interactive editor and being able to expect a stable mapping from the OSM model to the MATSim model may encourage users to use OSM as the primary data format. This means that instead of initially producing a MATSim network file from OSM and then, during the course of a project, editing the network file, one would edit the OSM data themselves. Conversion to the MATSim data model would happen ad-hoc, upon start of a simulation run.

[1] `https://github.com/matsim-org/josm-matsim-plugin`

Editing OpenStreetMap data is only a full substitute for editing a MATSim network if a user can emulate every modification of the target network by a corresponding modification to the OSM data. We introduced new OSM tags corresponding to the properties of the target model, such as flow capacity. The prefix `matsim:` was used to identify these tags. When these tags are found on an OSM entity, they are applied to the resulting MATSim entitities, overriding the attribute value which would be computed by other conversion rules. Note that these tags are not intended to be committed to the OSM database, but to remain in the local branch kept for the modeller's specific project. JOSM enables merging local changes to the data while updating from the server.

Even the desired identifiers of target entities can be specified, using the `matsim:id` tag. Since MATSim identifiers have to be unique and OSM tag values cannot be forced to be unique, this makes it necessary to introduce a global validation step, which ensures that target identifiers are unique before the network can be saved. This is notably the only aspect of the conversion for which this is the case: Up until this point, the validity of the model could always be determined locally. For instance, a way with tags which are uninterpretable by the converter is simply not converted to links. Interactively, if the user adds an uninterpretable tag to a way, such as an unknown-valued `highway` tag, the corresponding links will disappear from the model, leaving a validation warning behind, and upon an undo operation, they will reappear. For reasons of simplicity and clarity, this approach was not taken for identifier uniqueness. Instead, unique temporary identifiers are used for all MATSim entities during editing, which are changed to target identifiers when the model is saved to a file, after validating for uniqueness.

When the MATSim plug-in is enabled, a native MATSim network file can be loaded, edited and saved within JOSM. Upon load, it is converted to the OpenStreetMap data model, using a particular normalized sub-set of its modeling capabilities. Every link is converted into one way, so that all ways have exactly two nodes. Nodes and ways are tagged with `matsim:` tags only (no `highway` classification is done). Loading and directly saving a network is idempotent, and interactive operations are executed as locally as possible. This means that the plug-in can be used as a practical editor of native MATSim files, as an alternative to opening the MATSim network XML file with a text editor. If only a single link attribute is to be edited, then the rest of the file will not change.

This means that by mapping the MATSim data model to the OSM data model, as a by-product, a practical graphical editor of MATSim network files has been created, with features such as full undo and redo, extensibility in the Java programming language, programmable validation and quick-fixing of data errors, and interactive merging of changes made by different collaborators.

2.2. Public Transport

2.2.1. Public transport at different scales

If the focus of a traffic simulation is on private cars, user switching to public transport can be modeled as removing themselves from the road network and experiencing a travel time

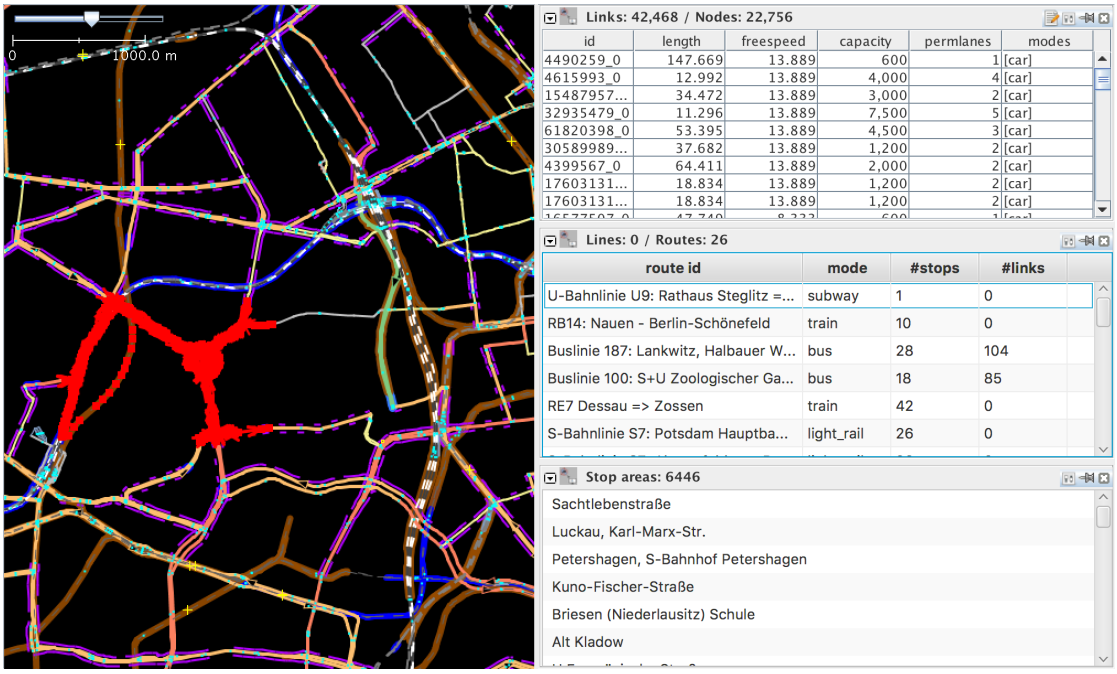


Figure 2.1.: The JOSM plug-in.

estimated by aggregate characteristics of the transit system, for example twice the time required by car on an uncongested network (Rieser et al., 2009). This simple approach cannot take into account the varying transit accessibilities of different locations.

An iteratively more fine-grained approach is possible if the following things are known:

- locations of transit stops
- transit routes, possibly with travel times between stops
- complete transit schedule with all departure times
- line runs through the road network
- vehicle allocation

The locations of transit stops alone can be used to make a more educated guess about the public transit travel times than if no information at all were available. The travel time from location A to location B can then be estimated by adding up an estimated walk time from A to the nearest transit stop, T_A, an estimated transit network travel time from T_A to T_B, the nearest stop from the destination B, and an estimated walk time from T_B to B (Nagel, 2016).

If a transit schedule is available, schedule-based transit can be simulated (Rieser, 2010). In this case, agents plan their trips through the transit network, wait for the next departure and arrive at their destination stop when the simulated vehicle does. The simulation

knows which agent plans to use which transit route for which trip. It can take vehicle capacities into account, as well as access and egress times. This means that effects like overcrowded vehicles, where passengers can be denied boarding and have to wait for the next departure, can be simulated, and vehicles are delayed if a larger number of passengers are boarding or leaving than the schedule accounts for. If a fare structure is known, the precise monetary disutility of the ride can be subtracted from the score so the trip can be more accurately weighed against using a private car.

While some public transport modes, like railway, normally operate without interaction with car traffic, this does not hold for buses, which can be caught in congestion behind private cars. Conversely, private cars may have to wait behind buses when passengers are boarding or alighting. If it is desired to simulate this kind of interaction, one requires the exact routes the vehicles take through the road network.

Since a public transit vehicle serves several line runs, often on a tight schedule, delays can accumulate. If the simulation is expected to pick up such effects, the data requirements exceed what can be expected to be acquired from public sources. It requires internal operational information from the network operator.

While OSM maps transit stop locations, transit routes and line runs, it does not map departure times, since it is a policy of OSM to only map visible entities.

With the General Transit Feed Specification (GTFS), developed by Google, there is a data format available which contains all the required information. When we started looking into the GTFS format as an input format in 2011, not one transit authority in Germany had published a dataset. Any study including public transport at schedule-level up to then required coordination with the local transit authorities, and manual data conversion. Shortly afterwards, datasets for the Berlin-Brandenburg region and for the city of Ulm were published. Later, a dataset for the Rhein-Neckar-Region was published, so that the number of official datasets in Germany is now three.[2]

Using OSM only, the best one can hope for (assuming the most complete level of tagging) is a representation of all transit lines, where buses drive on the road network, but without departure times or frequencies, which must be guessed.

Using GTFS only, the best one can hope for is a representation of the transit network at the level of a typical route planning application. Agents will plan their trips according to published departure times and published line switch times. The drawback is that the transit network is not connected to the road network, so interaction between transit vehicles and other traffic is not possible.

In order to achieve both, line runs must be taken from OSM, the schedule itself from GTFS, and routes and stops in OSM must be matched to routes and stops in GTFS. In general, GTFS data and OSM data are not cross-referenced. As of 2016, searching for tags or tag values containing `gtfs` yields only two clusters of tagged entities world-wide, one in Israel and one in Australia, and they use different tagging schemes.

[2] `https://transitfeeds.com/l/168-germany`, accessed October 2016

2.2.2. Public Transport in OSM

In April 2011, a proposal for a Public Transport Feature passed the OpenStreetMap voting process. It defines, among others, the following entity types:

- Route direction/variant
- Route master
- Stop position
- Platform
- Stop area

The distinction between stop position and platform is important:
"The stop position is the place where the vehicle usually stops on the rails or on the street. [...] The platform is the place where passengers are waiting for the vehicles."[3]

In particular, this means that a platform node will generally not be a part of the road network graph, but will be positioned next to the road, while the stop position will be a node which lies on the road. The platform potentially plays a role in the user-facing part of the simulation, where users travel to the point where they enter public transit. It can consist of as little as only one isolated node, for example when it represents a bus stop pole.

The stop position is used for the vehicle flow simulation and determines on which link and at what position on that link a vehicle will stop and potentially cause car traffic queuing up behind it.

In practice, this detailed schema is not commonly used according to the specification. For example, route relations will often not contain continuous ways as members and are thus not interpretable as routes of a vehicle through the road network. In the previous chapter, we defined a mapping from the schemaless OSM data format to the relational MATSim data format for road networks. Repeating this exercise for the transit schedule is complicated by the magnitude of inconsistencies in the tagging, which are probably due to the complexity of the tagging scheme.

I tagged and committed to OSM a `route` according to my interpretation of the full specification. It is shown in Figure 2.2 as a data sample. The list of relation members has two parts. First, the line stops are encoded as an alternating sequence of `stop` and `platform` members. Second, the line run on the road network is encoded as a sequence of Ways which are connected end-to-end. This is detected by JOSM and visualized in the right-hand column.

It was decided that the generic relation editor provided by JOSM (as shown in Figure 2.2) was good enough as an editor of routes, and a special editor for this plug-in was not necessary. While it does not enforce any part of the public transport specification, such as

[3] `http://wiki.openstreetmap.org/wiki/Proposed_features/Public_Transport`, accessed October 2016

Role	Refers to	
stop	S Schöneweide/Sterndamm (52.4537291, 13.509588)	
platform	S Schöneweide/Sterndamm (2 nodes)	
stop	Nieberstraße (52.4483925, 13.5121178)	
platform	Nieberstraße (2 nodes)	
stop	Engelhardstraße/Pilotenstraße (52.446727, 13.5111919)	
platform	Engelhardstraße/Pilotenstraße (2 nodes)	
stop	Segelfliegerdamm (52.4446987, 13.5107152)	
platform	Segelfliegerdamm/Waldstraße (2 nodes)	
stop	Sterndamm/Königsheideweg (52.4462591, 13.5071881)	
platform	Sterndamm/Königsheideweg (2 nodes)	
stop	Haushoferstraße (52.4476111, 13.5001667)	
platform	Haushoferstraße (2 nodes)	
stop	Ostweg (52.4489592, 13.4957273)	
platform	Ostweg (3 nodes)	
	highway (6 nodes)	
	highway (2 nodes)	
	Sterndamm (7 nodes)	
	Sterndamm (2 nodes)	
	Sterndamm (2 nodes)	
	Sterndamm (5 nodes)	
	Groß-Berliner Damm (18 nodes)	
	Pilotenstraße (9 nodes)	
	Engelhardstraße (8 nodes)	
	Segelfliegerdamm (6 nodes)	
	Königsheideweg (15 nodes)	
	Königsheideweg (2 nodes)	

Figure 2.2.: Relation members of a route in JOSM.

alternating stops and platforms, or gap-free line runs, it does give enough visual feedback for a person familiar with the specification to validate it by eyeball, and possibly repair it. More complex constraints, notably that the `stop` nodes must lie on the network route, are checked by a validator, which runs before data is uploaded to OSM. At this point, this validator is provided by the MATSim JOSM plugin, but it is planned to change it to the one provided by Golovko, 2016 (also see Sec. 2.3.2).

2.3. Transforming the OSM public transport schema

MATSim contains support for detailed, schedule-based public transport, where the travelling agents can produce and evaluate their public transport alternative based on real stop locations and line runs using a router, and where the vehicles are incorporated into the regular traffic flow alongside individual traffic (Rieser, 2010). Its input data model is shown in Figure 2.3. OSM entities are converted to MATSim entities according to Table 2.1.

Note that the `TransitRoute` class contains a field for a `NetworkRoute` which encodes the line run on the infrastructure network. Also note that the `TransitStopFacility` class contains a field for the a reference to the unique link in the infrastructure network to which it is assigned. It is used by the traffic flow simulation. It is the link where the transit vehicle stops, and where passengers can embark and disembark.

A `route_master` is only supposed to be used when there are at least two `route` variants, so it is possible to have a `route` without a master, and in this case, the `TransitLine`

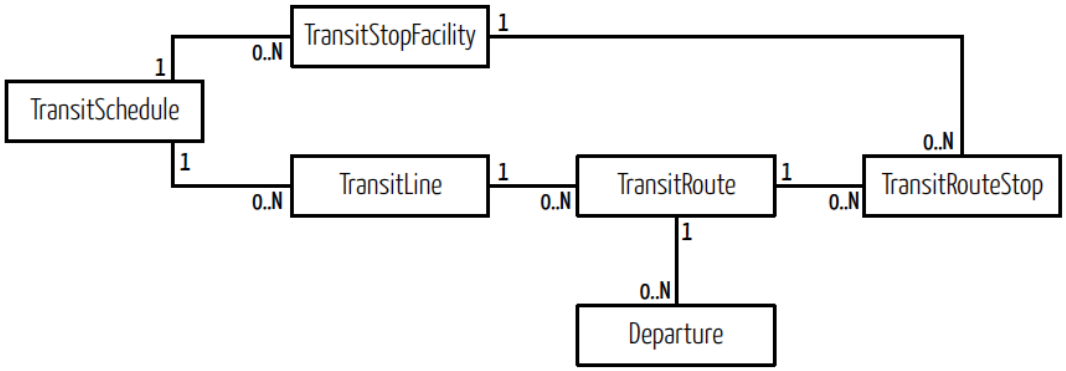


Figure 2.3.: The MATSim data model for public transport. Image taken from (Rieser, 2010).

OSM	MATSim
`route_master`	TransitLine
`route`	TransitRoute
`stop_area`	TransitStopFacility
`stop` and `platform` members of a `route`	TransitRouteStop

Table 2.1.: Entity mapping between OSM and MATSim.

containing it is implicit. The `stop_area` is optional according to the specification. This means that it cannot be relied upon to establish the connection between the `stop` and the `platform` in a `route`. Neither can it be established simply by looking at the alternating sequence, since both `stop` and `platform` members are specified as "recommended if available", which can be read to imply that in the worst case the availibility of `stop` and `platform` members can be mixed. We decided to implement a tightened version of the specification, so that `stop_area` relations are mandatory, and there is a 1:1 mapping to `TransitStopFacility` instances (before a splitting step to require a limitation of the current version of the MATSim data model, see Section 2.3.1). Note that there is no OSM counterpart to the `Departure` entity, since departure times are not within the scope of OpenStreetMap. As stated earlier, they must be acquired either from a different data source, or programmed based on estimated service frequencies.

We developed two data conversion schemes, nick-named *light transit* and *full transit*, and integrated them into the JOSM plugin. The main difference is about what is done with the `NetworkRoute` reference of `TransitRoute`. This is explained in the next two sections.

2.3.1. Light transit

Here, it is assumed that the network-based line runs encoded in the `route` entities are absent or unreliable, or that network-based public transport is simply not a requirement.

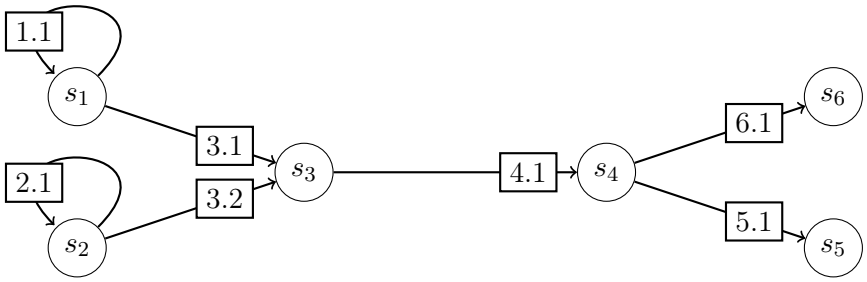


Figure 2.4.: Network model for *light transit*. Adapted from Rieser (2010, Figure 8.7).

For each `stop_area`, a `TransitStopFacility` is placed at the location of the `platform` if it is available, and at the location of the `stop` otherwise. The line run in each `route` entity is converted by creating a `TransitRouteStop` entity for each distinct `stop_area` referenced by a `stop` or a `platform` member. The `NetworkRoute` reference fields are left empty.

The resulting partial `TransitSchedule` is then given as an input to the core MATSim class `CreatePseudoNetwork` (Rieser, 2010, Section 8.1.4), which is designed to handle just this situation. It creates a network consisting of one node for each instance of `TransitStopFacility`, and one link (s_i, s_j) for each pair of stop facilities s_i and s_j which are neighbors in some route, with additional links (s_j, s_j) for stops s_j which come first in some route. It then splits each `TransitStopFacility` instance into several copies, one for each of its unique predecessors, and assigns each copy to its corresponding link. Note that this lets every vehicle serve its first stop on a loop link. This means that if a link has a stop on it, it can consistently be regarded as being near the end of the link, which is an assumption of the current implementation of the MATSim traffic flow simulation. Figure 2.4 illustrates the model.

This network is now merged with the regular street network. The two components, the street network and the transit network, are disconnected. Cars do not use the transit network, and travellers who switch modes during the day transfer between the two network components by a non-network mode, typically by walking, with a scaled beeline distance used as the travel time.

The link attributes of the transit network links are set to default values. MATSim did not originally support vehicle-based maximum velocities, so the free speed attribute of the transit network links would control the speed at which the vehicles would drive, assuming uncongested conditions on the dedicated transit links. It should be set to an estimated upper bound on the average speed of transit vehicles between stops. The public transit simulation code itself ensures that transit vehicles will wait at stops for their scheduled departure times and not run ahead of their schedule.

This procedure will generally only makes sense where separate transit schedule data are not available in the first place. The modeller will have to use estimates to code an approximate headway-based schedule. If a GTFS dataset is available, all the *light transit* information, i.e. departure times, routes and stop positions, can be taken from there

and OSM will only be used for the street network. The data conversion from GTFS to MATSim is straight-forward, and we use the same `CreatePseudoNetwork` algorithm as described above. A practical implementation has been developed as part of this work.[4] A simulation scenario which uses it is described by Kickhöfer et al. (2016).

2.3.2. Full transit

Here, it is assumed that the `route` fully encodes the network-based line run. We do not attempt to heuristically repair incomplete or inconsistent line runs automatically, as part of the data conversion. Rather, we observe that consistency checks and interactive editing tools for the public transit schema are available (Golovko, 2016), and assume that a user aiming to set up a full transit scenario will, in preparation, consistently map the line runs with the help of these tools. The transit lines are then converted, verbatim, to the MATSim data model.

Stops are treated very similarly to the simplified approach of Section 2.3.1. At first, one `TransitStopFacility` per stop area is created. In a post-processing step, `TransitStopFacility` instances are split so that one copy exists for each link from which it is served.

Note that some special treatment appears to be indispensable. Even if every stop area would consist of exactly one (`stop`,`platform`) pair, it would remain neccessary in certain cases. The reason is that, in OSM, stops are nodes, whereas in MATSim, stops are features of links. The simplest realistic example would be an intermediate stop of a ferry line. Vehicles in both directions use the same stop position and the same platform, but they arrive on different links.[5]

Not insisting on a 1:1 mapping between `TransitStopFacility` instances and any OSM entity class has an additional benefit. Since it is not strictly specified what a stop area is, it may consist of any non-matching number of `stop` and `platform` instances. The matching between these two entity classes is only implicitly given by routes using a certain (`stop`,`platform`) pair. Even that is optional, since it is allowed for a route to use only a `stop` or only a `platform` entry. There remain corner cases where the splitting operation must operate heuristically. Identifying a stop area with a `TransitStopFacility` before splitting, and having the splitting operation ensure that every route can serve the stop area, is a convenient way to handle this situation. Stop areas can be hierachical. If mappers are dissatisfied with the result of the splitting operation for a complex stop area (in particular, which platform location is assigned to which link), they can simply split a stop area into several sub-stop-areas to resolve ambiguity, but our scheme conveniently does not require this.

[4] `https://github.com/matsim-org/GTFS2MATSim`

[5] For a different solution to the same problem, in the context of airport modelling, see Grether et al. (2013, Figure 1a).

2.4. Matching OSM and GTFS

There appears to be agreement in the OpenStreetMap community that GTFS is the de-facto standard for transit data, and that the community should find procedures to integrate GTFS sources with OSM[6]. Yet a search for tags containing the substring `gtfs`[7] reveals a very sparse use, in the order of one percent of the use of `public_transit` tags.

There is an open source software tool called GO_Sync (Tran et al., 2013) which takes a GTFS feed, looks up the enclosing area in OSM and tries to establish a match between GTFS entities and OSM entities based on tags. Where that cannot be established, a GUI-based process is started to allow a user to link GTFS stops to OSM stops using candidates which are selected based on proximity. Functionality for linking routes appears to be limited or work in progress, and it uses the older OSM tagging scheme for public transit.

The missing ingredient for a full transit simulation, where line runs are taken from OSM and timetables from GTFS, would be a tool which facilitates semi-automatic one-to-one matching of stops and routes between OSM and GTFS. For confirmed matches, the OSM entity would be tagged with a reference to the corresponding GTFS entity. After converting the stops and routes from OSM to MATSim, departure and arrival times for each stop in a line run could then be taken from GTFS. The GTFS identifiers of routes and stops cannot be expected to be stable from one version of a GTFS dataset to the next. But neither can the routes, line runs and stop posistion themselves. Keeping a public transit network in OpenStreetMap up to date and consistent is a large effort either way, which may be why an example where this is actually happening is hard to come by. Additionally maintaining the OSM-GTFS link is probably not critical.

Specifically for using GTFS data with MATSim simulations, there are alternative approaches from the MATSim community. A software by Ordonez and Erath (2011), takes a GTFS dataset and a MATSim road network and produces a MATSim public transit file. Like our implementation, it is freely available. Unlike ours, it is agnostic about the source of the road network and does not use information from OpenStreetMap. It map-matches the stop sequences and the (optional) line run geometries from GTFS to the road network and allows the user to check, repair and save the automatically proposed line runs. Since the tool is specific to MATSim, the result of that effort cannot be committed back to OpenStreetMap.

2.5. Summary

The JOSM plugin for MATSim described in this chapter was developed for this work and for the Bachelor's thesis of its second developer, and it has been committed to the JOSM plugin repository, making it accessible directly from the JOSM software.

It allows interactive editing of OpenStreetMap road networks with an instantaneous preview of the simulation-related properties of each link. Since these properties are derived from a combination of OSM highway tags, user-configurable parameters and

[6] see `http://wiki.openstreetmap.org/wiki/General_Transit_Feed_Specification`

[7] `https://taginfo.openstreetmap.org/search?q=gtfs`, accessed October 2016

custom MATSim specific tags, our contribution can also serve as a visual debugging tool for network flow simulations.

The public transport tagging scheme for OSM has been criticized as being too complex for casual mappers.[8] While it may be possible to render on a map a route which is less than fully consistently tagged, a full transit simulation requires a consistent relational model of routes, links and stops. Tools like Public Transit Assistant, also a JOSM plug-in, already facilitate this, though in 2016, consistently tagged routes remain sparse. Our contribution is a converter from OSM to MATSim for such fully consistently tagged routes. We decided against heuristically filling holes in the model during data conversion. Interactive or automatic repair tools for OSM data can be developed independently. When a modeller uses such tools, the repaired data can thus be contributed back to OSM. Our converter comes embedded in our JOSM plug-in. The modeller can preview the simulated line runs from the visual editor.

We hope that our contribution will enable a future MATSim modeller who wants to create a full transit simulation for a city to do so by contributing fully consistently tagged public transit routes for that city to OSM, rather than making the neccessary changes in the MATSim data model. In the spirit of open source, such efforts can thus benefit other applications as well.

[8] As can already be seen from the discussion on the public transit feature page cited above. Also, there is already a proposal for a simplified version, but it has been in draft stage for several years.

Part II.

Demand

3. Towards volunteered digital travel demand data

A previous version of this chapter was published as Zilske and Nagel, 2012.

3.1. Introduction

Behavior-based transport modeling typically relies on travel diaries from a sample of the population of the study area. A travel diary is an account of all activities that a person performs within a day, with locations and times, and of all the trips taken between locations, with mode of transport. Such travel diaries are conventionally acquired by means of pen-and-paper questionnaires. This method carries the problem that many trips are not reported because they are deemed too unimportant by the subject to warrant reporting. This can be improved by using specialized data collection devices equipped with GPS receivers, which also allow data entry at the time activities or trips are commenced or finished (Auld et al., 2008; Battelle Memorial Institute, 1997; Wolf et al., 2004b). By doing plausibility checks during data entry, some errors can be corrected semi-automatically (Kochan et al., 2010).

There have also been studies using completely non-interactive collection of GPS data with the aim of estimating all considered attributes of trips and activities from the trajectory. In one large-scale study (Wolf et al., 2004a), trip attributes were estimated based on land-use indicators and demographic data. Since no additional information about the trips was available, the results were statistically compared with classical mobility surveys.

Both approaches have drawbacks: Being required to enter data before and after each trip is considered a burden, even if it is assisted by an intelligent device, while with the analysis of passively obtained trajectories, important attributes of trips and activities can only be statistically estimated.

A synthesis of these approaches is the prompted recall survey. Passively collected trajectories are analyzed, visualized, and presented to the subject for validation and completion. In newer such studies (Auld et al., 2008; Doherty et al., 2006), this has been done by means of Internet applications. Survey participants upload the acquired trajectories on their own and are immediately prompted to answer follow-up questions about them. Delays from letter delivery or manual preprocessing are eliminated. The Internet application determines times and locations of activities from the trajectory. They are presented to the participant, who can then remove badly recognized activities and add missed activities. There are also algorithms for an automatic recognition of

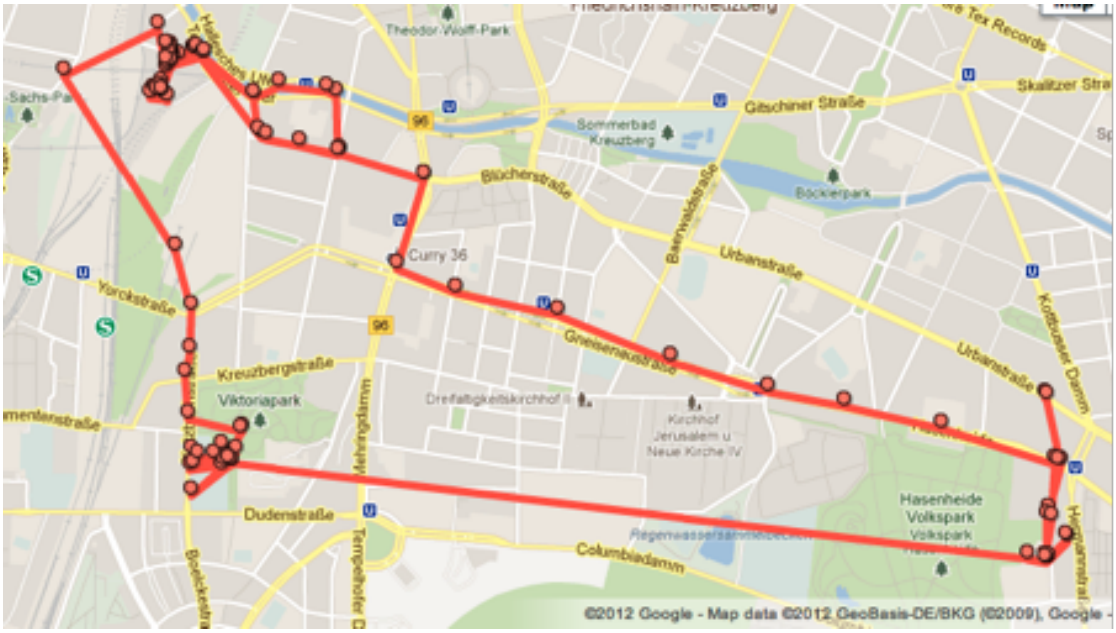


Figure 3.1.: One-day trajectory of the author as recorded by Google Location History.

the means of transport (Chung and Shalaby, 2005) and for the trip purpose (Bricka and Bhat, 2006; Wolf et al., 2001).

3.2. Travel surveys with commodity hardware

The GPS devices commonly used in travel surveys are not consumer products. They must be acquired from the study budget, distributed, and explained to the participants. The massive proliferation of smartphones seen in recent years raises the question if this is still necessary. In contrast to specialized GPS devices, smartphones are already available and familiar to a rising number of people. The familiarity extends to using location-based services (Mascolo, 2010). Smartphones are often equipped with GPS receivers, but can also locate the user by the alternative means of WiFi network signatures and the mobile phone network, which conserves the battery and, in contrast to GPS, also works inside buildings. Moreover, with services such as Google Location History (formerly known as Latitude), there are application platforms which let users acquire their movement history, share it, and even edit and analyze it. To the knowledge of the author, there is only one publication on the acquisition of behavioral data from Google Latitude (Ferrari and Mamei, 2011), where it was shown that daily routines ("usually goes to sports club X in their lunch breaks") could be reconstructed from data acquired over a large timeframe. The Location History application is pre-installed on Android handsets. It records the current location of the user at an adaptive sampling rate (Figure 3.1). Running in the background, it still admits the typical smartphone charge cycle of 24 hours.

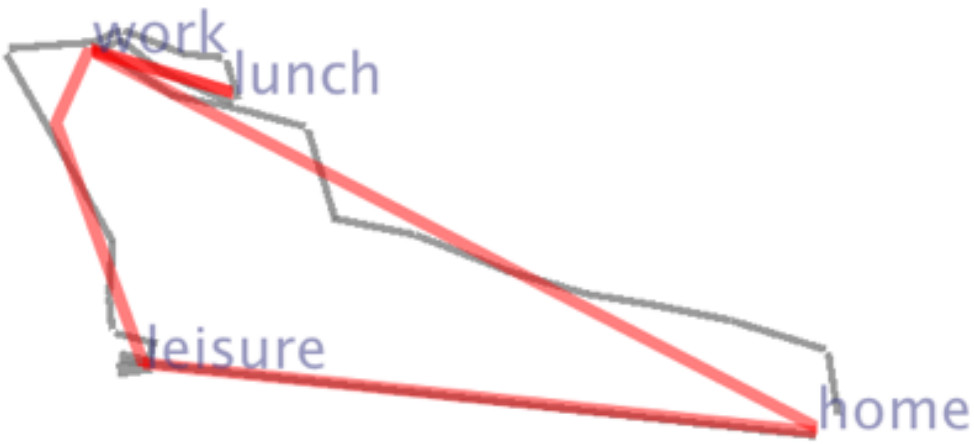


Figure 3.2.: The spatial structure of a travel diary (red), overlayed with the trajectory (gray).

Programmatic access to the Location History API for software developers has since been discontinued. When it was still available, we developed a prototype surveying tool, where a user could browse their daily trajectories, correct them, annotate them with activity types, and donate them for transportation research. The goal was to show that the participant will in many cases only have to acknowledge the correctness of the items in a schematic representation of a travel diary (Figure 3.2) without making changes.

In this figure, the red polyline connects the activity locations, which were determined by simply clustering consecutive recordings which lie within a small area, and then tagging clusters where more than five minutes were spent as activity locations. The activity types were added manually. Note that while all activity locations of this day were indeed identified, the trace contains one "false positive" (the leftmost red vertex, without label). Our proposed system consists of a user interface which admits efficient editing of such a suggested travel diary[1], and a supervised learning algorithm which quickly improves the quality of suggestions. It has been demonstrated, for example by Liao et al. (2007), that a machine learning model trained with a set of labeled trajectories can be used to label trajectories from other people with high accuracy. Note that for the purpose of travel diaries, a detailed reconstruction of routes through the road network is not necessary. The more modest but still challenging task of determining activity locations and modes of transport is sufficient. When used in place of a pen-and-paper survey, the approach

[1]The Location History web application, which has since been renamend again and is now called Timeline, now has this feature. Inferred activity locations and modes of travel can be changed by the user, which influences the inference for the next visit.

described here may lead to better data since there will be less reporting bias: The user will not have to recollect the travel diary from memory, but will be presented with an initial suggestion which may contain trips which would otherwise have been forgotten. In that way, it resembles a GPS-based prompted recall survey. For government-initiated surveys, building trust may be an issue. This may better be achieved using a smartphone and a transparent Internet platform with which the user is already familiar, than with a GPS device that most people have not seen before. In particular, in the medium term it will be necessary to address privacy concerns by fuzzifying certain locations directly at the source. This process would be made visible to the participant by presenting a preview of the final data before it is handed in for the survey.

3.3. Voluntarily contributed travel diaries

Apart from being used as a surveying tool, this technique could be developed into a system which can continuously acquire travel diaries from the public at large. The idea is to leverage the existing familiarity with the smartphone platform and create game-like incentives (McKenzie, 2011) for the voluntary donation of mobility data. Web-based tracking services for aspects like sports activities, nutrition and even mood are becoming popular, and they are being used because these platforms create value for their users. In our case, we would begin with giving users a structured presentation of their own mobility behavior, including amounts of time spent for different types of trips, and offer alternatives, like switching to public transit. Voluntarily contributed trajectories will be biased: It is to be expected that only a certain type of personalities will participate, and that they will select data only from certain days. At present, this is not found to be a sufficient deterrent to abandon this line of research. Data from voluntarily contributed trajectories could be fused with other data, such as cellphone data from mobile phone operators, or traffic counts, or aggregated time use surveys. The time use survey provides unbiased but non-localized activity patterns, while the cellphone data would lead to an initial origin/destination pattern. The data investigated in this chapter could presumably be used to enrich some of these patterns, and traffic counts could be used to correct for biases (see Flötteröd et al. (2011a) for the theory, and Moyo Oliveros and Nagel (2012) and Flötteröd et al. (2011b) for first applications).

4. Building a minimal traffic model from mobile phone data

A previous version of this chapter was published by Zilske and Nagel (2013). That paper was submitted within the context of the so-called D4D (data for development) Challenge (Blondel et al., 2012). In that challenge, a large mobile phone provider provided mobile phone call detail records (CDRs) from a developing country, and the challenge was to make this data useful for development related purposes.

4.1. Introduction

Transport planning is a multi-faceted exercise, necessitating many inputs from many different sources. One such source are simulations of the transport system. With such a simulation system, one can insert a proposed infrastructure or policy measure into the model, and observe the simulated reactions of the transport system. Downstream modules, such as the calculation of emissions (Kickhöfer et al., 2013) or accessibilities (Nicolai and Nagel, 2014) can be attached. An important issue with such models is that it is rather time-consuming and expensive to put them together; for example, the model for the German national assessment exercise takes several years to put together, which may be prohibitive especially for a developing country.

In this situation, it is interesting to consider alternative, and possibly faster and cheaper, approaches. With the availability of OpenStreetMap (OSM) data, one major obstacle has been removed. We have consistently found that it is possible to base traffic simulation models based on that data in spite of some shortcomings (see Chapter 1), and while the data quality of OSM differs heavily between urban and rural areas, it approaches that of commercial network data for large cities (Zielstra and Zipf, 2010). The main issue is that OpenStreetMap data does not contain flow capacities, i.e. the maximum number of vehicles that can leave a link during an hour. Instead, that number is estimated from road category information.

The other missing item in order to bring such a model up and running is the demand model. The demand model contains, in some way, information about all trips that are made from one location to another during one day. Such data is typically obtained from surveys. Two typical sources are the census or similar information (which typically contains, besides the home location, the work or education location), or trip diaries, which are in many countries obtained from asking a sample of the population about how, and at which locations, they spent a certain day. This can even be obtained from electronic sources, see Sec. 3.1.

However, sometimes such information is not available, or it is difficult to procure, for example because of privacy issues. In this situation, the generation of synthetic demand from passive electronic sources has become an increasingly active research field. Some of these investigations have a focus on route choice (e.g. Rieser-Schüssler et al., 2012), while others derive origin/destination matrices (Calabrese et al., 2011; Ma et al., 2013). Most of them employ some sort of intermediate model of behavior or trip generation before assigning traffic to the road. The guiding focus for the present chapter is the question in how far a meaningful traffic simulation can be constructed directly just out of OpenStreetMap network data, and anonymous cell phone traces as provided by the Data for Development (D4D) challenge (Blondel et al., 2012).

The chapter is structured as follows: In section 2, we describe the data sets used for creating the supply (network) and demand (population) data. Section 3 covers the construction of the simulation model. In section 4, we give some results of parametric simulation runs. In section 5, we discuss some of the issues we faced while constructing the model and directions for future research.

4.2. Data description

4.2.1. Road network

The road network data for this scenario is based on OpenStreetMap. The OpenStreetMap data was converted to a simulation network by assigning attributes related to traffic flow to road segments. This is done based on the value of the `highway` tag, a road classification scheme particular to OpenStreetMap (see Chapter 1). A graph representation of the road network is constructed from the OpenStreetMap data, where each intersection becomes a vertex and each road segment becomes a link. The precise geographic embedding of the road segments is discarded, and only their length is stored as a link attribute, along with their capacity, maximum speed in uncongested state, and number of lanes. Lacking a better source, we used the values from Figure 1.1. The resulting network for the entire country has approximately 22,000 nodes and 63,000 links.

According to the CIA world factbook web site[1], the length of the road network of Côte d'Ivoire is 80,000 km, of which 6,500 km are paved. The OpenStreetMap documentation features an overview of the international equivalence of highway tags, but no country in Africa is currently included in this overview. The combined length of all edges labelled with any value of the highway tag is 29,300 km. The combined length of all edges labelled with `highway=primary`, `highway=motorway` or `highway=trunk` is 9,000 km, suggesting that these categories together already contain some roads classified as unpaved by the other source, and that the rest of the network should definitely be considered unpaved. Still, the data only accounts for less than half of the reported network length, so it is to be expected that siginificant parts of the network, most of it probably in rural areas, is not available in the model. This contrasts with coverage in the more economically

[1] `https://www.cia.gov/library/publications/the-world-factbook/geos/iv.html`

developed countries, where the part of the road network not covered by OpenStreetMap can be considered negligible for traffic modelling purposes.

Upon inspection, certain areas of the country seem to have a good coverage, notably the area of the economic capital, Abidjan. For this reason, and because our simulation approach has so far been applied mostly for urban areas, we decided at this point to focus on the city.

4.2.2. Mobile phone sightings

The mobile phone data under consideration here consist of two sets of individual trajectories collected over a study period of 150 days: One set of high spatial resolution (HSR) data, and one set of long term (LT) data. The HSR set consists of trajectories generated from billing data, and tracks 50,000 individuals. The individuals are drawn from the customer base of one mobile phone operator, Orange, which claims five million customers, at an estimated total population of twenty million.

Locations are given by the number of the cell phone tower with which the mobile phone was communicating at the time of the record. Locations are only recorded while the user is in a call. After every two-week period, the population sample is redrawn, so it is not possible to track a single individual over more than two weeks. The LT set tracks another sample of 50,000 individuals, but over the whole study period, at the price of providing much lower spatial resolution, namely at the level of sub-prefectures. In this experiment, the idea was to directly generate a pattern of daily traffic from trajectories, so we use the HSR data set.

4.3. Traffic simulation model

The simulation is a loop which consists of:

- traffic flow simulation
- scoring
- replanning

This loop operates on an initial population of individiual agents which is generated from the mobile phone data. The following sections describe the phases in detail.

4.3.1. Initial Demand

Côte d'Ivoire has an estimated population of 20 million, according to the World Bank population data set on `google.com/publicdata`. We generate a synthetic one percent sample population by creating 200,000 synthetic agents. We simulate no particular day, but a typical work day, so we overlay four arbitrary days of mobility traces, each taken from a different sample (size 50,000) of mobile phone customers from the HSR data set. This means that we are combining data from multiple days to build a population for a

supposedly average working day. Clearly, this is not the same as having 200,000 samples from one day. It is, however, still better than the alternative, which is expanding the first 50,000 samples to 200,000.

Each agent is equipped with a mobility plan which consists of an alternating list of

- geo-located and timed activities
- leg descriptions, including mode of transport and route

Agents essentially divide their time between conducting an activity, and travelling.

For each agent, an initial mobility plan is devised which is consistent with the data: The agent has to be in cell C_i at time t_i for every reading i. This leaves many degrees of freedom. In particular, it is not known whether the agent is travelling when a reading is taken. Any partitioning of the time into activities and trips which is consistent with the readings is in principle admissible. We start by defining every reading i to be an activity which ends at the time t_i the reading is taken. If several consecutive readings happen at the same cell, only the latest of those is considered. At this time, the agent will start travelling towards the location of the next reading $i+1$ and, upon arrival, will stay there until the time t_{i+1}. During this time, the agent is considered to be conducting an activity. Activity locations are fixed to a geographical point which is randomly drawn from the Voronoi cell C_i of the tower where the reading was taken. For a short discussion of spurious handovers, see Sec. 4.4. It is assumed that activity locations have direct access to the road network, so they can be positionally identified with links. Each randomly drawn point is therefore snapped to the end of the nearest link, which is considered to be the activity location.

The plan is then checked for initial feasibility. Each leg is routed through the road network on the fastest route at maximum uncongested vehicle speed, as per the link attributes stored in the road network model. The travel time is summed up, and it is checked if the agent would under these assumptions be able to reach the next activity location in time. If this is not the case, the plan is redrawn, which means that different activity locations will be drawn within each cell tower's Voronoi cell. This procedure is iterated 20 times, and if no feasible plan has been found by then, the case is considered pathological and discarded. This does not necessarily mean that the input data does not resemble a real trajectory. Incomplete road network data is the most likely reason for these cases.

Finally, we filter and keep only those plans with at least one sighting in the Abidjan urban area, which is our study area.

The result of this initial demand generation process is a 1% synthetic population sample where every agent uses a car for every trip and tries to take the freespeed-fastest route through the road network; for an example see Figure 4.2. This initial population is then fed into the simulation loop for relaxation, the steps of which are described in the next paragraphs.

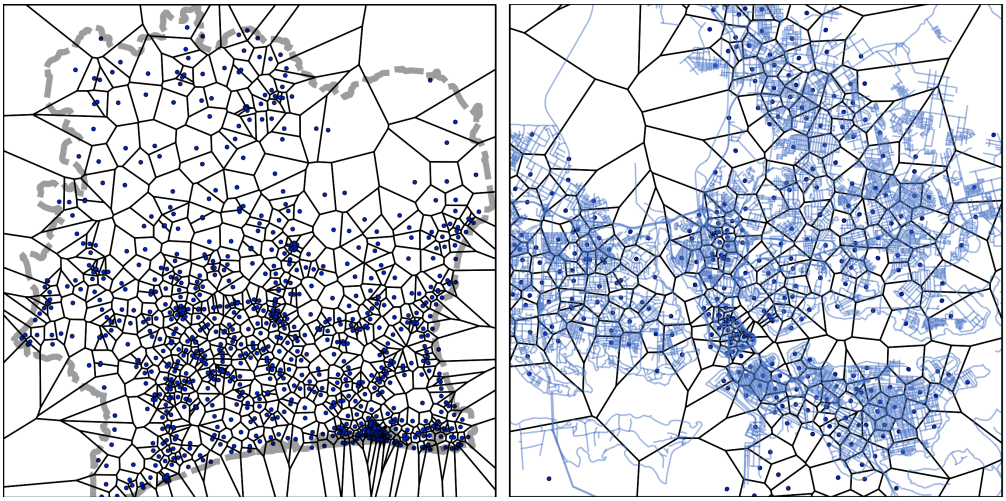


Figure 4.1.: Cell tower locations with their Voronoi cells. Comparing Voronoi cell sizes on a country-wide scale (left, overlayed with national border) and on the scale of the Abidjan urban area (right, overlayed with road network) show the large variation in cell size.

4.3.2. Traffic flow

The mobility simulation concurrently executes the mobility plans of the agents. Agents leave their activity location at the scheduled activity end time and head for their next destination. The road traffic is simulated using the queuing model of traffic flow (Gawron, 1998). In this model, the limited flow capacity of links is honored, so that in every time step, only as many vehicles can exit a link as specified. If more vehicles enter a link than its flow capacity admits, vehicles will accumulate on the link until its storage capacity is hit, which is determined by the length and width of the link. When a link is full, no more vehicles can enter, causing the congestion to propagate back upstream. The simulation records time use for each agent: The points in time where agents depart from and arrive at their activity locations are fed back to the agents as experience to evaluate the utility of the executed plan. Note that simulated vehicles are considered uniform. In this experiment, in contrast to explicitly modelling different vehicle types and their interaction (Agarwal et al., 2015), we model mixed traffic as a combination of uniform car traffic and the possibility to switch to an uncongested mode, which is described below.

4.3.3. Plan scoring

The agents evaluate the outcome of their plan with a simple utility-based approach. Time spent travelling is considered to contribute negative utility. Since, in this study, we do not have an activity model which would allow comparing the relative utility of time spent at one location with another, we consider time spent in activities to have no contribution to the utility function. Since we have no prior knowledge about the trip

```
<person id="10008_1">
 <plan>
  <act type="sighting"
   link="91273255_1059606493_1059606739_R"
   x="-455429.63088982203" y="593487.4664630938"
   end_time="07:42:00" />
  <leg mode="car"
   dep_time="07:42:00"
   trav_time="00:04:15"
   arr_time="07:46:15">
    <route type="links">
     91273255_1059606493_1059606739_R
     91273255_1059606493_1059606739
     91273253_1219665629_1059606739_R
     [...]
     125239948_338881494_338881537</route>
   </leg>
   <act type="sighting"
    link="125239948_338881494_338881537"
    x="-454496.5669239445" y="593286.9239709259"
    end_time="17:36:00" />
   <leg mode="car"
    dep_time="17:36:00"
    trav_time="00:00:31"
    arr_time="17:36:31">
     <route type="links">
      [...]
     </route>
    </leg>
    <act type="sighting"
     link="30630786_338406146_338406157"
     x="-454806.9268885712" y="593254.6327337974"
     end_time="18:35:00" />
 </plan>
</person>
```

Figure 4.2.: Example for an agent plan. This person had 3 call records on the examined day. From the location of the first call record ("sighting", location randomized within the cell) to the second, the free speed travel time is only about 4 minutes, from which the actually experienced, congested travel time may differ enormously. Upon arrival at the second location, the agent will wait until 17:36:00 and depart for its final destination.

strucure, and our modelling decision to split trips at the locations of the GSM readings is somewhat arbitrary in this respect, we consider the disutility of travelling to be linear in total travel time.

$$U = \beta_{trav} \cdot t_{trav} \tag{4.1}$$

Since in this experiment the utility function does not have any other terms, the value for β_{trav} is arbitrary, as long as it is negative.

4.3.4. Replanning

After each iteration, the agent population has the opportunity to change their mobility plans in reaction to the outcome of the mobility simulation. In this study, agents have three replanning options. Two of them are creative. Agents chosing these options produce a new plan and execute it in the next iteration of the mobility simulation. These options are route choice and mode choice. The third is switching plans, in which agents retry a previously executed plan from their plan memory based on its previously experienced utility. In each iteration, 10% of the agent population consider their route choice and mode choice, respectively. The remaining 80% can switch plans.

Route choice Agents reconsider their route through the road network. Instead of taking the least-cost path based on free speed travel times, the link travel times computed as an outcome of the last iteration of the traffic flow simulation are used. In the first iteration, the traffic will concentrate on main roads, leading to high traffic volumes and high traffic times. Agent reconsidering their route will divert to smaller roads in the next iteration.

Mode choice In the initial population, all trips are done by car. Our model summarizes all alternatives to driving a car in a second mode. Agents which choose this mode are not routed through the network at all. They experience a travel time calculated from the free-speed car travel time between the origin and destination locations, times a *travel time factor* which characterizes the mode. These agents do not interact with other agents while travelling. They are not impeded by other travellers and do not contribute to congestion themselves (Rieser et al., 2009). Depending on the travel time factor, some members of the population remove themselves from the road network. Note that the modal split is not part of the input data, but an output of the simulation, dependent in particular on the travel time factor.

Switching plans Every agent has a fixed-size plan memory, set to size 5 in this experiment. Agents which are assigned the option to switch plans pick one of their previously tried plans uniformly at random, and switch to that plan with a probability depending on the difference between the most recently experienced scores of both plans:

$$p_{ij} = 0.01e^{\frac{s_j - s_i}{2}} \tag{4.2}$$

In this equation, p_{ij} is the probability of switching frm plan i to j, s_i is the current score of plan i, and 0.01 is the probability of switching between equally scored plans. The simulation is iterated until the system reaches a relaxed state. We consider this to be the case as soon as the average agent score (i.e. travel times) and the mode share have stabilized.

4.4. Implementation and Results

The scenario was implemented using the MATSim agent-oriented transport simulation software (`http://matsim.org`). In order to be able to run experiments on a desktop computer, we decided to simulate the 1% sample of the synthetic population as described in the previous section, scaling the network capacity accordingly. In our experience with the process, this is sufficient to pick up large-scale characteristics of the system. One simulation run takes about an hour on a 2.2 GHz Intel Core i7 MacBook. A run consists of 180 iterations of the simulation loop, which we found to be enough for the quantities presented in this chapter to stop drifting. For the last 30 iterations, the creative replanning options, namely route choice and mode choice, are disabled, and agents only switch between existing plans. This is done to eliminate the bias introduced by having a large fraction of the agent population take new routes or the alternative mode without regard for their possibly low utility.

We produced several parametric simulation runs, varying the travel time factor for the non-car alternative. As can be seen in Figure 4.3, a travel time factor of 4 already leaves some agents unable to reach the point of their last sighting before midnight, which means that this state of affairs would be clearly inconsistent with the data. Factor 2 still seems admissible. The travel time factor can be interpreted as how long car travel times along a path in the congested network need to become compared to its free-flow state so that agents travelling along that path will be moved towards using the alternative mode.

In figure 4.4, we plot the resulting share of car drivers over the travel time factor. Since a factor of 4 is already considered inadmissible, the prediction would be a share of not much more than 0.2.

With a factor of 4, i.e. a situation which is quite congested according to Fig. 4.3, no systematic or directed traffic jam patterns emerge. The network just seems too full overall (see Figure 4.5). This is quite different from other similar studies such as by Nagel (2008, 2011), which always found quite well-structured congestion patterns, in particular into the city during the morning peak. Further inspection of the results leads to the observation that most of the congestion in our model seems to be away from the freeways, on the secondary road network. That is, under congested conditions traffic is unable to get out of the secondary street network. Once the traffic makes it onto the primary network, the model displays few if any restrictions. Clearly, this statement would need to be verified on the ground before being a possible basis for planning decisions. It could, for example, also be a consequence of the demand generation, which, in particular because of spurious cell handovers, may generate a lot more local traffic than there is in reality. If such verification on the ground would corrobate that the local congestion

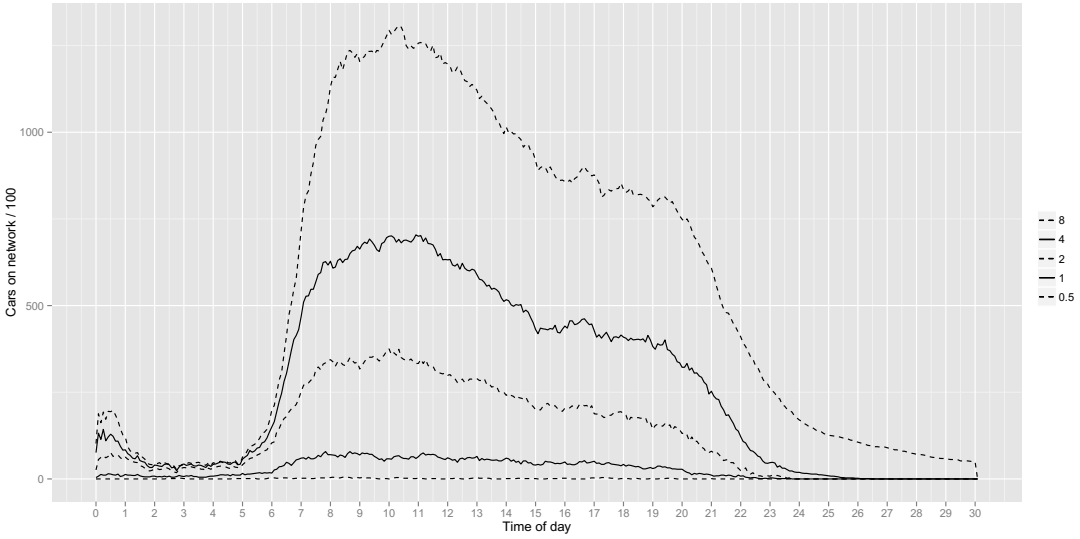


Figure 4.3.: Number of cars en-route over time of day, plotted for different values of the alternative mode travel time factor. One agent represents 100 travellers. Note that as only sightings from a single day are used to construct the artificial population, agents are expected to be at their final location by midnight at the latest. For factor 4, the network is already too full to permit this. If the alternative mode is faster than driving (factor 0.5), the network is, expectedly, cleared.

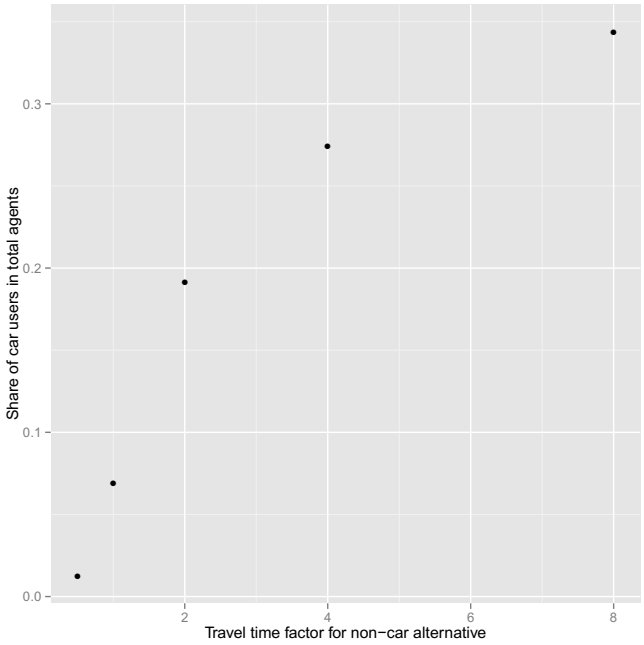


Figure 4.4.: Number of agents using a car, plotted over the alternative mode travel time factor. Note that an agent either does or does not use a car for all trips of the simulated day. We do not consider mode choice for individual trips.

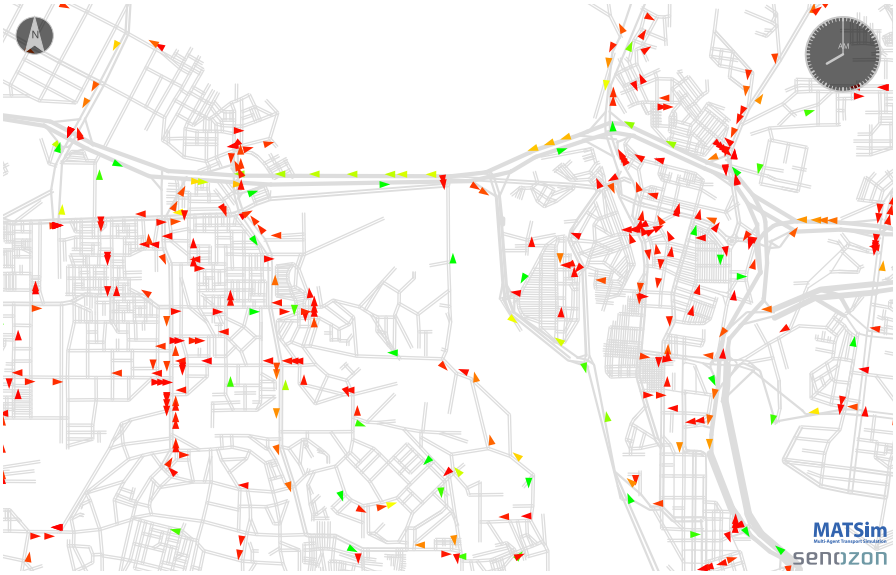


Figure 4.5.: Screen shot of simulated traffic.

effects are over-estimated, then methods to remove those spurious cell handovers from the demand generation would need to be inserted into the model.

Figure 4.6 displays the probability density of the total travel time per person per day. One notices a peak near 0.2 hours for the car mode, and near 0.6 hours for the non-car mode. While the 0.6 hour value seems plausible for a reasonable walk length of 30 minutes and an average number of daily trips of less than 2 (Pendakur, 2005), the 0.2 value seems way too low, suggesting again that we overestimate the number of local (very short) car trips.

4.5. Discussion and outlook

Modelling road network access In the present scenario, as well as in the MATSim software package, the assumption is that every activity location has direct access to the modelled road network. For rural areas in developing countries, this is clearly not met, if only because the OSM-based network model accounts for less than half of the presumed length of the actual road network. In this chapter, we focus on the Abidjan urban area, but for a country-wide study, it might be worthwhile to improve on this aspect.

We therefore intend, in future version of MATSim, to explicitly account for the distance between the random location and the network by modelling trips in several stages: network access, network travel and network egress. The travel time for the network stage are determined by the traffic flow simulation, as in the present chapter. Travel times for network access and network egress are determined by multiplying Euclidic distance by some factor which represents an unknown mode of travel through unknown terrain with an unknown detour. Conceptually, this would be done by extending the road network

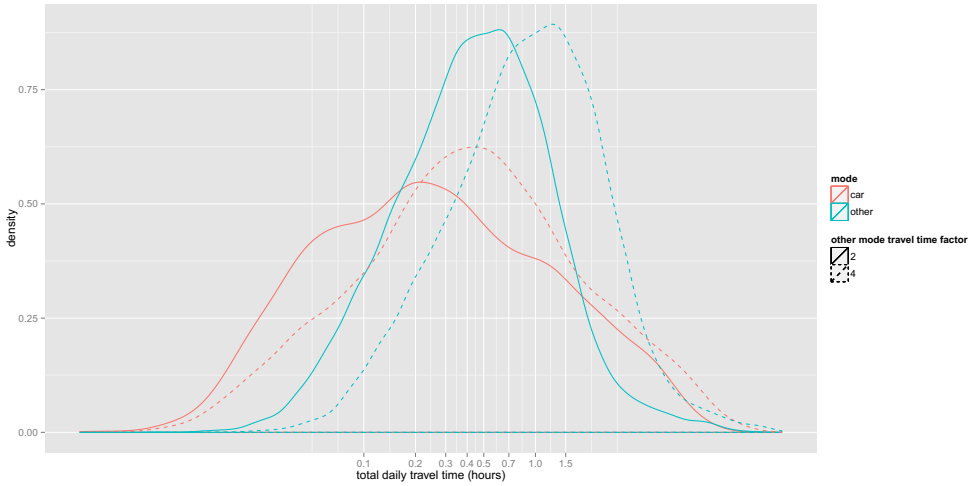


Figure 4.6.: The daily time spent traveling, for the driving and non-driving subpopulations.

with virtual access links which orthogonally connect activity locations to the nearest road network link. This would model a mode of travel composed of several stages, like walking to the next road, being picked up by a motorist, and continuing by car.[2]

Imputing behavioral meaning Most if not all similar studies go the path that they first attempt to create plausible daily activity plans from the mobile phone records and only then move on to an assignment of the traffic onto the traffic infrastructure. For the present investigation, we have deliberately chosen to immediately assign the mobile phone data to the road network, without an intermediate interpretational layer. There were two reasons to do so:

- We believe that plausible traffic patterns can already be obtained without that intermediate step, and that it saves a lot of time in order to get such simulations up and running. This could, for example, be important for situations with limited budget, or with situations with time pressure such as, say, disaster relief. Clearly, the claim that the results are realistic would need to be checked, for example by traffic counts data on the ground. Such data is, however, fairly straightforward to obtain, for example by, on a particular day, employing someone to stand next to the roadside and count vehicles.

- We believe that it is possible to impute the activity chains also *after* the traffic assignment. In fact, we believe that it may be better to do so, since the interpretational layer always means a loss of information that may still have been in the raw data, such as the deletion of seemingly implausible sightings, or certain variations in the temporal structure from one day to the next. With the approach discussed

[2]This feature has been implemented in MATSim in the meantime.

in this chapter, one could always keep the original plan based on the mobile phone data, but generate multiple alternative plans for every synthetic traveler that would be consistent with the mobile phone data. For example, it could be assumed that some phone calls would actually be done en-route, or that some activities would carry on after the last phone call at a certain location. Out of these multiple interpretations of the mobile data, the system could converge to a set of interpretations that is most consistent with other data, such as, for example, time-dependent traffic flow data. This will be the subject of future work (see Chapter 7).

- An alternative approach might be to use a time use survey, which is available in many countries, as additional data input. Time use surveys are similar to trip diaries in that they follow persons over days, but in contrast to trip diaries they typically do not register locations. Advantages of time use surveys over trip diaries include that they are considerably cheaper to obtain since the geocoding of the locations is expensive, at least with traditional approaches, and they have fewer privacy issues. In consequence, time use surveys are available in many places where trip diaries are not available. In addition, it may even be possible to use a time use survey from a neighboring city or country if the cultures are sufficiently similar. The imputation of activity chains from those time use surveys could be done in ways similar to those pointed out above: For each given sequence of cell phone sightings, one would select all possibly matching activity chains, or a randomly drawn subset. A data assimilation algorithm would then pick those activity chains most consistent with directly and anonymously measured data, such as traffic counts.

Statistical bias Trajectories sampled from mobile phone users alone are most probably biased. For example, not all members of the population have a phone, persons with a phone have vastly different calling patterns, and trips of persons who make fewer calls will be underreported. There is some indication that, for the purpose of mobility studies, such bias may not be as dramatic as it seems (Wesolowski et al., 2013). We believe that the approach discussed above, which is to do the data assimilation with the traffic model already up and running, might also help here. More specifically, one could imagine to give different statistical weights to every synthetic person. Based on other data, like for example time-dependent traffic flow data, one could re-weight the synthetic persons in order to bring the simulation closer to the data. This would presumably increase the weights of those types of persons that were under-weighted in the data, and decrease the weights of those types of persons that were over-weighted. Clearly, the approach will not work if certain types of persons are not included at all. We will investigate these issues in future work (see Section 6).

4.6. Conclusion

- The investigation demonstrates once more that it is possible to use publicly available OpenStreetMap data as the basis for traffic simulations. In the present situa-

tion, the coverage of the rural areas was still insufficient. However, one can either assume that this will improve over the years, or one could dispatch special investigations to insert the missing information if that turns out to be necessary for a specific study.

- The investigation also demonstrates that it is possible to obtain traffic patterns from mobile phone sightings without any layer of interpretation whatsoever. Some verification would be necessary to decide if they are close enough to reality in order to use the model for policy analysis.

- A parametric study demonstrates that the model is sensitive to the performance of non-car modes.

- Sec. 4.5 discusses a method how the model could be systematically improved further if additional data is available. As explained, such possible data could consist of, e.g., time use surveys or traffic counts.

5. Studying the accuracy of demand generation from mobile phone trajectories with synthetic data

A previous version of this chapter was published by Zilske and Nagel, 2014.

5.1. Introduction

Transport simulation scenarios usually make use of pen-and-paper trip diaries for their demand model. These are expensive to obtain. There is a substantial recent interest in using Call Detail Records (CDRs) as a data source for such simulations.

CDRs carry other information than travel diaries. They are mostly available for a longer timeframe, while travel diaries are typically obtained for a single study day. Conversely, travel diaries contain detailed data about the activities and trips conducted on the study day, while CDRs only witness the presence of the participant at a certain point in time in a certain mobile phone cell. Whether the person was travelling at that point in time, or conducting an activity, cannot be directly determined. Neither the mode of transport nor the activity type is available.

Privacy concerns inhibit the widespread use of such data. For research in an earlier stage where predictions in real-world scenarios are not yet attempted, it may be necessary and sufficient to work with abstractions from actual CDRs which are realistic in the sense that they reproduce statio-temporal properties of the behavior of people (Isaacman et al., 2012), but are unencumbered by the responsibility for the privacy of study participants.

In this work, we consider an agent-oriented transport simulation scenario and study the question of how much different the simulation outcome would be, if CDRs had been the only available input data for constructing the demand model.

5.2. Related work

A related subject is the construction of completely synthetic traces from a model parameterized from other sources. Isaacman et al. (ibid.) construct a CDR dataset drawing from several data sources, including a real CDR dataset. They assume that

1. A distribution of home locations
2. A distribution of work locations
3. A distribution of commute length given home location

4. A distribution of population density indexed by time

5. A distribution of calling habits

are available and note that these can either be available separately or extrapolated from the CDR dataset. Synthetic individuals with home and work locations are generated, a temporal calling pattern is drawn for them, and the calls are realized either at the home or at the work location. Adding additional locations is only touched briefly. Calls made while travelling are not considered. Furthermore, it appears that when a call is made, the probability of the user being at the home or at the work location is drawn independently for each call, which means that the number of location changes for each user does not appear to be considered important for the model. It is not discussed how to get the commute length distribution from a CDR dataset. The point of differential privacy is brought up, defined as the property that the output of the algorithm does not change much when a single individual is added to the input. This would, of course, be an argument for not putting dense traces or indeed any of the traces on the network directly, but to only work with distributions where no single trace or part of a trace is reproduced directly.

In a more recent work, Chen et al. (2014) construct a synthetic CDR dataset from

1. A travel diary survey.

2. A real CDR dataset.

For each travel diary, a CDR trace is drawn and used as the set of calling times for that individual. The location at each call time is taken from a simulation of the travel diary. This means that, like in the approach taken in this work, the probability of placing a call is independent of what the individual is doing at that point in time, in particular, travelling or being in an activity. The simulated locations are furthermore overlaid with parameterized noise, whose parameters are derived from the real CDR dataset, using some assumptions about the typical number of real activity locations. The noise function is different for calls at activity locations and calls while travelling. The previous and next sighting pairs at activity locations are used to demarkate the trip start and end time.

From a synthetic trace, activity locations and activity location types are determined by

1. Clustering sightings per device. It is not detailed whether the temporal structure goes into the clustering or whether it is purely spatial. Clusters are then classified into travel and activity clusters, through regression by their shape and number of sightings in them. Details about the model-based clustering which is used are not given.

2. Labelling the detected activity locations as home, work or other based on simple heuristics. Details are not given.

The step from activity locations per device back to daily travel plans is not taken. In particular, determining start and end times of activities are not discussed, and as in

Isaacman et al. (2012), the number of location changes, and thus the amount of travel, is not evaluated.

5.3. Synthetic CDRs

In order to have full control over the ground truth, for the present study the CDR data is synthetically generated from a simulated scenario. We start with a full implementation of an agent-oriented traffic model. The output of this model is a set of complete descriptions of mobility behavior of an agent population with labeled activities and space-time trajectories on the level of network links. We consider this the ground truth of a hypothetical scenario. Note that additional kinds of measurements can be taken from this output, in particular volumes on links at arbitrary time scales. We can now obtain synthetic CDRs from such a scenario. A MATSim plug-in was developed to accomplish this. The plug-in takes two additional inputs:

- A cell coverage, which partitions the simulated geographic area into mobile phone cells.
- A mobile phone usage model. The software exploits the benefits of an agent-oriented simulation framework, allowing for different population segments with different calling habits, but also allowing for experimental simplifications such as placing a call precisely on arrival at or departure from activity locations.

In every timestep, every agent gets to decide whether or not to make a phone call. When a phone call is made, the framework locates the agent within the cell coverage, and records a CDR. The first output of this step is a set of CDRs

$$(p_i, t_i, c_i) \tag{5.1}$$

where p_i is a person identifier, t_i a timestamp, and c_i a cell. The second output is a set of link traffic counts y_{ij}, the number of vehicles which have passed link i in time window j.

We consider this the available data for traffic modeling in the hypothetical scenario. This framework allows us to study methods for constructing demand models from CDRs, and how much information from CDRs, traffic counts, and possibly other input is needed to re-approximate the state of the traffic system in the ground truth scenario to which degree. It isolates these questions from the different question of how good the simulation model itself is at approximating reality.

5.4. Simulation driven by CDRs

We convert each CDR trajectory into a travel diary in a straightforward way. For every observed caller, a MATsim person agent is created. Every call is converted into an activity. Activities are connected by trips. Several calls in the same zone without a call

in a different zone between them are fused, since there is no evidence of travel between them. Similarly, there is no evidence for detours, so no additional activities are inserted. The only degree of freedom is the departure time from each activity location, which must be no earlier than the time of the last sighting at the activity location, and no later than the time of the first sighting at the next activity location minus the required travel time.

The simplest solution is to set the activity end time to the time of the last sighting at the activity location: the phone call is assumed to have taken place at the time the agent leaves the activity.

The resulting plans are simulated. The output of this step is of the same form as the ground truth scenario. The two scenarios can now be compared to assess the approximation quality.

5.5. Results

5.5.1. Test scenario

To test our software and illustrate its workings for a corner case, we use the following scenario:

- As a ground truth scenario, some agents take trips between some random activity locations on an uncongested network. In MATSim, this means that everybody uses fastest routes with respect to free speed travel time.
- Each link is its own mobile phone cell. Since MATSim works on the level of links, this means that CDRs are taken at the highest spatial resolution available.
- Agents make calls exactly when they start and end activities.

As expected, in this setup, traffic is reproduced exactly. The same happens when even more phone calls during activities or during travel are inserted, since they do not add additional information to this scenario. Also, they do not add artifacts to the simulation. This result is independent of the network or the demand.

5.5.2. Uncongested scenario

As a more realistic scenario, a travel demand model generated from real data is used. It is created from a 1998 household survey which contains complete trip diaries from one specific day of 2% of the Berlin population. The survey is not publicly available, but has been used before (Moyo Oliveros and Nagel, 2016; Scheiner, 2005). It contains activity locations, activity types, activity start and end times, and modes of transport for each trip. It does not contain any route information. We select all individuals who only travel by car and obtain 18 377 individuals. The network contains 61 920 links, of which a random 5% are chosen to collect volume counts in hourly time windows. We disregard the spatial uncertainty of sightings and identify one mobile phone cell with each link. We also disregard capacity constraints in the traffic network, i.e. for this study there is

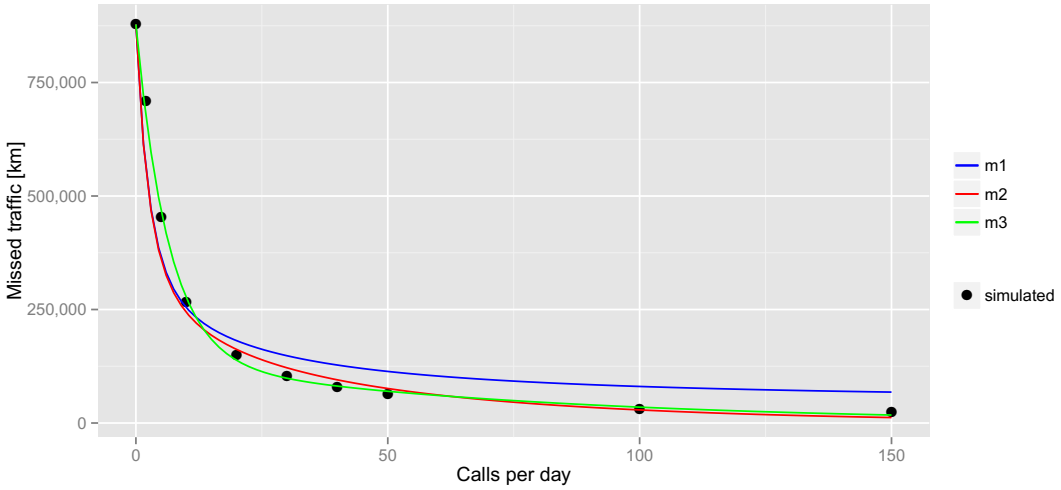


Figure 5.1.: Missed amount of traffic at different call rates. Simulation runs and models.

no traffic congestion. Every agent chooses fastest routes with respect to free-speed travel time. We obtain a total travelled distance of about 878 000 km.

Agents place calls with uniform probability throughout the day, at a specified daily rate λ. Every agent has the same call rate. Multiple runs from the same ground truth scenario are run, with varying call rates. While average call rates of 50 calls per day or higher are certainly not realistic, these cases are still important to consider, because in practice, CDRs or similar data points need not be caused by actual phone calls, but might also appear as a consequence of, for instance, internet usage. We consider the terms CDR, call rate, and cell, to be interchangeable with corresponding concepts in other current or future technologies which produce trajectories.

We define the missed total travelled distance in the network compared to the base case as the error measure. As expected, the error drops with increasing call rate, on account of fewer trips to and from activities missed by the sampling (Fig. 5.1). Note that even with a high average call rate of 50 calls per day, the approach still misses about 10% of traffic. Thus, in order to make this model practically useful, some compensation method needs to be devised. As a first step in this direction, we investigate analytical models which could explain the data.

Assuming Poisson distributed calls with rate λ, the probability of missing an activity of duration t is $P_{\lambda t}(0) = e^{-\lambda t}$. Summing this over the empirical activity time sample and multiplying with the average trip length gives the expected missed traffic under the simplifying assumption that every activity accounts for the same amount of traffic.

$$m_1(\lambda; (t_1, \ldots, t_n)) = L \frac{1}{n} \sum_{i=1}^{n} e^{-\lambda t_i} \tag{5.2}$$

This model systematically predicts larger values of missed kilometers than the simulation at higher call rates. Following up on the intuition that this may in part be due to a

correlation between the duration of an activity and the additional travel produced by it, we tried another analytical model, which incorporates trip distances, by considering the power sets of activity chains. For instance, an activity chain home-work-shopping-home has 16 ways of being sampled, among them home-shopping-home and home-work-home, but also shopping-home, since we do not consider home locations in a special way. We computed shortest-path travel distances for all partial plans of all agents, along with their probabilities depending on the call rate, and calculated the expected missed kilometers. This model fails to provide a better fit to the simulation runs. Its graph looks very similar to the much simpler model m_1, so it is omitted from the figure.

The only other way in which the simulation differs from the assumptions in m_1 is that in the simulation, agents also place calls while travelling. We tried to incorporate this effect into m_1 by capping the empirical activity durations at a given minimum duration t_{min}. The intuition behind this is that some time before and after the actual activity beginning and end, the agent is already or still near the activity location.

$$m_2(\lambda; (t_1, \ldots, t_n), t_{min}) = L\frac{1}{n}\sum_{i=1}^{n} e^{-\lambda \cdot \max(t_i, t_{min})} \tag{5.3}$$

This model has a best fit to the simulation runs at a parameter value of $t_{min} = 23min$, where it is now quite close to the simulation result with some remaining errors at intermediate call rates (Fig. 5.1).

Following up on this, we now disregard the empirical activity duration sample and consider two typical activity durations, long (t_1) and short (t_2), with a relative frequency β of short activities, which yields a bi-exponential model.

$$m_3(\lambda; t_1, t_2, \beta) = L \cdot \left((1-\beta)e^{-\lambda t_1} + \beta e^{-\lambda t_2}\right) \tag{5.4}$$

At values of $t_1 = 3.7h$, $t_2 = 19.8min$ and $\beta = 0.16$, obtained by the nls solver of the R software package (R Core Team, 2013), this gives a good visual fit to the simulation runs. While these values do not approximate the empirical mean activity duration of $6.5h$, the model seems to pick up the effects of the empirical activity durations, the trip structures, and of calls placed while travelling.

5.5.3. Congested scenario

In a more realistic scenario, where roads are capacity-constrained, route choice in MATSim resembles a dynamic traffic assignment procedure. We generate a new ground-truth scenario based on the same set of travel diaries but realistic link capacities, in a state resembling a dynamic user equilibrium with respect to travel times. Compared to the uncongested scenario, the total travelled distance increases by about 5% due to detours resulting from time-distance trade-offs.

Applying the same process as above means that dynamic traffic assignment is now performed between sightings. Carrying out the same parametric runs as for the uncongested case reveals consistently less missed traffic in relative as well as absolute terms compared to the uncongested case (Table 5.1).

Table 5.1.: Reproduction of total travel distance

uncongested scenario		
rate	total travel distance [km]	relative to base
base	877934	1.0000
2	168672	0.1921
5	424627	0.4837
10	612295	0.6974
20	729089	0.8305
50	814241	0.9275
100	846466	0.9642
150	853450	0.9721
activity start/end	877934	1.0000
congested scenario		
rate	total travel distance [km]	relative to base
base	920928	1.0000
2	180407	0.1959
5	467898	0.5081
10	668082	0.7254
20	798749	0.8673
50	878649	0.9541
100	898857	0.9760
150	902353	0.9798
activity start/end	918223	0.9971

This can be explained as follows: In the uncongested scenario, the only effect of raising the call rate is that more activities are detected, so that more trips are taken. In the congested scenario, it has the additional effect that trajectories are traced more accurately, reproducing a larger share of the 5% travel distance increase which is due to detours.

In the congested scenario, all routes between sightings are fastest routes with respect to the congested traffic conditions. In every reconstructed case, however, there is overall less traffic, and in that situation, average fastest routes between any given pair of points tend to be shorter. For this reason, a higher sampling rate, and hence a more accurate tracing of routes, leads to a relatively higher reproduction of total travel compared to the uncongested scenario.

To illustrate the different behavior of the uncongested and the congested simulation, we consider the difference of travelled kilometers between base case and reconstructed case on a per person basis (Fig. 5.2). Each panel shows the reconstructed travel distance of each person in one of the scenarios versus the real travel distance in the base case. The rows are for the congested and uncongested scenario. The columns are call rates per day. The rightmost column shows a run with the artificial behavior of placing calls

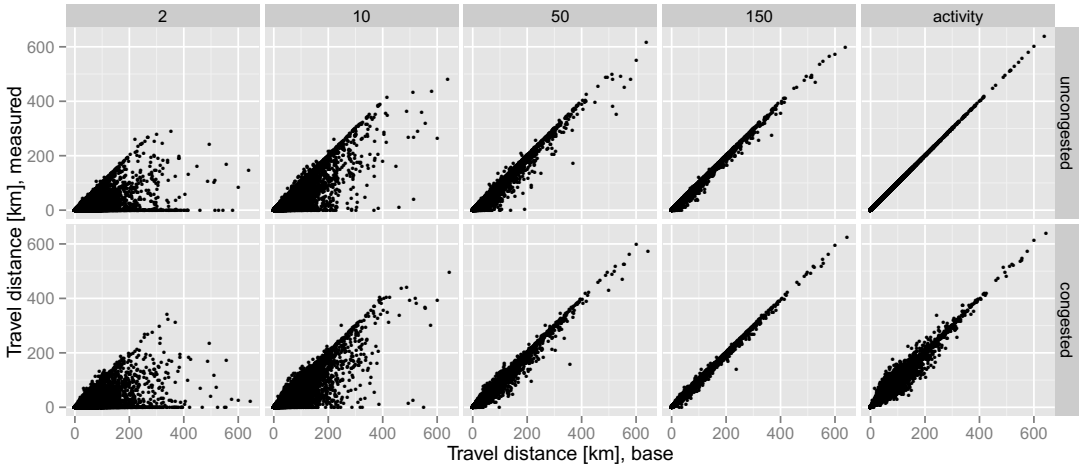


Figure 5.2.: Travelled distance per person, measured vs. base case, at different call rates

precisely at the beginnings and ends of activities. In the uncongested case, this reproduces travel distance per person exactly. More sightings would not add more information. In the congested case, we see a spread of missing or surplus travel distance per person. This reflects uncertainty about routes: The reconstructed scenario is in an equilibrium which is different from the base case. This means that under realistic traffic conditions, call rates which are higher than necessary to reproduce most activities still lead to an improvement in the reproduction of the traffic state, as individual routes are reproduced more faithfully.

The data points above the diagonal in the uncongested cases are a special case. They represent agents for which a missed activity results in a longer travel distance, because the activity is located on a tour of shorter length but longer duration. Missing the activity allows the agent to take a faster but longer tour through its remaining destinations.

5.6. Summary

The purpose of this chapter was to introduce the idea of simulating the acquisition of CDRs, which is a sampling process, as a framework to evaluate methods to create demand models from CDRs and to give a first estimate of the component of the sampling error which is due to the temporal resolution of the traces. Using a rich model of ground truth as a reference, it is possible to test hypotheses about the relationship between the unknown activity-trip-structure and the observed behavior. Increasing the sampling rate, even at already high rates, still has additional utility in reproducing traffic. This cannot be explained by more activities being hit, because many short activities are still not hit, even at such high rates, and those activities are not less likely than longer activities to produce additional traffic. Rather, it is explained by the fact that while it is unlikely that an additional call will fall in the timespan of a yet unsampled short activities, it

is more likely that it will fall in the timespan of a trip, giving evidence for part of the actual movement. Under congested conditions, calls during trips become even more relevant, because they give evidence for congestion-induced detours. This is information which, in travel diaries, will only show up in the form of stated travel times. In short, in the relationship between temporal sampling rate and measurement error, there are two regimes: Where decreasing error is due to less missed trips, and where it is due to tighter lower bounds on trip lengths.

The modelling method itself, as specified in section 5.4, is simplistic in that the resulting traffic will always be too little, because unsampled activities are simply missing. Adding additional trips to the plan of an individual, within the constraints of the potential path area induced by the CDRs, would be possible, but would require additional behavioral parameters. However, taking in some simple observable data, like the total travel distance or link volume counts, either the travel demand or the simulation output could be scaled to compensate. Our setup allows for the evaluation of such methods.

So far, we have only considered car traffic. We started out with agents using cars and recreated trajectories on the premise that they were from car users. However, there are multi-modal scenarios available which are open to similar methods: Synthetic CDRs can be generated from such a scenario without attaching the information whether the user is driving, using public transit, or walking, and these CDRs would then be used to reproduce a multi-modal scenario. This would need to incorporate a mode detection model, and this model would be validated by the degree to which the original modal split is reproduced. As far as public transit is concerned, it would be interesting to what degree line occupancy and line switch patterns could be reproduced.

We need to consider how to make use of the fact that real CDR datasets can span several days. This question arises both in synthesizing and in using CDRs. Our usual view of MATSim is that we simulate a typical work day. Synthetic CDRs spanning several days can be easily obtained by letting the same simulated day restart and continuing to apply the phone usage model. However, this would amount only to differently sampled concatenations of the same day, with zero temporal variability in the underlying travel behavior. Since MATSim is a stochastic model, it has variability between different runs with the same input data, but it is not clear if this translates well into temporal variability (Horni et al., 2011).

There are other modes of obtaining trajectories without user interaction. Services like Google Location History (formerly called Latitude) take location measurements directly from the handset and store them on a server, with explicit consent of the user, and without involving the phone operator. This acquisition mode has different characteristics from CDRs: The acquisition is not tied to the user using the phone, but runs in the background, at a rate which is determined by a trade-off between accuracy and energy usage. Our experimental setup can be easily adapted to synthesize such trajectories, with some of their special characteristics in mind (e.g. GPS does not work on underground trains.)

Phone network cell coverage data is publicly available, which can be used to improve the rather simplistic CDR simulator, so that it generates more realistic input data which

takes into account mobile phone cells of varying size.

6. Creating a scenario from mobile phone trajectories and traffic counts

A previous version of this chapter was published by Zilske and Nagel, 2015.

6.1. Iterated dynamic traffic assignment and calibration

In the MATSim transport microsimulation (Balmer et al., 2009), each traveler is modeled as an agent who chooses from a set of plans (Raney and Nagel, 2006). A plan is a whole-day travel and activity diary with complete route information. All agents are simulated on a road network, carrying out their plans. The result of the network simulation is fed back to the agent, which uses it to re-assign a choice probability to its executed plan, based on its perceived utility given the network conditions. This process is iterated until an equilibrium is reached, where all choice probabilities of the plans of all agents are such that the concurrent actions of all agents produce traffic conditions which lead to the same utility perception based upon which the plan was chosen (Nagel and Flötteröd, 2012). The plan distribution from which the agents draw is a required input to the simulation. In the most typical case, each agent represents one individual from a travel diary survey, and its plan is the stated behavior on the survey day. A choice set of alternative plans is dynamically generated over the iterations, each one by mutating a previous plan in one choice dimension, such as departure time or route. The initial plan, the free choice dimensions, and the parameters of the utility model which determines the choice probabilities together describe the choice distribution.

The Cadyts calibration system (Flötteröd, 2009) works by influencing plan choice probabilities considering known traffic counts. It directs the synthetic population towards choices more consistent with measurements. It calculates an offset to the systematic utility of each plan iteration by iteration. The offset is calculated based on how much the choice of the plan contributes to the whole traffic system fitting to the traffic counts. Plans which cross links where flow is underestimated are favored and vice versa. The agent population uses this adjusted utility for its choices, resulting in a posterior choice distribution which incorporates the information from the traffic counts.

In this chapter, we describe an application of this method to fuse traffic counts into an agent-oriented demand model constructed from CDR trajectories. While a travel diary contains start and end times of activities, a CDR trajectory only witnesses the presence of the individual at a certain instant in time in a certain mobile phone cell. Activities and trips during which no call is made are not witnessed. When a traffic demand model is constructed from a set of CDR trajectories, assuming it can be scaled with respect to

the overall sample size, the demand will likely be underestimated because of missing trips which are not covered by calls. Population segments with lower mobile phone usage are underrepresented. This is where the calibration scheme fits in. The set of travel plans feasible under the CDR trajectories serves as a prior demand model, which is calibrated to fit the traffic counts.

6.2. MATSim and Cadyts

6.2.1. MATSim

MATSim combines a traffic demand model based on individual daily travel plans with a microscopic traffic flow simulation to iteratively calculate a dynamic user equilibrium. Its demand model consists of a population of agents

$$A_1, \ldots, A_N \ . \tag{6.1}$$

Each agent has a mutable set of plans which can be understood as a choice set. The options are identical in the fixed dimensions (typically, the chain of activities with type and location), and vary in the open dimensions (typically, routes, modes of transport, and departure times). Every plan is assigned a mutable score, V_i, initialized to $+\infty$. Often, the score can be interpreted as utility.

Initial plans are auto-completed by the simulation as much as possible; for example, links are assigned to coordinates, and shortest path routes are computed if no routes are in the initial plans. Then, the following steps are iterated:

- Each agent chooses from its plan set according to a random utility model, where the choice distribution follows $P(i) = \exp(V_i)/\sum_j \exp(V_j)$.
- The chosen plans are loaded onto the network.
- For every chosen plan, V_i is re-calculated as a function of the plan's performance during the network loading (e.g. valuing travel time negatively) and assigned to that plan.
- Each agent in a random subset of the population adds a new plan to its plan set (identical to its other plans in the fixed choice dimensions, and distinct in the open dimensions) and removing an existing one if its plan set is now greater than a specified maximum.

The simulation is run until the variables on which the utility perception depends (e.g. dynamic link travel times) have converged to a steady state, and hence the choice distribution has become stationary. At that point, plan set mutation is ceased, so that the choice distribution now strictly follows the perceived utilities, and the simulation is continued until it converges a second time.

6.2.2. Cadyts

Cadyts is a calibration scheme which, when applied to MATSim and a vector of link traffic counts y, works by directing the plan choice probabilities of the whole agent population towards choices more consistent with the counts. This is achieved by calculating an offset to the score V_i of each chosen plan, iteration by iteration. Under certain additional assumptions, e.g. about the error distribution of the measurements, the adjusted choice distribution can be shown to approximate the posterior choice distribution given y (Flötteröd et al., 2011a; Flötteröd and Liu, 2014). It follows

$$P(i|y) = \frac{\exp\left(V_i + \sum_{ak\sim i} \frac{y_{ak}-q_{ak}}{\sigma^2_{ak}}\right)}{\sum_j \exp\left(V_j + \sum_{ak\sim j} \frac{y_{ak}-q_{ak}}{\sigma^2_{ak}}\right)} \tag{6.2}$$

where y_{ak} is the traffic count measurement on link a in time interval k, σ^2_{ak} is that measurement's error variance, and q_{ak} is the simulated value corresponding to that measurement. The condition $ak \sim i$ denotes that plan i crosses link a in time window k.

Intuitively, the offset is calculated based on how much this choice of the plan contributes to the whole traffic system's fitting to the traffic counts. Plans which traverse links where flow is underestimated are favored and vice versa, and σ^2 denotes the trust level that is put into the measurement – high trust levels lead to small values of σ^2 and thus to large correction terms.

This calibration can be seen as reducing uncertainty about travel behavior in the open choice dimensions, but it can also be applied to estimate the population size (Flötteröd and Liu, 2014) if each agent is given an additional, synthetic plan to do nothing, disappearing from the scenario.

6.3. From call detail records to a population of agents

A CDR dataset consists of records of the form

$$T_n := [(p_n, t_1, c_1), \ldots, (p_n, t_K, c_K)] \tag{6.3}$$

where p_n is a person identifier, t_k are timestamps, and c_k are cell tower identifiers.

As in Chapters 4 and 5, each trace T_n is converted into a travel plan in a straightforward way: Calls are converted into activities. Several calls in the same cell without a call in a different cell between them are fused, that is, they are converted into a single activity that starts no later than the first call and ends no earlier than the last call in the same cell. No additional activities are added. Activities are connected by trips (only the car mode is considered here). Congestion is disregarded. It is assumed that fastest routes on the empty network are taken. The only degree of freedom under consideration is the departure time from each activity location, which can be chosen anywhere between the time of the last sighting at location i and the latest possible departure time to make it to the next sighting location $i + 1$ in time.

The full agent population is constructed by expanding the population generated from traces. Specifically, we create C agents $A_{n1}, \ldots, A_{nC}$ per trace T_n. The agents are initially equipped with one random realization of the trace T_n, and over the iterations (cf. section 6.2.1), they create new random realizations, varying in time structure. In addition, they are given a special plan which, if chosen, lets them stay at home. Agents choosing the stay-at-home option are considered to be removing themselves from the simulation.

The resulting agent population is

$$A_{11}, \ldots, A_{1C}, \ldots, A_{N1}, \ldots A_{NC} \tag{6.4}$$

This expanded population is used as a buffer, which the calibrator uses to steer the demand towards matching the known link volume counts. The utility function is constructed so that, for each agent, the probability of choosing one of its travel plans is $p^0_{nc} = 1/C$, and the probability of choosing the stay-at-home-plan is $1 - p^0_{nc}$. In consequence, the prior expected behavior of the simulation is that the population size is N, and on average one instance of each trace is realized.[1] The hyperparameter c, the expansion factor, is selected by the modeller. It needs to be large if highly underestimated demand segments are to be compensated for, so that there is a sufficient number of individuals in the population to draw from.

The population expansion described here is a particularly straight-forward way of implementing uncertainty about the CDR sample in the MATSim-Cadyts-ensemble, because it reduces the estimation of weights, as well as which temporal realization of a CDR trajectory to use, to individual agent decisions. By calculating offsets to this prior utility of plans, the calibrator simultaneously adjusts the population size, the weights assigned to the individual traces, and the temporal realization of the trajectories.

This results in a distribution of individual choices among possible trajectories and stay-at-home plans. In particular, we obtain posterior travel probabilities p_{nc}. The sum over the posterior travel probabilities of the agents associated with trace T_n, $w_n = \sum_{c=1}^{C} p_{nc}$, is the expected number of instances of trace T_n to appear in any iteration of the calibrated scenario after achieving stationarity, and $(w_1, \ldots, w_N)$ is a weight vector with which the CDR dataset has effectively been resampled, a common concept in synthetic population generation, where a survey population is adjusted to fit exogeneously given marginal sums (Bar-Gera et al., 2009; Kagerbauer et al., 2015; Müller and Axhausen, 2010), whose role is in the present case assumed by the traffic counts.

The output of this step is of the same form as the ground truth scenario. The two scenarios can now be compared to assess the approximation quality.

[1] Initializing with $1/c$ is also plausible on theoretical grounds. The choice of prior probabilities adheres to the principle of maximum entropy for choosing Bayesian priors: The probability distribution of the number of travelling agents which realize CDR trajectory T_i prior to link counts is the binomial distribution $B(n = c, p = 1/c)$, which has maximum entropy among all probability distributions of mean one which are sums of c independent Bernoulli trials with arbitrary success rates $p_1, \ldots, p_c$ (Harremoës, 2001), i.e. all possibilities of initializing the travel probabilities p^0_{nc}.

6.3.1. Related work

Iqbal et al. (2014) convert traces into so-called transient O/D-Matrices, where every pair of successive calls in different cells generates an entry in a tower-to-tower matrix. As in the present approach, it is not attempted to detect trips. A pair of calls is only counted if they are more than 10 and less than 60 minutes apart. The resulting matrix is used as the initial demand of an iterative procedure involving a traffic simulator and link counts, similar to the present approach. In particular, the initial matrix is decomposed into two, where one has all O/D-pairs where the nodes are adjacent, and one has the rest. The iterative procedure finds two scale factors, one for each matrix. This is motivated by the need to eliminate false displacement, the phenomenon where a mobile phone oscillates between two neighboring towers without actually moving. The final matrix is taken as a calibrated demand.

6.4. Examples

6.4.1. Loop scenario

We consider a simple network consisting of only one home location, one work location, only one route connecting each location with the other, and a population which is divided into two segments of 1000 individuals each. One segment departs for work at 7am and one at 9am. The entire population leaves work and heads home at 5pm. All individuals make a phone call and produce a CDR precisely at the time they leave and arrive at their home location. Most individuals also use their phone at work and place calls when they arrive and when they leave, but members of the early-rising population segment do so only with a probability of 70%. This condition is designed to reflect the real-world case where a certain calling behavior is associated with certain kinds of travel behavior.

In the traffic demand reconstructed directly from the resulting CDRs, the non-calling sub-segment of the early-rising population will effectively stay at home, because their travel plan is constructed from an undersampled trajectory without a sighting at the work location. It does not contain a trip. This leads to an underestimation of the traffic demand from the home location to the work location at 7am to 700 travelling individuals, and from the work location to the home location to 1700 individuals.

Once adding a traffic measurement with the reference volume of $y = 1000, \sigma^2 = 100$ during hour 8, the observed population segment which leaves at 7am is scaled up by the calibrator to fit that number, compensating for those unobserved early-risers who do not use their phone at work (Fig. 6.1 top left). The validation measurement in the opposite direction at hour 18 follows (Fig. 6.1 top right). If $y = 2000, \sigma^2 = 200$ at hour 18 is chosen as the calibration measurement instead (Fig. 6.1 bottom row), meaning that only the total number of travellers is known but nothing from which relative population weights could follow, both population segments are scaled up proportionally.

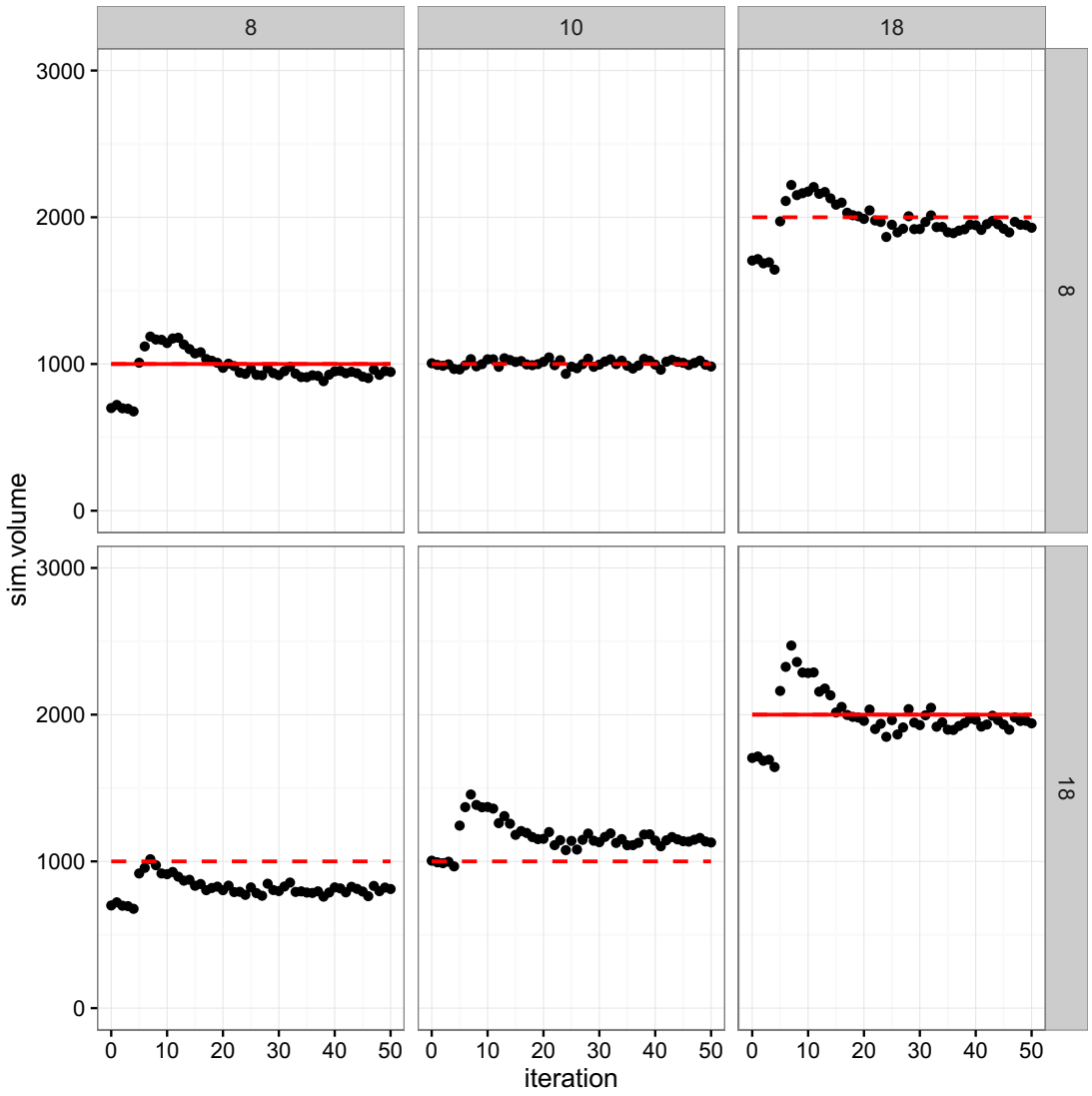
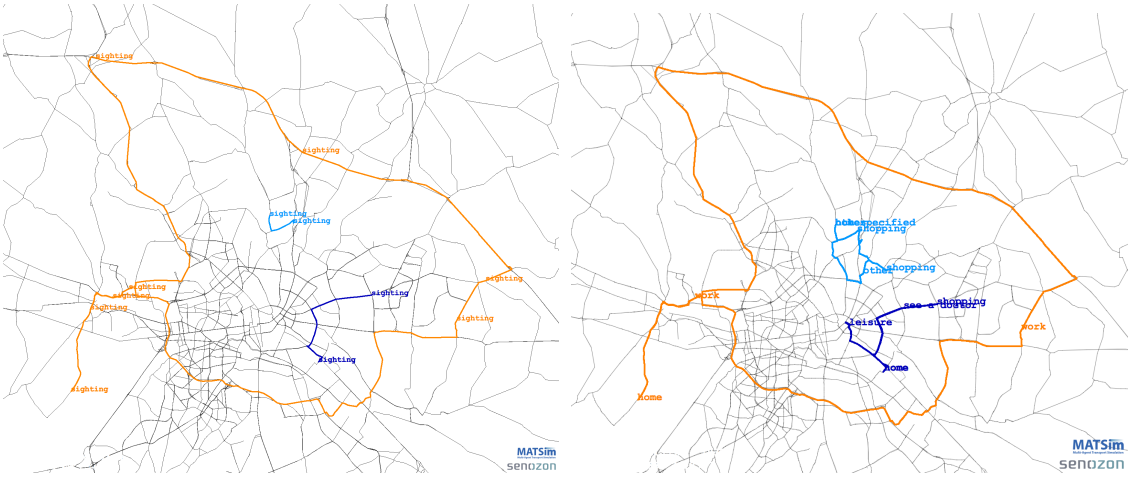


Figure 6.1.: Simulated link volume over iterations, at hours 8, 10 and 18. The red lines, dashed and solid, denote the real value. The calibration target (solid red lines) is the measurement at hour 8 (top) or hour 18 (bottom).



(a) (b)

Figure 6.2.: Travel chains for three different travellers (a) and the underlying ground truth behavior (b).

6.4.2. Berlin scenario

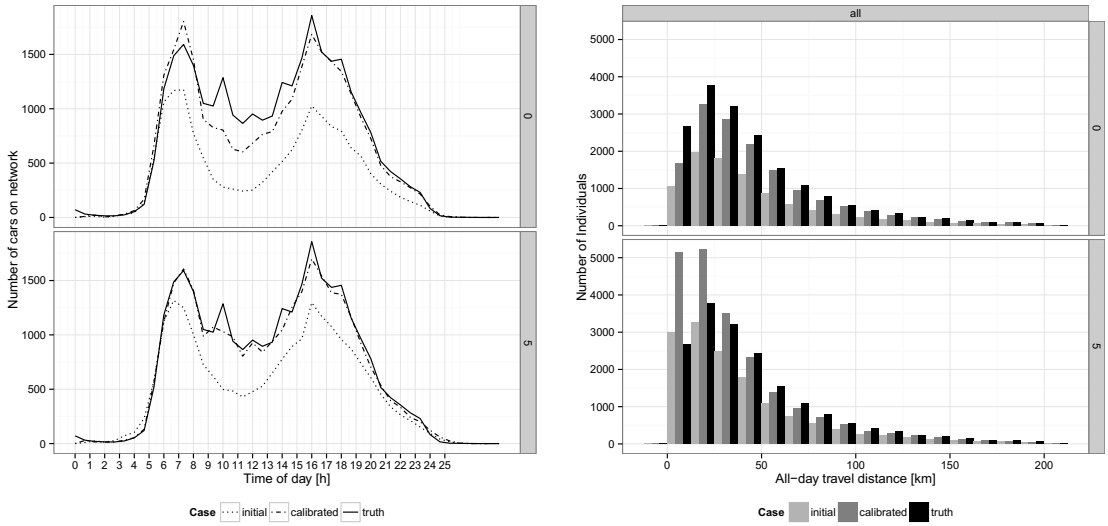
The scenario is the same as in Section 5.5.2. Agents place calls randomly at an individual daily call rate. Deliberately constructing a strong correlation between phone usage and travel behavior, we partition the agent population into two segments called workers and non-workers, where a worker is defined as an individual stating at least one work-related activity in the survey. The call rate of the workers is fixed at 50 calls per day (frequent callers), and that of the non-workers at 5 calls per day (infrequent callers).

Some travel chains obtained by the process are shown in Fig. 6.2a, while the true underlying behavior is shown in Fig. 6.2b. The orange plan contains a work activity, thus corresponding to a frequent caller. While the activity chain alone gives the traveller the freedom of many routes around and through the city, the sightings effectively pin one of the trips to the northern route. The two plans in blue do not contain a work activity, and are in consequence not sampled frequently. Several activities and related travel are missed. In fact, the light blue trace does not even result in a round trip any more.

With any mobile phone data set in hand, the modeller has to decide on a threshold how many calls per day are necessary for a trace so that it can be meaningfully included in the model input.

Using the binary-distributed synthetic data, we compare two options:

- Leave the sparse traces out of the simulation. This effectively means accepting a lower sample size and possibly introducing a bias towards a traffic pattern associated with frequent callers.
- Include the sparse traces even though their spatio-temporal resolution is such that they contain only limited information.



(a) Network load over time of day

(b) All-day travel distance distribution

Figure 6.3.: Network load and travel distance distribution for the scenario where the non-worker demand segment is missing (top) or represented by undersampled trajectories (bottom).

Fig. 6.3a shows network load over time for the initial situation where the population constructed from the available traces is simulated without adjusted weights, for the final estimation where the weights are adjusted towards fitting the link counts, and for the ground truth.

The first scenario shows the full effect of removing non-workers from the sample. In the initial estimation, there is too little traffic, but especially the load during mid-day is too small. In the final estimation, this gap is partly compensated for. In turn, the morning peak is overestimated, because there are only well-sampled traces of workers, which are mostly morning commuters, to draw from: In order to reduce the underprediction of mid-day load, the morning peak load has to be overestimated.

In the scenario where the traces of the non-workers, sampled at a low rate, are included, the final estimation has a closer fit to the ground truth (Fig. 6.3a bottom). In the initial estimation, the demand share generated from the undersampled non-worker traces is not only too low, but diffused over time (Fig. 6.4 right): Possible trajectories through few sightings have more temporal freedom than those through many sightings. In the final estimation, while still too low, its time structure more closely resembles the ground truth: The temporal uncertainty of the CDR data is reduced by taking the link counts into account. Intuitively, the sparsely sampled trajectories are fitted to that share of the measured volumes which is not accounted for by well-sampled trajectories. The overall final demand estimation is better because it now contains this time-adjusted non-worker demand as a component.

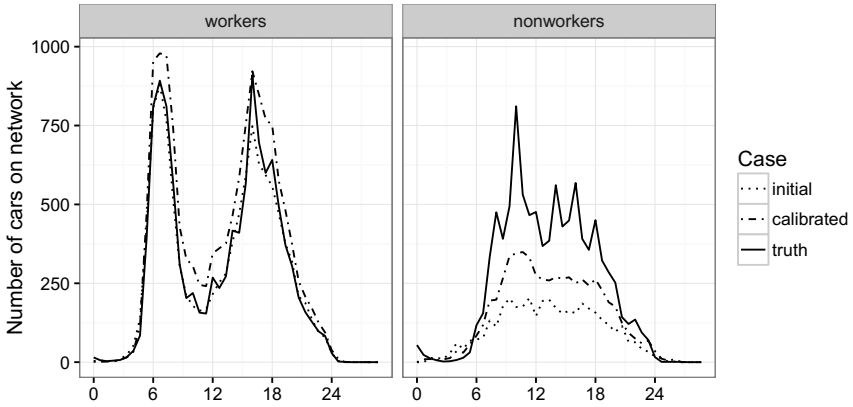


Figure 6.4.: Network load over time of day, separated by demand segments.

Considering the all-day travel distance distribution (Fig. 6.3b) reveals that it is distorted in both cases. In the first scenario, where the infrequent callers are excluded, the number of individuals travelling little is underestimated. There are at least two independent causes for this. The first is that workers travel more than non-workers, and traces of non-workers are missing by construction. Secondly, the estimation process itself is in this case biased towards far-travelling individuals: When the initial demand is too low overall, the contribution of most links to the Cadyts score correction (equation 6.2) is positive, so the utility offset of a plan is the larger the more links it crosses. In consequence, far-travelling agents will on average end up with a higher probability of travelling. This effect is absent when the initial demand is a priori scaled to the known change in sample size. But an alternative interpretation of this experiment is that a population segment is missing from the sample altogether, without this fact or indeed the true size of the travelling population being known to the modeller, so the initial demand was left unchanged here.

In the second scenario, where the infrequent callers are now included, the number of individuals travelling little is overestimated. Since the initial overall travelled distance is much closer to the truth, the calibration signal and hence the bias towards longer trips introduced by the plan correction is not as strong. It is dominated by an effect in the opposite direction which is created by the plan creation itself: Since the travelled distance of each plan is by construction at the lower bound of what is consistent with the sightings, the distance distribution is shifted to the left.

6.5. Discussion and Outlook

The preceding consideration points to a straight-forward way of improving this process. Even when there is no full travel survey available, which is the reason for looking at CDRs in the first place, there are often some aggregate data, or marginal sums, like the travel distance distribution we just considered. These can be directly fused into the

model. A histogram bin is exactly analogous to a link volume count. It is known for each plan if it belongs into it or not, just like it is known if a plan will travel over a link in a time window. If we add a measurement for each histogram bin into the calibrator, it will calculate a utility offset for each plan which belongs into it, and it will steer the population to fit the histogram more closely.

The loop scenario illustrates how CDR trajectories carry information for a fully agent-oriented traffic model to make use of which O/D matrices do not: Trajectories contain an all-day trip structure, which is preserved by the simulation. We only need one measurement to influence traffic in both directions. Conversely, this means that eventually, our restriction not to add additional feasible activities between confirmed sightings may not be adequate. The framework of time geography allows to pick any trajectory with or without additional activities within the constraint of the confirmed sightings. The generation of new plans can be incorporated in the simulation loop, so that agents draw from the whole set of feasible trajectories. However, in this case the distribution of the number of activities should be available and integrated. Otherwise, there would be no behavioral justification for taking detours: Agents would make additional trips just to satisfy link counts.

It should be emphasized that the weights resulting from this estimation cannot be directly interpreted in terms of population segments: They do not weigh individuals nor travel plans, but possibly undersampled trajectories. Trajectories which are undersampled to the point that they do not resemble a yet missing demand component with respect to traffic counts will end up with a low weight, even if they come from an underestimated population group.

In our exposition as well as in the experiments, we focussed on the temporal dimension of uncertainty in the CDRs by identifying cells with links. The concept of making calibrated draws from the possible realizations of a CDR works with arbitrary cell sizes.

We used extreme call rates of 5 and 50 to specifically demonstrate how the system reacts if a specific traffic segment is underestimated or missing. While average phone call rates of 50 calls per day are certainly not realistic, these cases are still worth considering even outside of illustrative scenarios, because in practice, data points similar to CDRs need not be caused by actual phone calls, but can also appear as a consequence of, for instance, internet usage. We consider the terms CDR, call rate, and cell, to be interchangeable with corresponding concepts in other current or future technologies which produce trajectories.

6.6. Summary

We formulated the problem of fusing CDRs with traffic counts as a reduction to the calibration of individual choice probabilities in an iterated dynamic travel assignment scheme. The approach thus inherits known properties from the mobility simulation and from the calibrator.

Our experimental scenario illustrates two cases:

- When a large population segment is missing or removed from the CDR sample

because of its low daily call rate, the remaining sample is scaled up and reweighed in the process to fit link counts.

- When the same population segment is kept in the sample, represented by sparse traces generated by only 5 calls per day, the process is able to reduce the resulting temporal diffusion by producing trajectories which are more consistent with the traffic counts. This case yields a better fit to the real traffic flow.

Overall, the results demonstrate that even a heavily biased cell phone dataset, together with anonymous traffic measurements, can be used to re-construct the traffic state over time quite well. Any algorithm which attaches behavioral interpretation to a CDR trace can be used in the plan generation step to enrich the model.

7. From traces to plans

In the preceding considerations and experiments, the question of how a single trace is enriched to a plan has been touched upon only briefly. Following a minimalist approach, we simply "connected the dots" without allowing for activities unconfirmed by sightings. We introduced demand elasticity by effectively giving each agent a weight parameter, and found that when this model is fitted to link measurements, the overall traffic volume follows. This approach introduces a bias in the direction of shorter-travelling agents and a larger effective population size.

Still, it may be sufficient if the goal is to estimate the state of the traffic system, or for short-timeframe predictions, such as the reaction of the system to an unplanned disturbance. In this case, the agent-oriented model only uses the current locations of all simulated drivers and their next destination. The procedure described so far may already produce these items sufficiently for many such scenarios.

However, for other applications, a more realistic synthetic population resembling one generated from surveys is required. At least, the population size estimation and the distribution of all-day travel distance should not already be biased *by construction*. Until now, the plan distribution from which a single agent draws can be characterized by having a minimal set of activities covering the sightings, and shortest-path legs connecting them. In particular, there are no activities which do not cover a sighting.

The guiding question of this chapter is how to generate a plan whose number of activities and total trip-length are not by construction correlated with the number of sightings.

A trace constrains the possible movements of an individual. The potential path area (PPA) (Hägerstrand, 1970) is the area to which the possible movement of an individual between two sightings is confined, given the trace and a maximum velocity. Constructing a feasible plan amounts to filling the PPA between each pair of sightings, using all degrees of freedom.

We call a plan satisfying the constraints set by the sightings *feasible*. In principle, given a trace, the task of producing a feasible plan can be achieved by an algorithm of the following form:

1. Construct an arbitrary plan.
2. Accept if it is consistent with the trace, reject otherwise.
3. Repeat until a plan is accepted.

When iterated, this framework amounts to rejection sampling (Flötteröd et al., 2011a): The construction of the proposed plan, in whichever way it is done, is a draw from a prior

distribution of plans, $p(i)$. Repeated drawing from $p(i)$ while rejecting incompatible plans amounts to drawing from the conditional distribution of those plans that are feasible.[1]

This approach is impractical, firstly because of the magnitude of the space covered by a general proposal distribution. Creating plans by randomly choosing activity locations in a study area and accepting those plans that happen to be consistent with a given trace will lead to an impractical or practically infinite number of rejections. Secondly, it is not clear how to construct the proposal distribution:

One possibility is to use an empirical set of plans as the prior distribution, for example taken from a conventional trip diary survey. However, the motivating scenario for this work is that an empirical set of plans is not available.

However, the higher the density of sightings in a trace is, the more it already resembles a plan. Looking at traces of increasing density, a human analyst or an algorithm will be able to infer features like travelled distance or number of activities with increasing accuracy (see Chapter 5).

Thus, it seems viable to define a criterion, such as being over some threshold value for the frequency of sightings, and use the subset of traces which satisfy that criterion to construct the prior set of plans with a simple heuristic such as the one discussed in Chapter 6.

Still, using such an empirical distibution of plans as the prior distribution to draw from has limitations: It is likely that, under realistic sample-sizes, this strategy would often not find an acceptable plan for a sparse trace at all, no matter how many draws are taken.

Secondly, this procedure would amount to a re-weighing of an empirical sample of dense plans. As such, it seems unlikely that the sparse traces would add much more information compared to the more parsimonious alternative of only using the link counts for re-weighing.

The other possibility is to produce fully synthetical plans using a model which reproduces marginal statistics of the set of traces, such as average travelled distance, detour factor or population densities. Rejection-sampling from this model gives a population which is consistent with the actually given traces while maintaining degrees of freedom.

Somewhere in between these possibilities is using an empirical set of plans as a foundation for the proposal distribution, but instead of drawing from it directly, using its plans as templates and manipulating them in some dimensions, for instance altering the activity times and locations. This is the approach investigated in this chapter.

Relatedness to survey inflation The problem is similar to the one faced in initial demand generation from a survey population, and the possible solutions are analogous. There, the problem is that the sample size may be too small to, for instance, have at least one simulated person per building block. One possible solution is to use the survey sample to estimate parameters of a stochastic, plan-producing model, then taking a sample of

[1] Note that feasibility is a simulation-based constraint, not an analytical one. If a plan is feasible or not depends on endogeneous simulation parameters, particularly link conditions. Compare Zhang et al. (2017)

the desired size from that model (Müller and Axhausen, 2010). Another solution is to inflate or clone the survey population. This refers to using several copies of the same plan, but not verbatim, but varied in one or several dimensions. For example, the activity locations may be blurred within some specified radius or within a traffic zone, or the time structure may be altered (Kickhöfer, 2014).

These considerations lead to the following generic algorithm:

1. Construct a plan from each dense trace.
2. Inflate this set of plans, with spatio-temporal randomization.
3. Use the inflated set as the prior plan distribution for the above algorithm.
4. If this fails to produce plans with enough variability to cover each individual in the set of traces with a satisfying acceptance rate, repeat with relaxed conditions (more randomization) for the plan inflation.

An operational implementation, however, cannot be made of this principle, as acceptance rates would still be too low in practice. Also, since the inflated plan set is in any case rather a heuristic than a draw from a carefully constructed distribution, it seems reasonable to depart from the rejection-sampling principle and instead construct a plan to fit right away the trace which is to be covered.

It was decided to do the inflation in trace-space rather than in plan-space:

1. Inflate the set of dense traces by *enriching* the sparse traces using the dense traces as templates. The enriched sparse traces contain all the sightings of the original trace plus additional ones.
2. Construct a plan from each trace.

In the next section, an implementation of this scheme is described.

7.1. Constructing plans

The input to the algorithm is a set of traces. First, it is partitioned into sparse traces and dense traces. Dense traces are traces where it is assumed that their sampling rate is sufficient so that it gives an account of all or most of the locations the subject has visited. How traces are to be classified in an empirical dataset is left open here. A threshold for the number of calls per day could be used. In the artificial data used here, this partitioning comes by construction. The outline of the algorithm is:

1. For each sparse trace, construct a new trace by the following procedure:
2. Draw a random sample from the set of dense traces.
3. Rank the elements of the sample by a similarity measure to the sparse trace. Choose the closest one.

4. Map the locations witnessed by the sparse trace to locations on the dense trace.

5. For each location witnessed by the dense trace, insert a new sighting into the sparse trace, choosing a location so that the movement profile of the constructed trace matches that of the dense trace.

Related work The procedure was inspired by Schneider, 2011, which is concerned with freight vehicle chains. For a synthetic population of firms and households with locations, freight vehicle trip chains are constructed by geometrically transforming samples from a database of observed chains. A freight chain originating from a firm is produced by picking a template chain from the database which matches the attributes of the firm, anchoring it at the location of the firm, and then assigning template destinations to real destinations around the location such that the spatial profile of the template chain, as characterised by trip length and distance from home, is not distorted by more than a specified maximum error. If that is not possible, backtracking is performed, first by making alternate location choices for preceding destinations in the chain, and eventually relaxing the requirement that the template chain match the attributes of the firm.

7.1.1. Similarity

The distance from the starting point, graphed over the time of day and linearly interpolated over time between sightings, is a simple projection to visualize a human trajectory. It was used, for instance in the original Location History application by Google, which has since been renamed Timeline and, somewhat ironically, does not contain distance-time graphs anymore. The starting point is the location of the first sighting of the day. In the following, the expression "distance from home" is used in that sense. It is not implied that the actual home location of an individual is inferred.

We calculate a dissimilarity measure μ between the distance-from-home profile $f(t)$ of a sparse and that of a dense trace $g(t)$ as follows:

1. Let $\hat{g}$ be g resampled at the sighting times $t_1 < ... < t_n$ of f. Now f and $\hat{g}$ are aligned.

2. Let μ be the area between between f and $\hat{g}$.

The area between these piecewise-linear functions can be computed by summing up the contributions μ_i of every interval t_i, t_{i+1}, and they are

$$\mu_i = \begin{cases} (t_{i+1} - t_1)\dfrac{(\hat{g}(t_i) - f(t_i))^2 + (f(t_{i+1}) - \hat{g}(t_{i+1}))^2}{2(f_{i+1} - f_i - \hat{g}(t_{i+1}) + \hat{g}(t_i))} & \hat{g}(t_i) > f(t_i),\ \hat{g}(t_{i+1}) < f(t_{i+1}), \\ (t_{i+1} - t_1)\dfrac{(\hat{g}(t_i) - f(t_i)) + (\hat{g}(t_{i+1}) - f(t_{i+1}))}{2} & \hat{g}(t_i) > f(t_i),\ \hat{g}(t_{i+1}) > f(t_{i+1}). \end{cases}$$

with symmetrical expressions for the remaining cases.[2]

[2] See Agarwal et al., 2010, which discusses similarity up to scaling and translation and an extension to higher dimensions, but helpfully includes the simple case required here.

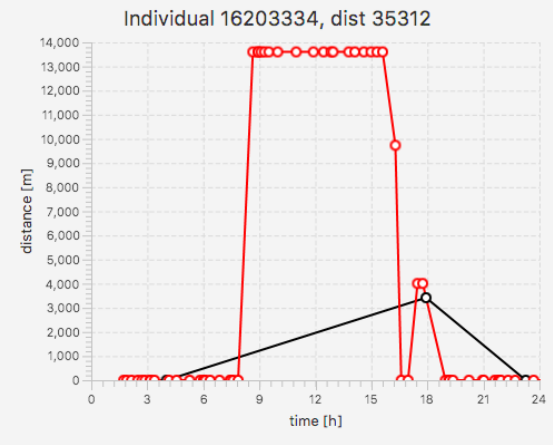


Figure 7.1.: Distance-from-home plot for a sparse trace (black), and a dense trace (red) which is similar in terms of the distance-from-home measure.

Informally, this measures the distance between f and g disregarding what g does between the vertices of f (Figure 7.1). If the agent which produced g had placed phone calls at the same points in time as the one which produced f has, then f and g would have a high degree of similarity. The goal of the next step is to construct an *enriched* trace whose f then resembles g also in these intervals.

7.1.2. Enrichment

Consider a sparse trace f and a dense trace g with reasonably small μ to f. We now take f, g etc. to be the full location over time, in contrast to the previous section, where we used the distance-from-home projection. An enriched version $\hat{f}$ of f is created, using g as a template.

1. Initialize $\hat{f} \leftarrow f$. Let $t_1 < ... < t_n$ be the sighting times of f, and $u_1 < ... < u_m$ the sighting times of g.
2. For each (t_i, t_{i+1}) of f:
3. Calculate a similarity transformation T_i which maps $\begin{pmatrix} g(t_i) \\ g(t_{i+1}) \end{pmatrix}$ to $\begin{pmatrix} f(t_i) \\ f(t_{i+1}) \end{pmatrix}$. It is unique up to reflection by the perpendicular bisector of $\overline{f(t_i)f(t_{i+1})}$. We arbitrarily choose the orientation-preserving case, but could also randomize by choosing between the two. Note that this step also uses the resampling of g at the time steps of f.
4. For all sightings $g(u_j)$ with $t_i < u_j < t_{i+1}$, let $\hat{f}(u_j) \leftarrow T_i(g(u_j))$, inserting the transformed sightings into $\hat{f}$.

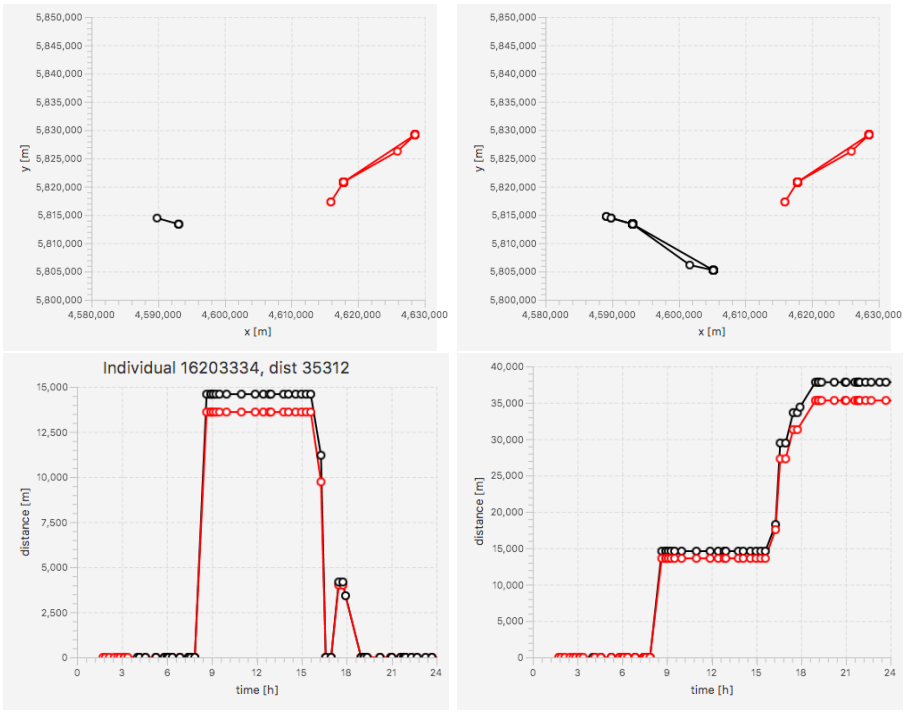


Figure 7.2.: Top: The same traces as in Fig. 7.1 in space, before (top left) and after enrichment (top right). Bottom: After enrichment, the formerly sparse trace closely resembles the dense trace, both in terms of the distance-from-home function (bottom left) and the cumulative travelled distance function (bottom right).

Figure 7.2 shows the effect of this operation. Choosing a g with a relatively small μ has the effect that each T_i is relatively length-preserving.[3] Since the procedure inserts the images under T_i of detours in g between t_i and t_{i+1} into f, this brings the all-day travel distance of $\hat{f}$ close to that of g.

We incorporate this procedure into a simulation run, building on what was described in Chapter 6. Enrichment is used as a replanning strategy. When this strategy is chosen for an agent, a new random realization of that agent's trace f is constructed:

1. Draw a random sample, e.g. 10 percent, of all dense traces.
2. From that sample, choose the dense trace g with the best μ.
3. Enrich f to $\hat{f}$ with g as a template.
4. Randomly choose a temporal realization of $\hat{f}$.

The random sampling from the population of dense traces (item 1) serves two purposes: Firstly, it saves computation time, since computing the similarity measure is an expensive operation. Secondly, it randomizes the procedure, so that there is variability in plan generation. Otherwise, only the most similar dense trace would always be used for each agent, and no variability would be seen except in temporal structure. In particular, the aggregate all-day travelled distance would again be fixed except for re-routing, only at a higher value.

7.1.3. Experiment

Basic experiment We divide the population into two segments, heavy-callers and little-callers. In contrast to the run described in Chapter 6, these are not distinguished by being workers or non-workers, but randomly divided. We generate synthetic traces like described previously, and run the simulation loop, iteratively constructing plans using the trace-enrichment procedure described in this chapter.

Population cloning is switched off. The idea is that we now have a process which is at least not obviously, by construction, skewed towards producing plans with too little travel. For that reason, it should not be necessary to compensate for that by scaling up the population size. Instead, the population size is considered known and fixed at the ground truth value.

We compare the results against a re-run of the cloning procedure from Chapter 6 on the now randomly-divided population. The all-day travel distance distribution is still skewed towards too little travel, though to a somewhat lower degree (Figure 7.3, top row). The leftmost histogram bin also contains individuals who do not travel at all. Especially this group is still overrepresented, even after trajectory enrichment. The overall traffic volume is reproduced reasonably well (Figure 7.4, top row).

[3] A person going around their home in circles can trick the measure described here. In case this turns out to be a problem, it can be modified to also look at the cumulative travelled distance.

Histogram correction Building upon the outlook in Section 6.6, we run another experiment where we now consider the all-day travel distance distribution known from a survey, with each histogram bin corresponding to a measurement. The measurements are, for simplicity and applicability of the Cadyts approach, assumed to be independent. A term is added to the score formulation of Equation 6.2:

$$P(i|y,z) = \frac{\exp\left(V_i + \sum_{ak\sim i} \frac{y_{ak}-q_{ak}}{\sigma_{ak}^2} + \sum_{a\sim i} \frac{z_a-q_a}{\tau_a^2}\right)}{\sum_j \exp\left(V_j + \sum_{ak\sim j} \frac{y_{ak}-q_{ak}}{\sigma_{ak}^2} + \sum_{a\sim j} \frac{z_a-q_a}{\tau_a^2}\right)} \tag{7.1}$$

where z_a is the measured number of individuals in histogram bin a, τ_a^2 is that measurement's error variance, z_a is the simulated value corresponding to that measurement, and $a \sim i$ runs over the (single) histogram bin a into which i belongs. The role of link passages is taken by the histogram bins. For the error variances, both of the link passage counts and the histogram bin counts, the default assumptions of the Cadyts implementation are used.

We re-run both cases, the cloning case and the trajectory enrichment case, with this additional scoring term. In the experiment, both sum terms are multiplied with a weight factor of 100 to get a strong response: It is assumed that we have little prior behavioral information in V_i and simply want to fit a travel demand consistent with the sightings to the histogram and the counts. Figure 7.3, bottom row, shows that these runs, in both cases, closely reproduce the histogram of the ground truth.

Note that this procedure addresses the problem of calibrating the enrichted trajectories against any marginal distribution, since a term for any marginal distribution can be added in the same way. This holds even for multi-dimensional marginals, and even for cases where not even a full marginal distribution is known, but only a single value, for instance, the number of people who use public transit in the city center.

7.1.4. Activity locations

Many traffic simulation applications require at least some behavioral information attached to the trajectory. For example, to model a traveller who reacts to a network disturbance, such as a closed road, by taking another way, it needs to be known which observed visited locations are significant activity locations, and which are merely waypoints with no explicit utility to the traveller.

The procedure described so far can be extended with a simple activity detection procedure. Since it operates on traces, it is applied after the trace enrichment step.

When a temporal realisation of a trace is constructed, there is some freedom when determining the departure time at a sighting location: It must be later than the latest instant at which the presence of the agent at the location is witnessed, and earlier than the latest possible departure time required to reach the location of the next sighting in time. This slack can be very small, or indeed negative, when the random trace enrichment draws a trace which is not realizable under the present network travel times.

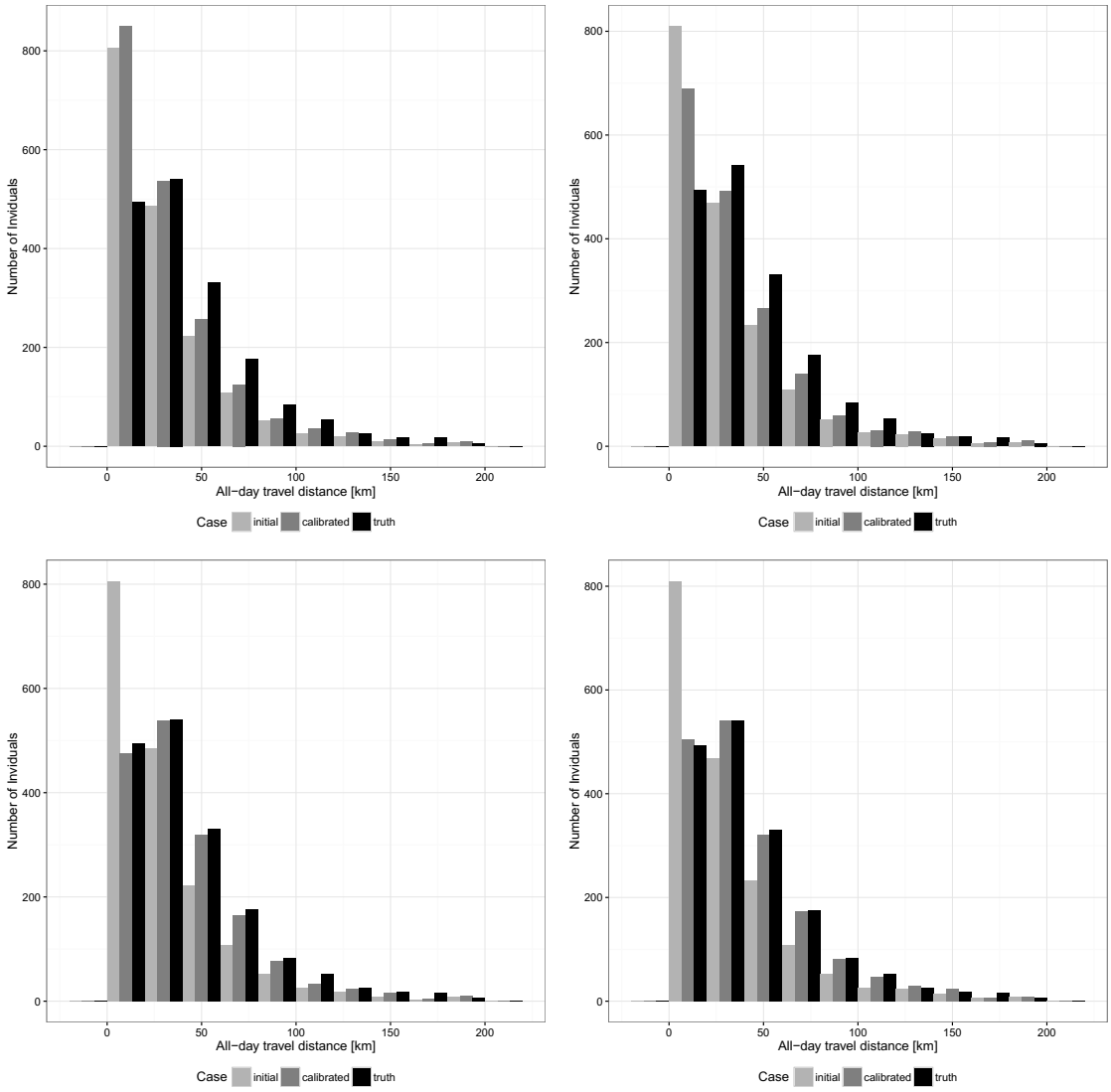


Figure 7.3.: All-day travel distance histogram over people. Top left: cloning (Chapter 6); top right: enrichment (this chapter). With the enrichment procedure, the distribution of the total all-day travel distance over the population is still distorted, albeit less heavily than with cloning. Especially agents who travel little or not at all (leftmost bin) are still overrepresented. Bottom left: cloning and histogram correction; bottom right: enrichment and histogram correction. The histogram correction achieves a very good fit in both cases.

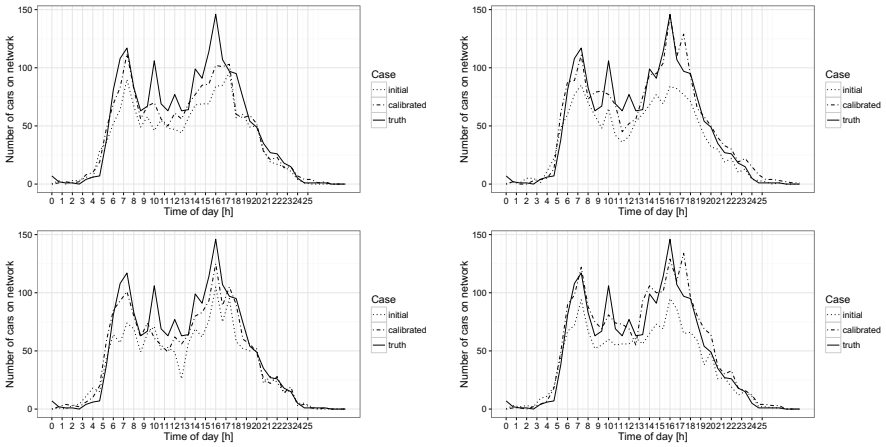


Figure 7.4.: Traffic volume over time of day. Top left: cloning; top right: enrichment; bottom left: cloning and histogram correction; bottom right: enrichment and histogram correction. Trace enrichment gives a consistently better fit than cloning.

An agent placing multiple calls while on a car trip will produce a string of activity locations with very small slack. Constructing a feasible plan for this trace will always produce activities with very small duration for these sightings. This intuition leads to the following simple criterion for distinguishing real activity locations from waypoints:

> After constructing a feasible plan, consider all planned activities with a very short planned duration to be waypoints. Remove them from the plan.

An example of the joint effect which the sampling by sightings, the random realization, and the simple activity detection has on the activity structure of an individual is given in Figure 7.5. For illustration purposes, the traces shown there are all dense traces, so no enrichment is taking place.

Note that this method is biased in that it will never recall real, short activities. Instead of using the randomly drawn activity duration as the value compared against the threshold, one could use the expected duration across several random realizations of the trace, so that only activities which are *necessarily* short are removed. Waypoints could additionally be required to be close to the least-cost path from the previous to the next location.

In contrast to GPS-based activity detection schemes, which often use a threshold number of successive measurements at the same location to classify activities, our scheme is aimed at sparser traces and assumes that few activities will be witnessed by more than one sighting. Accordingly, it takes any sighting to be a potential activity, which are only rejected when there is evidence that they are a waypoint. Randomizing this procedure would enable us to fit the demand to a known distribution of the number of daily activities by adding a scoring term as in Section 7.1.3. Note that this procedure

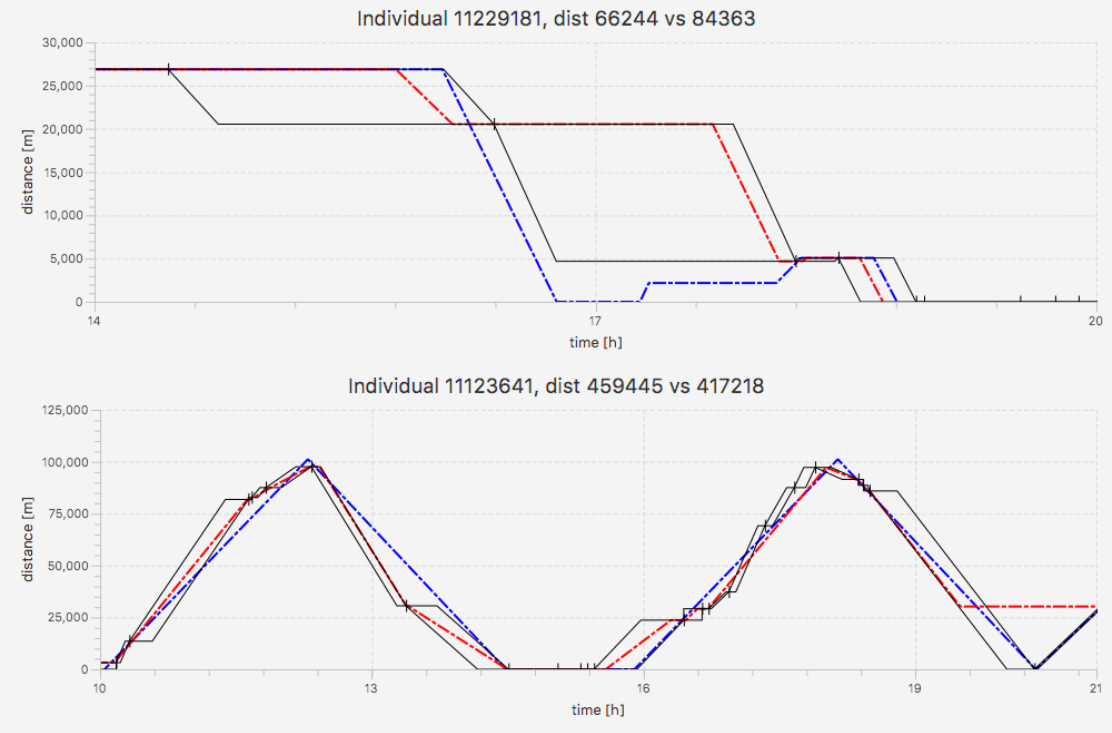


Figure 7.5.: Ground-truth trajectories (blue) and one random realization each (red), projected to the distance from the initial location. Trips are interpolated with constant speed. Constant intervals in the graph are segments without movement (activities). The black vertical tick marks are sightings, which, whilst on a trip, do not coincide with the ground-truth trajectory because of the constant speed interpolation. The black polylines are constraints for the trip to the next location, induced by the sightings. When the red line shortcuts a sighting location, the algorithm classified that sighting as an intermediate waypoint.

is sensitive to network travel times and will pick up evolving travel time estimates in a congested scenario.

The scheme was tested on an uncongested scenario with the entire population sampled at 50 sightings per day, without enrichment, to evaluate it in isolation. A threshold of 10 minutes was used for the activity duration. Shorter activities are classified as waypoints. This yields 63913 trips, compared to 72861 trips without a cut-off, and 62749 trips in the ground truth.

7.2. Discussion and Outlook

The next step after activities and legs are identified is to label activity types and modes of transport, which is a large topic on its own (see e.g. Chen et al., 2014). Statistical data on their distribution can be used as a side-input and put in the simulation loop, similar to the distance distribution.

The likelihood of a plan given a trace is currently binary: either a plan is feasible or it is not, and only feasible plans are created. Given a probability distribution of traces given a plan (from a model of actual cell usage of moving devices), the likelihood of an individual plan given an individual trace could be added to this model as yet another scoring term. In a similar way, even the likely correlation of call frequency and activity type, an issue not addressed in this work, could be taken into account. This is likely to be not as important in the future: Currently, location datapoints in typical datasets are caused by call activities of users, but that may change. For datasets which are based on location area codes (LAC) and contain handovers between clusters of cells, this is the case even today: Datapoints are caused by the people crossing location area boundaries, giving a more direct account of mobility behavior but with a decreased spatial accuracy.

The procedure introduced in Schneider, 2011, which constructs freight chains and inspired the work in this chapter, has some resemblance to the one presented here, but also some differences. The only attribute considered in this chapter is similarity with respect to the spatio-temporal profile. The plan is not anchored at one fixed point (the firm's location), but at each sighting, and the intermediate points are constructed by explicit geometric transformation rather than by looking for candidates and checking for spatial distortion. Hence, no backtracking is required. On the other hand, the inserted waypoints are points in continuous space. They are not drawn from a candidate pool, and in fact I do not concern myself with the geographic reality at this point – there may well be a lake at the inserted activity location. It is assumed that coordinates are assigned to locations by a downstream step in the mobility simulation, where the nearest road segment is picked.

The weights only work as heuristic correction terms. It is not claimed that the final distribution is, in fact, the original distribution scaled by those weights. For that, we would need the proposal probabilites of the plan constructor, and we would need to ensure that each possible plan will be proposed with positive probability. The analogous problems exists in sampling network paths. A rigorous solution for this case, using Metropolis-Hastings sampling, is presented in Flötteröd and Bierlaire, 2013, where it is also pointed out that it should be feasible to find a similar method for constructing all-day travel plans. Like in that paper, it would entail giving the Metropolis-Hastings modification operator parameters with which the process can be steered to the region of interest in state space (where the feasible plans are), and calculate the transition probabilities based on those parameters.

7.3. Summary

We presented a Monte Carlo method for constructing plans with the following features:

- Takes a set of mobile phone traces as its only required input.
- Can be put in the loop of an iterated dynamic traffic assignment and a transport behavior model.
- Produces plans which are, on a person level, consistent with the traces.
- Tested with a binary criterion for the consistency of a plan to a trace (a plan is either feasible or not), but can use any non-normalized likelihood function for a plan given a trace.
- Desired population-level or person-level sampling probabilites can be specified as non-normalized weights.

Part III.

Modularity

8. Adding freight traffic to MATSim

A previous version of this chapter was published as Zilske et al. (2012). Parts of the material are based on the following previous publications: Schröder et al. (2012) and Agarwal et al. (2015).

8.1. Introduction

MATSim is mainly designed for large-scale simulations of personal vehicle traffic in urban areas. Commercial vehicle traffic has usually been modeled as a background network load without much adaptive behavior and without taking into account its distinct physical characteristics such as lower speed and higher road capacity consumption. This chapter explores the integration of freight transport into MATSim. It focuses on freight-related adaptations to the software and on behavioral aspects in relation to the freight agents.

8.2. From people with activity plans to freight operators with schedules

8.2.1. Passengers

The travel demand model implemented in MATSim consists of a set of agents representing individual users of a traffic system. Every agent is equipped with a plan, which describes locations, times and types of all the activities the agent will conduct, with legs connecting each physical activity location to the next. Each leg is travelled using a specified transport mode and, depending on the transport mode, along a specified route through the transport system.

All agents simultaneously execute their plans in a concurrent simulation of the transport system. The simulation picks up congestion effects, missed public transit connections, and delayed arrivals at activity locations. The results of the simulation are fed back to the agent as observations, and they are used to evaluate the plan using a scoring function which can be personalized for each individual, for example by depending on their age or income. In reaction to the traffic flow simulation, some agents modify their plan using one of several re-planning modules, which correspond to the choice dimensions available to the agent. This includes choosing a new route, switching transport modes, and choosing new times for activities.

Simulation, scoring and re-planning are iterated until a relaxed state resembling a dynamic user equilibrium is reached. The planning and re-planning model employed here is tailored to the daily schedules of private passengers who use the transport system to get

from home to work and to leisure or shopping locations: Agents might re-schedule shopping trips to avoid rush-hour congestion, or switch to public transport for commuting. Additionally, plans wrap around over night: It is assumed that the last activity of the day is the same as the first (usually: being at home), and in consequence, the starting time of the first activity is the same as the starting time of the last activity of the day. The advantage is that one does not need to make special assumptions for the temporal boundaries of the system, similar to periodic spatial boundary conditions in materials science simulations.

8.2.2. Carriers

Up to now, real-world scenarios set up with MATSim have modeled the freight traffic share of the demand by using a set of plans with activities at the depot and at pick-up and delivery locations, but with no variability in any choice dimension except route choice. We improve on this situation by modeling freight vehicles as non-autonomous agents employed by and serving the interests of freight operators. The missing choice dimensions of freight vehicle drivers are then realized as logistics decisions made by the freight operators who employ them. In the freight transport sector, decisions are distributed among actors with different roles. Freight transport decisions include lot-size choice, path-searches in logistical networks, vehicle choice and tour planning. The planning problem of a freight operator is therefore quite different from its passenger counterpart: Firstly, the success of freight transport plans is not determined by the utility of time spent at activity locations, but rather by their commercial success. They have to meet the requirements of customers, like meeting time windows and providing enough capacity at a reasonable cost. Secondly, freight operators often do not operate one single vehicle but several, and their options include rescheduling deliveries from one vehicle to another or even changing the number of vehicles which are used at all.

Therefore, a new software layer populated by carrier agents was introduced into the simulation. Each carrier agent represents a firm with a vehicle fleet, depots and contracts. Contracts determine type and quantity of goods to be carried. A contract contains the respective origin and destination as well as pick-up and delivery time windows. The plan of a carrier agent contains a schedule of a tour for each of the vehicles in the fleet. The schedule contains planned pick-up, delivery or arrival times at customer locations and a route through the physical network. In our basic model, all vehicle schedules of a carrier begin and end at one of its depots.

When a simulation scenario is initialized, the carrier agents build a schedule for each of their vehicles, including a route through the transport network, with pick-up and delivery activities corresponding to their contracts. At the interface between the freight operator plans and the mobility simulation, the set of routed vehicles from each carrier plan is injected into the traffic demand as individual driver agents, where they move through the traffic system along with passenger vehicles. While executing their plans, the freight driver agents report their shipment-related activities back to the carrier.

When all plans have been executed, agents evaluate the success of their plan. The carrier agents use a custom utility function that captures their economic success. Their

cost is calculated as a sum of vehicle-dependent distance and time costs incurred by their scheduled vehicles, and some individual fixed costs, plus penalties incurred by missed time windows. Finally, carrier agents create new plans to improve their performance in the next iteration. For instance, a time dependent vehicle routing heuristic can be plugged in to replan vehicle schedules. Shipments can be switched between vehicles, or even an entire vehicle added or removed. During repeated executions of their plans, passengers as well as carriers collect experience from the transport system. The carriers pick up congestion and other disturbances in the traffic system when they incur a higher cost through longer vehicle usage, or by penalizing missed pick-up and delivery times.

8.2.3. Implementation

MATSim has a modular and extensible architecture, allowing users to plug in custom reporting tools as well as specialized re-planning modules. The traffic flow simulation delivers its output as a stream of events which is used as input by other core parts of MATSim as well as by user-provided code. While implementing the freight extension, we took care to avoid tight couplings to the rest of the system as well as changes to the core code. The interface between MATSim and the freight extension could be kept conveniently small:

1. The carrier agent layer produces a population of freight driver agents which are injected into the mobility simulation upon start.

2. It listens to simulation events, keeping track of the distance travelled by the freight drivers and of the experienced pick-up and delivery times.

3. After each iteration of the mobility simulation, any number of user-provided scoring and re-planning modules for the carriers are called. They can make use of the time-dependent link travel times from the mobility simulation. Re-planning modules could, for example, employ dynamic vehicle routing algorithms to optimize tours based on the experienced travel times. Another re-planning module might be a transport market model, which would redistribute contracts among carriers. In principle, anything which affects the plans or contracts of a carrier can be plugged in.

It was not required to make any changes to the mobility simulation or even to introduce custom code for the freight driver agents. As far as the mobility simulation is concerned, freight drivers execute routes through the network, just like private vehicle operators. Changes to the mobility simulation would indeed be required for custom within-day behavior as opposed to plan-following. For example, the current implementation does not allow for a freight vehicle to wait if the goods it is supposed to pick up are unexpectedly not available yet due to traffic conditions on the upstream leg. It is assumed that the penalty of being late in one leg of a multi-leg delivery is avoided as an outcome of the evolutionary algorithm, and that in a relaxed system state, the planned pick-up time for a shipment is no earlier than the time at which it becomes available.

8.3. Traffic flow simulation with vehicles of different speed

The queue model of traffic flow implemented in MATSim (Cetin, 2005) (Gawron, 1998) is designed to be computationally faster than car-following models. It operates on the principle that the time a vehicle spends on a link is split between moving to the end of the link and waiting in a queue. When a vehicle enters a link at time t, its earliest link exit time $t_{exit} = t + \Delta t$ is determined, where Δt is the time it would take a vehicle to pass the link under free flow conditions. Vehicles exit a link according to the following rule: In every time step t, vehicles whose t_{exit} has passed are removed from the head of the queue, but only as many as the capacity per time step cap of the link permits, and only if there is enough space on the next link. In (Cetin, 2005), the free flow link travel time of a vehicle is simply $\Delta t = l/v_0$, where l denotes the length of the link and v_0 the free-flow velocity permitted by the link, which is the same for all vehicles. However, the same source already notes that links need not be first-in-first-out queues, but priority queues in which vehicles are sorted by their t_{exit} value, and that Δt can, in principle, be any function of the current link condition. If we keep it simple and assign to each link a free-flow velocity $v_0(veh)$ depending on the vehicle type, this already has the following consequences:

1. Faster vehicles pass slower vehicles under non-congested conditions. A link is non-congested if every vehicle leaves the link exactly at the moment when its free-flow travel time has passed. As in the homogeneous case, vehicles do not influence each other.

2. If a vehicle enters a link and hits the end of an already established queue, i.e. if all other vehicles on the link have exceeded their free-flow travel time, it passes no other vehicle but leaves the queue only after all other vehicles have left it.

3. In the general case where some vehicles have exceeded their free-flow travel time and some have not, a faster vehicle will only pass those vehicles which are still in their free-flow phase.

However, even this seemingly simple modification of allowing vehicle-specific free speeds requires the following considerations:

1. The approach would under-estimate the length of the queue. Firstly, in reality, vehicles whose free-flow exit time has expired may still already be part of the queue. As more vehicles become part of the queue, it grows in physical length, so every following vehicle has less space available for free-flow (and being passed). Secondly, one needs to make a decision regarding the queue density. From a traffic flow theoretical perspective, this density is determined by the fundamental diagram of the link, by looking up the congested density corresponding to the outflow of the link. Given that the outflow of the link may change in congested conditions with spillback from downstream links, it might be more parsimonious to just use the maximum density. This typically over-estimates the number of vehicles that are on a link under congested conditions, but it is consistent with the current approach.

2. Accurately computing the position of the end of the queue may use up the computational advantage of the current model. Even with the simple approach, the current simple first-in, first-out data structure needs to be changed into a priority queue. The impact on computational performance will need to be tested.

It remains to be shown to which degree this model can be calibrated to reproduce travel times of a real-world scenario with lane-changing and passing.

8.4. Simulating sub-populations with different planning horizons

Simulating long-haul freight traffic alongside a population of private car users raises the problem of different planning horizons. Personal traffic demand is modeled as a collection of plans for a day, and the agent which re-plans its schedule considers its options regarding daily activities. Trips which cannot be understood as being part of a daily routine, such as long-distance travel to a vacation spot, often do not appear in an activity-based demand model. In any case, one day is the time frame considered in replanning, and it is also the time period which is simulated in each iteration of the mobility simulation. On the other hand, freight traffic contains a substantial share of long-haul traffic with tours spanning several days, or shipments taking several days. There are two ways of solving this problem:

1. Leaving the simulated time period at one day. This requires that multi-day trips are broken down into segments starting at 00:00 and ending at 23:59.

2. Switching the simulated time period to something longer. Passenger plans are either repeated every 24 hours, or also switched to something longer.

If the purpose of the simulation is modeling decisions of freight operators, which includes weighing long tours versus short tours, the second option is desirable. As before, it is advantageous to have some periodicity in the simulated time period, since this reduces the problems with the boundaries. The next longer period which comes to mind is the week. With respect to freight, a week is plausible since many freight companies attempt to have their drivers at home over the weekend, even if this is just because of weekend restrictions for freight vehicles. In order to get from a population with daily plans to a population of weekly plans, a first step could be to unwind the plans, replicating them to fill a week. This is straight-forward except for the question of what to do in early iterations of the mobility simulation, where many vehicles may be severely delayed up to the point where they do not get home by the end of the day. Two options come to mind:

1. Treat the weekly plan the same way daily plans are treated. This means that delays are accumulated, and in the extreme case, an agent could be caught in congestion for several days.

2. Reset all agents to their home locations at a time where almost everyone plans to be at home (such as 03:00) and have them start their day afresh.

The re-planning step also requires some consideration for weekly plans. The initial demand is generated from data for one typical workday. This leads to the idea that re-planning should be continually performed on a daily plan which is then unwound in each iteration, so the population changes their daily routine for the whole week at once. If passenger data specifically for Saturdays and Sundays were available, three classes of days could be considered and re-planned separately: workdays, Saturdays and Sundays. In each iteration, each of the three day templates would be re-planned and a new weekly plan would be built from five copies of the workday plan and one copy of the Saturday and Sunday plan. However, if the unwound weekly plan is re-planned as a whole, most probably every workday in a plan will end up differently. The variability in the resulting plans may only represent uncertainty, but it could also suggest that agents have reasons for following different routines on different days of the week, for example by splitting their weekly work-hours unevenly over the days to avoid peak-hour congestion. Another option worth exploring and which is expected to speed up the relaxation process is to warm-start the passengers by using an already relaxed set of daily plans which was simulated without the freight share, in a simulation of one day. These plans would then be unwound to a week, the freight share would be added, and the weekly simulation with freight traffic would be started from there.

8.5. Conclusion

We discussed some aspects of implementing and adding a freight model to a multi-agent transport simulation which was designed with passenger traffic in mind. Some of them might be of independent interest in other applications: Moving from single autonomous agents to groups of agents which replan as a group with shared resources and obligations is a necessary step to implement shared usage of available cars by families or ride-sharing pools, a feature currently investigated by Dubernet, 2016; Dubernet and Axhausen, 2013. Sub-populations with varying planning timeframes can occur whenever the initial demand is generated from different data-sets, for example where weekend travel diaries are available or not available. A traffic flow simulation with heterogeneous vehicle fleets is helpful for scenarios where a large share of road traffic comes from non-car modes, as is common for scenarios in developing countries, but to what degree the interactions between different modes could be captured requires further investigation. The carrier agent class described in the first part of this chapter was implemented as the bottom layer of a multi-tier freight market model with the goal of simulating logistics decisions at different levels.

As stated, it was possible to add the freight extension entirely based on extension points, in this case the pre-existing events infrastructure, a pre-existing infrastructure to add code before and after iterations, and a newly added infrastructure to add additional vehicles into the mobility simulation. This motivates the now following Sec. 9, which presents a more general solution for a transparent and versatile simulation software.

9. Transparent and versatile simulation software

A previous version of this chapter was published as Zilske and Nagel (2016).

9.1. Introduction

Since its inception, the MATSim project[1] (Horni et al., 2016) has aimed at being a transparent and versatile research tool. Versatility means that the tool can be used for novel research questions which are not on the mind of its inventors. Transparency means that researchers who publish results obtained with the help of the tool can and will phrase the steps which they undertook in a way which makes them easy to reproduce (Gandrud, 2015), as in "We used revision r of the software with settings S on input I, and obtained the following results." Anyone with access to the data and software will then be able to re-run the original experiment, and anyone with sufficient knowledge of the software and the domain will be able to re-evaluate the original findings.

Translated to software engineering terms, the demand for versatility requires the software to be extensible. There are certainly research problems which are novel, yet can be phrased as reductions to MATSim runs, without the need to modify the tool or indeed write any code beyond generating input and transforming output. Many applications of MATSim, however, are not of this type, but require extending the functionality of the software.

Any software product can be extended in functionality as long as the source code is available. Also, when the modified source code and the input data are in a public repository, under version control, any research citing a revision number is, in principle, reproducible. However, the further the custom version diverges from the main line, the less likely it is that another community member or a research advisor will be able to confidently interpret the results using previous knowledge of the software, because it is not clear which parts of the behavior of the software system have changed under global modifications.

Extending software in a transparent way means, rather than modifying code in place, to use interfaces which the software provides as hooks for extensions. In that case, the extended system consists of a previously published revision of the original software, together with custom code implementing the extension interfaces, as well as custom code wiring custom and standard components together. A full plug-in system removes the need for the last step: Extensions are only declared, not imperatively created and connected.

[1] also see http://matsim.org

Their instantiation is managed by the framework, which can now keep a model of which extensions are active in a particular configuration, since that concern is removed from user code.

In this chapter, we discuss the architecture of MATSim and its plug-in system as it relates to the goals of versatility and transparency. Section 9.2 briefly introduces MATSim just as much as is needed to explain what it means to extend it. In Section 9.3, its *extension points* are identified, which are interfaces against which a user can implement new functionality. Section 9.4 describes the implementation of the extension mechanism itself, using the Guice software framework for dependency injection. In section 9.5, we conclude with some remarks about possible future developments on the software side of the MATSim project.

9.2. Controller

MATSim is an iterative simulation. A simulation run begins with constructing an initial person-based travel demand model, which is done in user code prior to calling the simulation itself. The simulation progresses in a loop, which relaxes the demand with respect to a utility function. When a termination criterion is satisfied, the loop terminates, and control is returned to the client, where analysis code can be run.

A class called `Controller`[2] implements the simulation loop as depicted in Figure 9.1, starting after the initial demand generation and ending before analysis:

`Preparation` initializes the agent population read from an input file to a state where it can be executed by the physical simulation. This includes registering activity locations with network links and finding initial routes.

`PhysicalSimulation`[3] executes the travel demand of the agent population on the capacity-constrained transport network. It produces a timeline of the simulation outcome, represented by a stream of `Events`.

`Scoring` observes the `Events` stream, calculates for each agent a score for its outcome, and stores it in the agent memory.

`Replanning` lets each agent mutate its planned travel behavior.

Each iteration consists of three pre-defined process steps, `PhysicalSimulation`, `Scoring`, and `Replanning`, while `Preparation` is a pre-processing step. The simulation loop with its process steps has been in place since the beginning of the MATSim project, first by sequentially calling stand-alone scripts, later by calling the process steps from an early version of `Controller` (Nagel and Axhausen, 2016).

[2] Wrongly spelled `Controler` in MATSim.
[3] Called `Mobsim` (= mobility simulation) in MATSim.

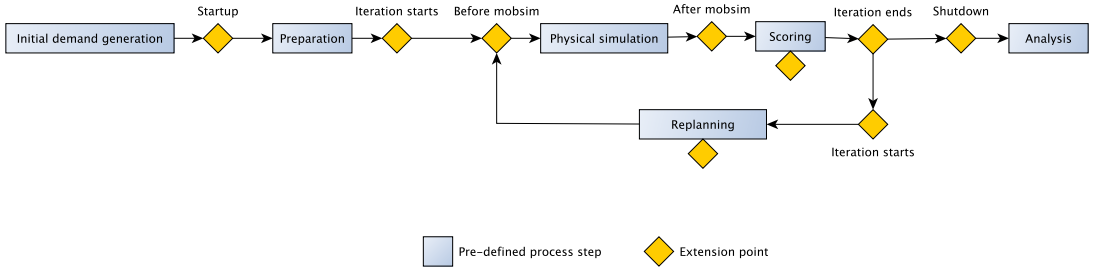


Figure 9.1.: The extensible MATSim simulation loop. (Adapted from Horni et al., 2016.)

9.3. Extension Points

The implementations of these process steps can be switched by the user. The standard implementations shipped with MATSim are almost completely independent components, sharing state only through two simple mutable data containers, `Population` and `Network`, and the `Events` stream. This makes them easily replaceable. Historically, the way to achieve this was to subclass the `Controler` instance and override the methods that were calling these process steps.

Often, rather than replacing the entire `PhysicalSimulation` or `Replanning` stage, a user wants to extend or modify the behavior of their standard implementations. The general way of doing this was to subclass or copy the standard implementation class. Additionally, the user would subclass `Controller` and replace the *construction* of the standard implementation instance, in order to decorate it with the new behavior. Only the `Replanning` implementation was designed as extensible from the beginning, and user-defined behavior could be added by referencing a class name from a configuration file.

Finally, users wanted to add code into the control loop, for example preparing the `Population` in a specific way at the beginning of each iteration, or adding iteration-specific analysis at the end of each iteration. Again, this was often achieved by subclassing `Controller`, overriding the corresponding methods, and inserting the additional code as desired.

While this practice worked when each researcher was working on extending MATSim by a different aspect for one-off experiments, it fails when it comes to integrating those aspects into a single system.

In order to address the issue of inserting additional code into the control loop, a `ControllerListener` infrastructure was added (Grether and Nagel, 2013): At several points in the control flow, additional process steps are called which can be registered by the user as implementations of designated interfaces extending `ControllerListener`. The `ControllerListener` interface is a simple execution hook, similar to the Java `Runnable`. An instance of `ControllerListener` can expect to be called at the point in the control flow for which it registered,

Several implementations of a single listener type can be provided, and they are called in an undefined order. For this reason, one implementation of a listener type cannot

assume a computation specified in another implementation of the same listener type to have been carried out. For `ControllerListener` implementations which register for the points at the `Scoring` or the `Replanning` step (cf. Figure 9.1), the contract is that these listeners are conceptually executed concurrently to the corresponding process step.

It also became possible to replace some of the implementations of the pre-defined process steps by changing factory properties on the `Controller` instance. Additionally, it became possible to replace some of the objects which are used by specific process steps, such as the routing algorithm, which is used during `Replanning`, or the current link traversal time estimate and generalized cost function, `TravelTime` and `TravelDisutility`, which are in turn used by the routing algorithm.

The rest of this section describes the standard process steps from Section 9.2 in sufficient detail to motivate their extension points. In Section 9.4, we describe the implementation of a component or plug-in system, where the user does not write and maintain component construction code, but describes components, and declares the extension points they implement, in recombinable modules.

9.3.1. Physical simulation and events

The `PhysicalSimulation` concurrently simulates travel plans in a capacity-constrained traffic system. Given a fixed traffic infrastructure, it is a function of a set of plans to a set of outcomes. As a software component, its dependencies are the scenario data containers, `Population` and `Network`, and it produces a stream of `Events`, describing the outcome. A custom implementation of a `PhysicalSimulation` need not be written in the Java programming language: The framework includes a helper class to call an arbitrary executable which is then expected to write its event stream into a file (Balmer et al., 2009; Strippgen, 2009).

The traffic flow simulation moves the agents around in the virtual world according to their plans and within the bounds of the simulated reality. It documents their moves by producing a stream of `Events` (Rieser et al., 2016). Examples of such events are:

- An agent finishes an activity
- An agent starts a trip
- A vehicle enters a road segment
- A vehicle leaves a road segment
- An agent boards a public transport vehicle
- An agent arrives at a location
- An agent starts an activity

Each event has a timestamp, a type, and additional attributes required to describe the action like a vehicle identifier, a link identifier, an activity type or other data. In theory, it should be possible to replay the traffic flow simulation just by the information

stored in the events. While a plan describes an agent's intentions, the stream of events describes how the simulated day actually was.

As the events are so basic, the number of events generated by a traffic flow simulation can easily reach a million or more, with large simulations even generating more than a billion events. But as the events describe all the details from the execution of the plans, it is possible to extract mostly any kind of aggregated data one is interested in. Practically all analyses of MATSim simulations make use of events to calculate some data. Examples of such analyses are the average duration of an activity, average trip duration or distance, mode shares per time window, number of passengers in specific transit lines and many more.

The scoring of the executed plans makes use of events to find out how much time agents spent at activities or for traveling. MATSim extensions can observe the traffic flow simulation by interpreting the stream of `Events`.

9.3.2. Scoring

The parameters of the standard MATSim scoring function are configurable. The code which maps a stream of traffic flow simulation `Event`s to a score for each agent is placed behind a `ScoringFunctionFactory` interface and replaceable. Within the factory method, users can build a custom utility formulation which can be different for each synthetic person. There are building blocks for common terms like the utility of performing an activity, or the disutility of travelling. Custom terms can be added. For instance, a module which simulates weather conditions might calculate penalties for pedestrians walking in heavy rain. A modeler who wishes to compose a scoring function from the standard MATSim utility and the rain penalty can re-use the former and add the latter.

9.3.3. Replanning

Replanning in MATSim is specified by defining a weighted set of strategies. In each iteration, each agent draws a strategy from this set and executes it. The strategy specifies how the agent changes its behavior. Most generally, it is an operation on the plan memory of an agent: It adds and/or removes plans, and it marks one of these plans as selected.

Strategies are implementations of the `PlanStrategy` interface. The two most common cases are:

- Pick one plan from memory according to a specified choice algorithm, and mark it as selected.
- Pick one plan from memory at random, copy it, mutate it in some specific aspect, add the mutated plan to the plan memory, and mark this new plan as selected.

The framework provides a helper class which can be used to implement both of these strategy templates. The helper class delegates to an implementation of `PlanSelector` , which selects a plan from memory, and to zero, one or more implementations of `PlanStrategyModule`, which mutate a copy of the selected plan.

The most commonly used strategies shipped with MATSim are:

- Select from the existing plans at random, weighted by their current score.
- Mutate a random existing plan by re-routing all trips.
- Mutate a random existing plan by randomly shifting all activity end times backwards or forwards.
- Mutate a random existing plan by changing the mode of transport of one or more trips and then re-routing them with the new mode.

Routes are computed based on the previous iteration's traffic conditions, measured by analyzing the `Events` stream. Using the same pattern, a custom `PlanStrategy` can use any data which can be computed from the physical simulation outcome.

The maximum size of the plan memory per agent is a configurable parameter of MATSim. Independent of what the selected `PlanStrategy` does, the framework will remove plans in excess of the maximum from the plan memory. The algorithm by which this is done is another implementation of `PlanSelector` and can be configured.

9.3.4. Travel time, disutility and routing

Re-routing as a building block of many replanning strategies is a complex operation by itself. It can even be recursive: For example, finding a public transport route may consist of selecting access and egress stations as sub-destinations, finding a scheduled connection between them, and finding pedestrian routes between the activity locations and the stations. With the `TripRouter` service, the framework includes high-level support for assembling complex modes of transport from building blocks provided by other modules or the core.

The `TripRouter` provides methods to generate trips between locations, given a mode, a departure time, and an identifier of the travelling person (Dubernet, 2013). It is used in the replanning stage of the simulation loop to modify the behavior of the simulated agents by replanning their trips. A trip is a sequence of plan elements representing a movement. It typically consists of a single leg, or of a sequence of legs with *stage activities* in between. For instance, public transport trips contain stage activities which represent changes of vehicles in public transport trips.

The behavior of the `TripRouter` is assembled from `RoutingModule` instances, one of which is associated with each mode. A `RoutingModule` defines the way a trip is computed, and declares the stage activities it generates.

The association between main modes and `RoutingModule` instances is configurable, and a user can provide custom `RoutingModule` implementations. This is required for use-cases which require custom routing logic, for instance if a new complex mode of transport is to be evaluated.

The standard behavior is that there are two `RoutingModule` implementations: For *network routing*, and for *teleportation*. By network mode, we refer to a mode whose trips are represented by paths through the network. By teleportation, we mean that travel times are determined based on a non-network property of the origin-destination relation,

such as the bee-line distance. Even though no network route is calculated, this concern is implemented by a `RoutingModule`, because it determines the characteristics of a trip, given origin, destination and mode.

The network router, but not the teleportation router, makes use of two further interfaces: First, `TravelDisutility`, which answers queries about the time-dependent traversal cost of each edge. And second, `TravelTime`, which returns the time-dependent traversal time for each edge. This is used by the least-cost path algorithm to advance the time as paths are expanded.

The standard implementation of `TravelTime` answers queries from an in-memory map from $(link, time)$ to $duration$, which is initialized with free-flow traversal times as determined by the road category. After each iteration, it is updated with measured travel times, which are obtained by observing the `Events` stream.

The standard implementation of `TravelDisutility` evaluates a linear function of the link length and the travel time, the coefficients of which are configuration settings.

9.4. Dependency Injection

We now have identified several interfaces as extension points to modify the standard behavior of the simulation. In Figure 9.2, we show them in a class diagram in the context of the components whose behavior they modify. Notice their different multiplicities. There is a single `ScoringFunctionFactory`, since it can be written to create a different function object for each agent, in case heterogeneous scoring over agents is desired. Because of the way the replanning works, there is a single `PlanSelector` for plans removal, but there can be an arbitrary number of weighted `PlanStrategy` instances. There is one `RoutingModule` for each defined mode of transport, and there is one `TravelTime` and one `TravelDisutility` for each mode which is routed on the network.

The problem is now to provide a facility by which a user can create a `Controller` instance whose functionality is extended by custom implementations of one or more of these interfaces, with the additional requirement that such extensions be combineable.

At first glance, this could be done by making the `Controller` a JavaBean-like class with settable properties. Taking into account the different multiplicities, and the fact that some of the extension points are indexed by mode of transport, this could be usable like this:

```
Controller controller = new Controller();
controller.setScoringFunctionFactory(new RainScoringFunctionFactory());
controller.addTravelTime("bike", new BikeTravelTime());
```

Modularity could be realized by passing the `Controller` instance to several methods, where each method would correspond to a module, and would modify the properties of the `Controller` to register the extensions this module provides.

The main problem with this approach is this: The no-argument constructors of the user-provided classes in the above example are an idealization. Typically, these classes will have dependencies of their own, and some of these will be services provided by MATSim. For instance, a user-defined `ScoringFunctionFactory` may want to use the `TripRouter`

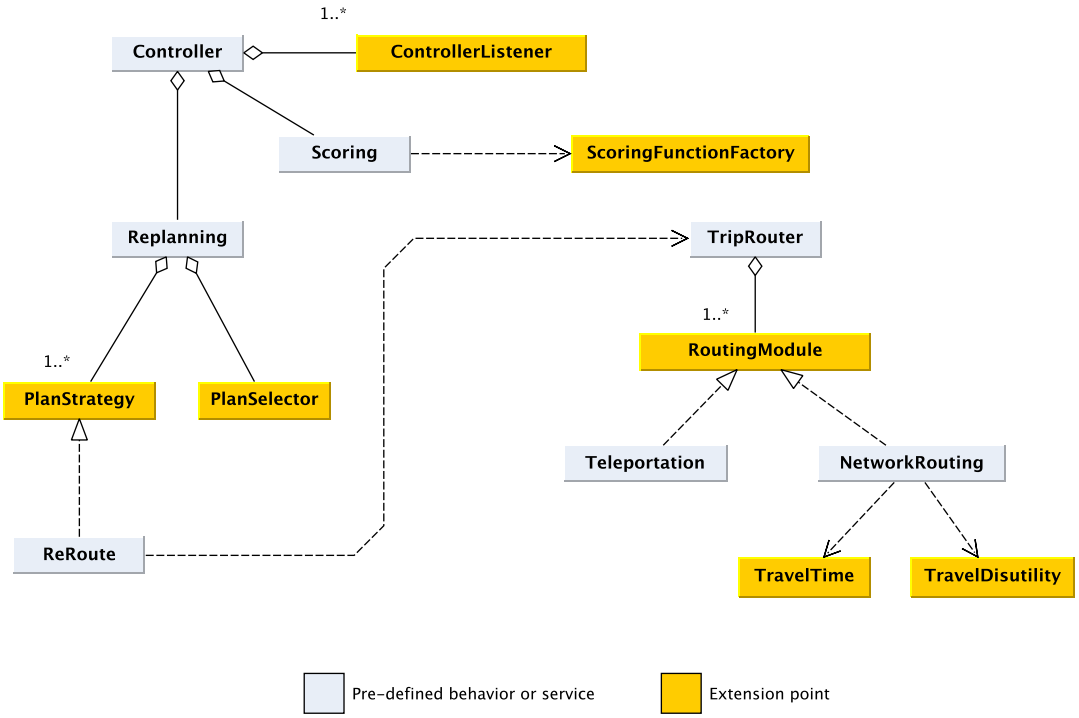


Figure 9.2.: Extension points in the context of the components whose behavior they modify, as a Unified Modeling Language (UML) class diagram. Solid links with a diamond-shaped base mean *is composed of*. Dashed links with a triangle-shaped base mean *is implemented by*. Dashed links with an arrowhead mean *uses*.

to judge a chosen alternative relative to others. To make this possible, `TripRouter` would have to be a gettable property on `Controller`. However, a `TripRouter` internally consists of `RoutingModule` instances, which in turn are settable properties on the same level. Hence, the `TripRouter` is only constructable and gettable as soon as all desired `RoutingModule` instances are set, and there is no easy way to enforce this, except asking the user to sort their usage of getters and setters and checking internally that no setter of A is ever called after something which depends on A has been constructed.

9.4.1. Manual Dependency Injection

The fully-general, elementary solution to this problem is to require the entire object graph to be constructed bottom up, creating the most basic objects and services first and passing them to higher-level objects by their constructors. This forces the user, on a programming language level, to create the objects in the correct order, producing code like the following:

```
BikeTravelTime bikeTravelTime = new BikeTravelTime();
TripRouter tripRouter = new TripRouter(bikeTravelTime);
RainScoringFunctionFactory rainScoringFunctionFactory =
  new RainScoringFunctionFactory(tripRouter);
Controller controller =
  new Controller(rainScoringFunctionFactory, tripRouter);
```

With this, the property of modularity is lost. There is no general way of factoring parts of code like the above into independent methods with a common signature. Every such script by every user would have to contain the constructor calls for every required object. Moreover, this approach breaches encapsulation: While `TripRouter` is meant as a service interface, to be consumed by user code rather than implemented, its instantiation has now been moved to user code, exposing the implementation class.

9.4.2. Framework-assisted Dependency Injection

An established solution to these problems in enterprise software is framework-assisted dependency injection (Fowler, 2004), where dependencies within a set of managed objects are resolved by a software framework. This enables a user-side simplicity and modularity similar to the inital example, even when the user-provided classes have additional dependencies, while achieving the safety and generality of the explicit approach where all dependencies are passed by constructor.

Guice (*Guice*) is a dependency injection framework developed by Google.[4] Its core concepts are *binding* an interface type to an implementation, and *injecting* managed instances of bound types, sometimes called components, with other components. Bindings are defined within *modules*, which are Java classes. An advantage to earlier frameworks such as Spring (*The Spring Framework – Reference Documentation*), which defined its bindings in XML files, is that module definitions are automatically included in refactoring tools provided by popular IDEs, and no additional tooling is needed. Also, since bindings are Java statements, it is easy to transform the general language provided by Guice to declare bindings into a more domain-specific language, specifically for binding extensions to our designated extension points.

An example for a module which extends two extension points by adding two bindings, one for a custom `ScoringFunctionFactory`, and one for a custom `TravelTime`, is the following:

```
bindScoringFunctionFactory().to(RainScoringFunctionFactory.class);
addTravelTimeBinding("bike").to(BikeTravelTime.class);
```

Note that these two statements do not share any variables, so they can be decomposed into two modules. In particular, bindings do not have to occur in any particular order. In contrast to the example in Section 9.4.1, the `TripRouter` instance does not already have to be constructed when `RainScoringFunctionFactory` is bound.

Once an interface type is bound, it can be injected into instances which are managed by the framework, in particular of classes on the right-hand side of bind statements. Since such instances are created by the framework, their classes must have an injectable

[4]The current release version, 4.0, was used for this implementation.

constructor, which is one taking either no arguments or only arguments of other bound types. When the framework creates an instance, its dependencies have already been created and can be passed to the constructor (injected).

For example, the main implementation class of the `Scoring` stage includes an annotated field definition by which it declares a dependency to the possibly user-defined `ScoringFunctionFactory` component, which it then uses to create `ScoringFunction` instances for simulated entities:

```
class ScoringFunctionsForPopulation implements PlansScoring {
  final ScoringFunctionFactory sff;
  @Inject
  ScoringFunctionsForPopulation(ScoringFunctionFactory sff) {
    this.sff = sff;
  }
  public void scorePlans() {
    for (Person person : persons) {
      ScoringFunction sf = sff.createScoringFunction(person);
      // ...
    }
  }
}
```

At startup time, the framework detects the dependency by inspecting the constructor annotated with `@Inject`. It finds the corresponding binding and passes the appropriate instance when the constructor is called. As with regular initializations made in constructors, the instance method `scorePlans()` can expect a fully initialized instance.

Since modules are Java code, binding statements can be made conditional upon configuration parameters. In MATSim, a user-extensible data structure with system-wide configuration parameters is read from a file and made available at startup time. At that point, a set of modules shipped with MATSim, together with user-provided modules, is evaluated by the framework to construct the object graph, which is afterwards immutable for the simulation run.

A slightly simplified version of the object graph of the standard configuration of MATSim is shown in Figure 9.3. Every dashed line corresponds to a binding. The dashed boxes are bound interfaces appearing on the left-hand side of bind statements, and the solid boxes are implementation types appearing on the right-hand side of bind statements. Each injection produces a solid edge in the graph.

For cases where it is not possible to leave the creation of instances to the framework, for example when an instance needs to be built programmatically in multiple steps, a similar construct can be used to specify a custom `Provider` (denoted by double arrows in Figure 9.3), a functional interface which is called by the framework and is expected to return an instance. Orthogonally, another construct exists for binding a single concrete instance of the type or the `Provider`, when it is already available at configuration time (denoted by gray background in Figure 9.3).

Since the creation of components is managed by the framework, their dependency graph is inspectable at run-time. In contrast to the hand-drawn class diagram of Figure 9.2, the object diagram can be plotted and saved to the output directory of a simulation

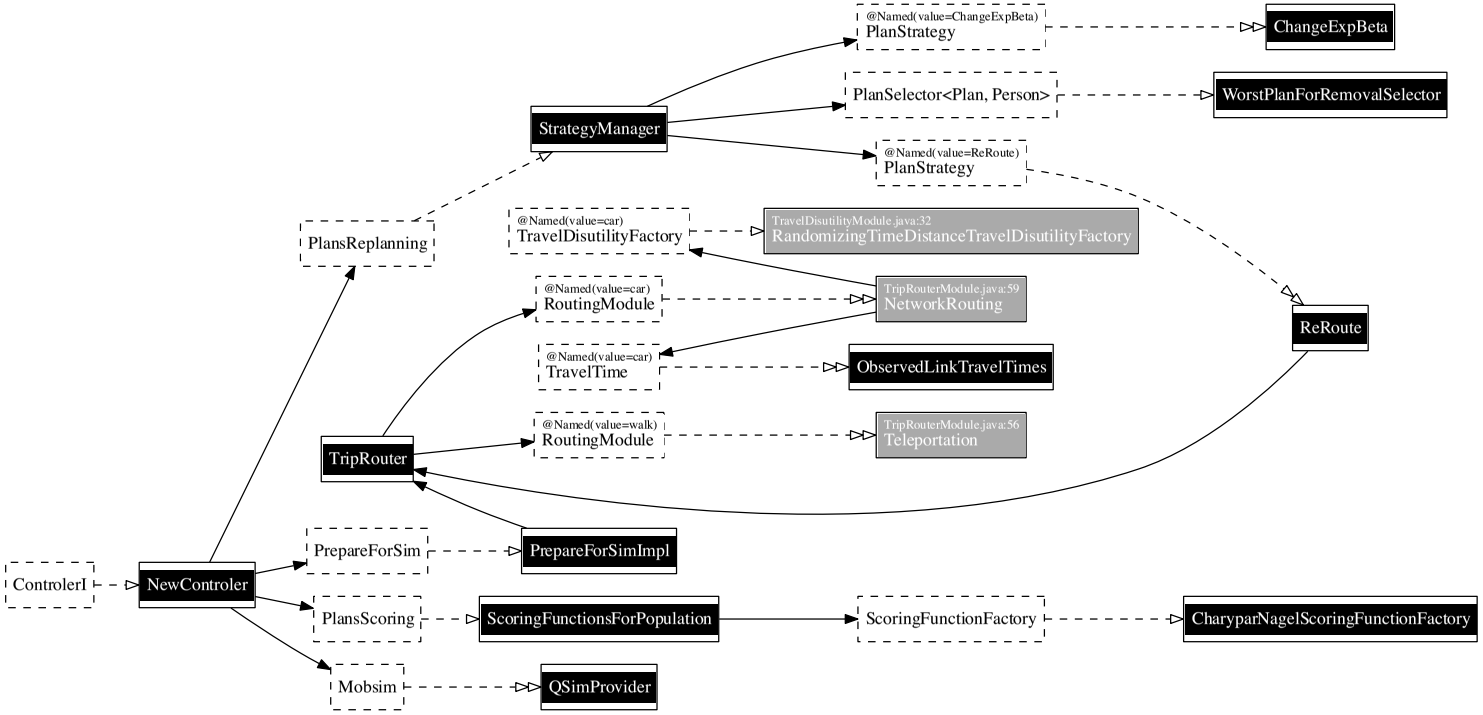


Figure 9.3.: The run-time object graph for a typical MATSim instance. The included classes implement the simulation logic. The graph is created at startup time, determined by a set of system and user-defined modules and their configuration. It is immutable for the duration of a simulation run. Since it is framework-managed, the graph can be plotted and saved automatically with the simulation output. This is the graph resulting from the MATSim standard configuration. Experimental extension points not mentioned in the text, as well as trivial or technical components such as proxies, were manually excluded.

run. It conserves and visualizes the structure as determined by the set of modules and the configuration. The (slightly simplified) example of Figure 9.3 shows a standard configuration of MATSim, but the graph will track configuration changes and custom modules introduced by users of the framework.

9.5. Discussion and Future Work

We believe that being able to automatically generate visualizations of the configuration of a simulation run like in Figure 9.3 is an important step for transparency. Before, user code plugged into MATSim would typically contain statements such as

```
controller.setXFactory(xFactory);
```

Since such a statement could be inserted multiple times, time-consuming code inspection was necessary to find out in which sequence they were called, and if the factory may already have been used before being replaced by yet another one. The approach described in Section 9.4 completely solves this, by (1) prohibiting that a component managed by the dependency injection framework can be switched after it has been used for the first time, and (2) by automatically logging the full dependency graph with each run.

Framework-assisted dependency injection is a pattern more typically used in enterprise rather than research software. We are tracking other developments in the Java world which may be put to use for system-level simulation development. In particular, the process step summarized under physical simulation is a complex, extensible piece of software by itself. A close-up view would identify another set of extension points on that layer. This is also where interaction between agents takes place. MATSim does not use any framework for agent-based microsimulations, but implements all its aspects, including concurrency, directly in Java.

The stream of microsimulation `Events` described in this chapter is implemented as a simple event publisher, where interested parties can subscribe and register a call-back function, which is called every time an event of a specific type occurs. When seen as a virtue, this pattern is sometimes called reactive programming, and its main use cases are GUI applications and real-time stream processing, but also event-driven parts of simulations: It is a natural programming style for describing how the state of one object, such as the cell of a spreadsheet or the current estimate of the average speed on a link, is supposed to change over time in reaction to other observable processes, such as other cells of the spreadsheet or vehicles entering and leaving the link. To support this style, there are frameworks, such as Project Reactor (*Reactor*), which provide syntax similar to the Java 8 Stream interface, but for events emitted by sequential, pushing producers. In this framework, event streams are first-class objects which can be transformed and combined with functional operators such as filter, map and fold. For instance, if one only wanted to react to every fifth event, one would traditionally implement a state machine in the event handler, which counts to five and only reacts every fifth time. With framework-assisted reactive programming, one would transform the original event stream into one which only emits every fifth event, and subscribe to it with the same syntax as used for the original one. This sometimes leads to more concise code with less explicit state.

However, sometimes the call-back oriented or reactive programming style is chosen not because it is the most expressive style for the problem, but out of technical neccessity. Consider the behavior-driven part of simulations: For agents, the inversion of control flow is unnatural, and they will always have to implement state machines based on the values of their properties. Rather than writing:

```
int state = 0;
public void determineNextAction() {
    if (state % 3 == 0) travelTo("A");
    else if (state % 3 == 1) travelTo("B");
    else if (state % 3 == 2) travelTo("C");
    ++state;
}
```

one would prefer to write

```
public void act() {
  while (true) {
    travelTo("A");
    travelTo("B");
    travelTo("C");
  }
}
```

Note that in the first example, `travelTo` would return immediately, and calling a second `travelTo` action in the same function call would be possible though meaningless. In the second example, behavior is coded like a script for the simulated entity, and the state is captured in the program flow.

However, in the Java programming language, this programming style would only be possible if every simulated entity would run in its own thread, which generally is not an option because threads have a high performance overhead, which is why simulations in Java generally use the reactive style for agents.

The interest in light-weight alternatives to threads has most recently been re-popularized by their presence in the Go programming language, but in contrast to other and much older programming languages such as Scheme, Java does not have first-class continuations, which are the missing bit to implement a non-reactive agent framework natively in Java. The Continuations Library (*Continuations Library*) implements them using bytecode instrumentation, and Project Quasar (*Quasar*) uses them to implement fibers, which are similar to threads but without the performance overhead, as a Java virtual machine extension. Neither requires extending the language itself.

9.6. Conclusion

We identified and described a set of interfaces to customize and extend a traffic simulation software package. As a mechanism for the user to implement custom behavior using these interfaces, previous versions of the software relied on a system of settable factory properties on multiple levels, which had grown over time as the need for more such interfaces developed. Its main drawback was that the responsibility for creating objects

in the correct order was left to the users: They would have to find a point in the control flow late enough so that all dependencies necessary to create an instance of a custom class would be available, but early enough so that the instance itself would not yet have been requested. Alternatively, the creation of the entire object graph would have to be done in user code, which would introduce duplicate code over different configurations, and expose all dependencies of all components, requiring all configuration code to change whenever a constructor signature changed.

For this chapter, that mechanism was developed into a plug-in system. It was implemented using dependency injection provided by the Guice framework. This solves the problem of mutable factory properties, unspecified order of object creation, and public visibility of dependencies. All components are constructed at startup time, in an order induced by the dependency graph, and with a check on completeness and uniqueness. The construction process and the resulting component graph are inspectable. This can be used to draw object graphs visualizing the system configuration, which can be saved with the simulation output, so that the precise configuration used to produce a simulation run is transparent. The described mechanism has already been used externally to develop a car-sharing model based on MATSim (Laarabi and Bruno, 2016).

Bibliography

VSP Working Paper. See http://www.vsp.tu-berlin.de/publications. TU Berlin, Transport Systems Planning and Transport Telematics.

Annual Meeting Preprint. Washington, D.C.: Transportation Research Board.

Agarwal, A., M. Zilske, K.R. Rao, and K. Nagel (2015). "An elegant and computationally efficient approach for heterogeneous traffic modelling using agent based simulation". In: *Procedia Computer Science* 52.C, pp. 962–967. ISSN: 1877-0509. DOI: 10.1016/j.procs.2015.05.173.

Agarwal, Pankaj K, Boris Aronov, Marc van Kreveld, Maarten Löffler, and Rodrigo I Silveira (2010). "Computing similarity between piecewise-linear functions". In: *Proceedings of the twenty-sixth annual symposium on Computational geometry*. ACM, pp. 375–383.

Auld, Joshua, Chad Williams, Abolfazl Kouros Mohammadian, and Peter Nelson (2008). "An automated GPS-based prompted recall survey with learning algorithms". In: *Transportation Letters: The International Journal of Transportation Research* 1.1, pp. 59–79.

Baier, R. et al. (2009). *Handbuch für die Bemessung von Straßenverkehrsanlagen*. Ed. by W. Brilon, L. Dunker, G. Hartkopf, G. Kellermann, K. Lemke, G. Reichardt, and W. Schnabel. Köln: FGSV Verlag GmbH.

Balmer, M., M. Rieser, K. Meister, D. Charypar, N. Lefebvre, K. Nagel, and K.W. Axhausen (2009). "MATSim-T: Architecture and Simulation Times". In: *Multi-Agent Systems for Traffic and Transportation*. Ed. by A.L.C. Bazzan and F. Klügl. IGI Global, pp. 57–78.

Bar-Gera, Hillel, Karthik Charan Konduri, Bhargava Sana, Xin Ye, and Ram M Pendyala (2009). *Estimating Survey Weights with Multiple Constraints Using Entropy Optimization Methods*. Annual Meeting Preprint 09-1354. Washington D.C.: Transportation Research Board.

Battelle Memorial Institute (1997). *Global positioning systems for personal travel surveys: Lexington Area Travel Data Collection Test*. Tech. rep. URL: https://www.fhwa.dot.gov/ohim/lextrav.pdf.

Blondel, Vincent D., Markus Esch, Connie Chan, Fabrice Clerot, Pierre Deville, Etienne Huens, Frédéric Morlot, Zbigniew Smoreda, and Cezary Ziemlicki (2012). *Data for Development: the D4D Challenge on Mobile Phone Data*. eprint: arXiv:1210.0137.

Bricka, Stacey and Chandra R Bhat (2006). *A Comparative Analysis of GPS-Based and Travel Survey-based Data*. Annual Meeting Preprint 1972. Washington, D.C.: Transportation Research Board.

Calabrese, Francesco, Giusy Di Lorenzo, Liang Liu, and Carlo Ratti (2011). "Estimating origin-destination flows using mobile phone location data". In: *IEEE Pervasive Computing* 10.4, pp. 0036–44.

Cetin, N (2005). "Large-Scale parallel graph-based simulations". PhD thesis. Switzerland: Swiss Federal Institute of Technology (ETH) Zürich.

Chen, Cynthia, Ling Bian, and Jingtao Ma (2014). "From traces to trajectories: How well can we guess activity locations from mobile phone traces?" In: *Transportation Research Part C: Emerging Technologies* 46.0, pp. 326 –337. ISSN: 0968-090X. DOI: 10.1016/j.trc.2014.07.001.

Chung, Eui-Hwan and Amer Shalaby (2005). "A Trip Reconstruction Tool for GPS-based Personal Travel Surveys". In: *Transportation Planning and Technology* 28.5, pp. 381–401.

Date, Chris J and Hugh Darwen (1997). *A Guide To Sql Standard*. Vol. 3. Addison-Wesley Reading.

Doherty, Sean T, Dominik Papinski, and Martin Lee-Gosselin (2006). "An internet-based prompted recall diary with automated GPS activity-trip detection: System design". In: *Washington, DC, January*, pp. 22–26.

Dubernet, T. (2013). *The New MATSim Routing Infrastructure*. Presentation at the MATSim User Meeting, Zurich. URL: http://archiv.ivt.ethz.ch/vpl/publications/presentations/v452.pdf.

– (2016). "Joint Decisions". In: *The Multi-Agent Transport Simulation MATSim*. Ed. by A. Horni, K. Nagel, and K. W. Axhausen. Ubiquity, London. Chap. 28. DOI: 10.5334/baw.

Dubernet, Thibaut and Kay W Axhausen (2013). "Including joint decision mechanisms in a multiagent transport simulation". In: *Transportation Letters* 5.4, pp. 175–183.

Ferrari, Laura and Marco Mamei (2011). "Discovering daily routines from google latitude with topic models". In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*. IEEE, pp. 432–437.

Flötteröd, G. (2009). "Cadyts – A free calibration tool for dynamic traffic simulations". In: *Swiss Transport Research Conference*. http://www.strc.ch/conferences/2009/Floetteroed.pdf.

Flötteröd, G., M. Bierlaire, and K. Nagel (2011a). "Bayesian demand calibration for dynamic traffic simulations". In: *Transportation Science* 45.4, pp. 541–561. DOI: 10.1287/trsc.1100.0367.

Flötteröd, G., Y. Chen, and K. Nagel (2011b). "Behavioral Calibration and Analysis of a Large-Scale Travel Microsimulation". In: *Networks and Spatial Economics* 12.4, pp. 481–502. ISSN: 1566-113X. DOI: 10.1007/s11067-011-9164-9.

Flötteröd, Gunnar and Michel Bierlaire (2013). "Metropolis–Hastings sampling of paths". In: *Transportation Research Part B: Methodological* 48.1, pp. 53–66. ISSN: 0191-2615. DOI: 10.1016/j.trb.2012.11.002.

Flötteröd, Gunnar and Ronghui Liu (2014). "Disaggregate path flow estimation in an iterated dynamic traffic assignment microsimulation". In: *Journal of Intelligent Transportation Systems* 18.2, pp. 204–214. DOI: 10.1080/15472450.2013.806854.

Follmer, Robert, Dana Gruschwitz, Birgit Jesske, Sylvia Quandt, Barbara Lenz, Claudia Nobis, and Katja Köhler (2010). *Mobilität in Deutschland 2008 – Methodenbericht.* Tech. rep. URL: http://daten.clearingstelle-verkehr.de/223.

Fowler, M. (2004). *Inversion of Control Containers and the Dependency Injection pattern.* online article. URL: http://martinfowler.com/articles/injection.html.

Gandrud, Christopher (2015). *Reproducible Research with R and RStudio.* CRC Press, p. 294. ISBN: 1498715379.

Gawron, C. (1998). "An Iterative Algorithm to Determine the Dynamic User Equilibrium in a Traffic Simulation Model". In: *International Journal of Modern Physics C* 9.3, pp. 393–407.

Golovko, Darya (2016). *Public Transport Assistant Plugin for JOSM. Google Summer of Code 2016, Final Report.* Tech. rep. URL: https://summerofcode.withgoogle.com/archive/2016/projects/4703186757615616/.

Goodchild, Michael F. (2007). "Citizens as sensors: the world of volunteered geography". In: *GeoJournal* 69.4, pp. 211–221. ISSN: 1572-9893. DOI: 10.1007/s10708-007-9111-y.

Google, Inc. *Guice.* URL: https://github.com/google/guice (visited on 01/25/2016).

Grether, D. and K. Nagel (2013). "Extensible Software Design of a Multi-Agent Transport Simulation". In: *Procedia Computer Science* 19, pp. 380–388. ISSN: 1877-0509. DOI: 10.1016/j.procs.2013.06.052.

Grether, D. and T. Thunig (2016). "Traffic Signals and Lanes". In: *The Multi-Agent Transport Simulation MATSim.* Ed. by A. Horni, K. Nagel, and K. W. Axhausen. Ubiquity, London. Chap. 12. DOI: 10.5334/baw.

Grether, D., S. Fürbas, and K. Nagel (2013). "Agent-based Modelling and Simulation of Air Transport Technology". In: *Procedia Computer Science* 19, pp. 821–828. ISSN: 1877-0509. DOI: 10.1016/j.procs.2013.06.109.

Hägerstrand, T. (1970). "What about people in Regional Science?" In: *Papers in Regional Science* 24, pp. 6–21. DOI: 10.1007/BF01936872.

Haklay, Mordechai and Patrick Weber (2008). "OpenStreetMap: User-generated street maps". In: *IEEE Pervasive Computing* 7.4, pp. 12–18. DOI: 10.1109/MPRV.2008.80.

Harremoës, Peter (2001). "Binomial and Poisson distributions as maximum entropy distributions". In: *IEEE Transactions on Information Theory* 47.5, pp. 2039–2041.

Horni, A., K. Nagel, and K. W. Axhausen, eds. (2016). *The Multi-Agent Transport Simulation MATSim.* Ubiquity, London. DOI: 10.5334/baw.

Horni, Andreas, David Charypar, and Kay W. Axhausen (2011). "Variability in Transport Microsimulations Investigated for MATSim: Preliminary Results". In: *Proceedings of the 11th Swiss Transport Research Conference (STRC).* URL: http://www.strc.ch.

Iqbal, Shahadat, Charisma Choudhury, Pu Wang, and Marta C Gonzalez (2014). "Development of origin-destination matrices using mobile phone call data". In: *Transportation Research Part C* 40, pp. 63–74.

Isaacman, Sibren, Richard Becker, Ramon Caceres, Margaret Martonosi, James Rowland, Alexander Varshavsky, and Walter Willinger (2012). "Human mobility modeling at metropolitan scales". In: *MobiSys '12: Proceedings of the 10th international conference*

on Mobile systems, applications, and services. Low Wood Bay, Lake District, UK, pp. 239–252.

Johnson, Rod, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Alef Arendsen, Darren Davison, Dmitriy Kopylenko, Mark Pollack, et al. *The Spring Framework – Reference Documentation*. URL: http://docs.spring.io.

Kagerbauer, Martin, Nicolai Mallig, Peter Vortisch, and Manfred Pfeiffer (2015). "Modellierung von Variabilität und Stabilität des Verkehrsverhaltens im Längsschnitt mithilfe der Multi-Agenten-Simulation mobiTopp". In: *Straßenverkehrstechnik* 59.6.

Kickhöfer, B. (2014). "Economic Policy Appraisal and Heterogeneous Users". PhD thesis. Berlin: Technische Universität Berlin. DOI: 10.14279/depositonce-4089.

Kickhöfer, B., F. Hülsmann, R. Gerike, and K. Nagel (2013). "Rising car user costs: comparing aggregated and geo-spatial impacts on travel demand and air pollutant emissions". In: *Smart Transport Networks: Decision Making, Sustainability and Market structure*. Ed. by T. Vanoutrive and A. Verhetsel. NECTAR Series on Transportation and Communications Networks Research. Edward Elgar Publishing Ltd, pp. 180–207. ISBN: 978-1-78254-832-4. DOI: 10.4337/9781782548331.00014.

Kickhöfer, B., D. Hosse, K. Turner, and A. Tirachini (2016). *Creating an open MATSim scenario from open data: The case of Santiago de Chile*. VSP Working Paper 16-02. See http://www.vsp.tu-berlin.de/publications. TU Berlin, Transport Systems Planning and Transport Telematics.

Kochan, Bruno, Tom Bellemans, Davy Janssens, Geert Wets, and H J P Timmermans (2010). "Quality assessment of location data obtained by the GPS-enabled PARROTS survey tool". In: *Journal of Location Based Services* 4.2, pp. 93–104.

Kozlowski, Pawel (2013). *Mastering Web Application Development with AngularJS*. Packt Publishing Ltd.

Krajzewicz, D, Georg Hertkorn, Julia Ringel, and Peter Wagner (2015). "Preparation of Digital Maps for Traffic Simulation; Part 1: Approach and Algorithms". In: *3rd Industrial Simulation Conference 2005*. URL: http://elib.dlr.de/21013/.

Kreil, Michael (2012). *OpenPlanB: innovation without permission*. URL: http://openplanb.tumblr.com/.

Kühnel, Nico (2014). "Grafisches Editieren von Straßen- und ÖV-Netzmodellen". Bachelor's Thesis. Technische Universität Berlin.

Laarabi, Mohamed Haitam and Raffaele Bruno (2016). "A Generic Software Framework for Carsharing Modelling Based on a Large-Scale Multi-agent Traffic Simulation Platform". In: *International Workshop on Agent Based Modelling of Urban Systems*. Springer, pp. 88–111.

Liao, Lin, Dieter Fox, and Henry Kautz (2007). "Extracting places and activities from GPS traces using hierarchical conditional random fields". In: *The International Journal of Robotics Research* 26.1, pp. 119–134.

Ma, Jingtao, Huan Li, Fang Yuan, and Thomas Bauer (2013). "Deriving operational origin-destination matrices from large scale mobile phone data". In: *International Journal of Transportation Science and Technology* 2.3, pp. 183–204. DOI: 10.1260/2046-0430.2.3.183.

Mann, M. *Continuations Library*. URL: http://www.matthiasmann.de/.

Mascolo, Cecilia (2010). "The power of mobile computing in a social era". In: *IEEE Internet Computing* 14.6, p. 76.

McKenzie, Grant (2011). "Gamification and location-based services". In: *Workshop on Cognitive Engineering for Mobile GIS*. URL: http://ceur-ws.org/Vol-780/paper4.pdf.

Moyo Oliveros, M. and K. Nagel (2012). *Automatic Calibration of Microscopic, Activity-Based Demand for a Public Transit Line*. Annual Meeting Preprint 12-3279. Also VSP WP 11-13, see http://www.vsp.tu-berlin.de/publications. Washington, D.C.: Transportation Research Board.

– (2016). "Automatic calibration of agent-based public transit assignment path choice to count data". In: *Transportation Research Part C*, pp. 58–71. DOI: 10.1016/j.trc.2016.01.003.

Müller, Kirill and Kay W Axhausen (2010). *Population synthesis for microsimulation: State of the art*. Tech. rep. DOI: 10.3929/ethz-a-006132973.

Nagel, K. (2008). *Towards simulation-based sketch planning: Some results concerning the Alaskan Way viaduct in Seattle WA*. VSP Working Paper 08-22. See http://www.vsp.tu-berlin.de/publications. TU Berlin, Transport Systems Planning and Transport Telematics.

– (2011). *Towards simulation-based sketch planning, part II: Some results concerning a freeway extension in Berlin*. VSP Working Paper 11-18. See http://www.vsp.tu-berlin.de/publications. TU Berlin, Transport Systems Planning and Transport Telematics.

– (2016). "Matrix-Based pt router". In: *The Multi-Agent Transport Simulation MATSim*. Ed. by A. Horni, K. Nagel, and K. W. Axhausen. Ubiquity, London. Chap. 20. DOI: 10.5334/baw.

Nagel, K. and K. W. Axhausen (2016). "Some History of MATSim". In: *The Multi-Agent Transport Simulation MATSim*. Ed. by A. Horni, K. Nagel, and K. W. Axhausen. Ubiquity, London. Chap. 46. DOI: 10.5334/baw.

Nagel, K and G Flötteröd (2012). "Agent-based traffic assignment: Going from trips to behavioural travelers". In: *Travel Behaviour Research in an Evolving World – Selected papers from the 12th international conference on travel behaviour research*. Ed. by R M Pendyala and C R Bhat. International Association for Travel Behaviour Research, pp. 261–294.

Nicolai, T. W. and K. Nagel (2014). "High resolution accessibility computations". In: *Accessibility and Spatial Interaction*. Ed. by A. Condeço, A. Reggiani, and J. Gutiérrez. Edward Elgar, pp. 62–91.

Ordonez, S. and Alexander Erath (2011). *Semi-automatic tool for map-matching bus routes on high-resolution navigation networks*. Tech. rep. DOI: 10.3929/ethz-a-006742032.

Parallel Universe. *Quasar*. URL: http://docs.paralleluniverse.co/quasar/.

Pendakur, Setty (2005). "Non-motorized transport in African cities: Lessons from experience in Kenya and Tanzania". In: *Sub-Saharan Africa Transport Policy Program Working Paper* 80.

Project Reactor. *Reactor*. URL: http://projectreactor.io/.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.

Raney, B. and K. Nagel (2006). "An improved framework for large-scale multi-agent simulations of travel behaviour". In: *Towards better performing European Transportation Systems*. Ed. by P. Rietveld, B. Jourquin, and K. Westin. London: Routledge, pp. 305–347.

Rieser, M. (2010). "Adding Transit to an Agent-Based Transportation Simulation: Concepts and Implementation". PhD thesis. DOI: 10.14279/depositonce-2581.

Rieser, M., D. Grether, and K. Nagel (2009). "Adding mode choice to a multi-agent transport simulation". In: *Transportation Research Record* 2132, pp. 50–58. DOI: 10.3141/2132-06.

Rieser, M., A. Horni, and K. Nagel (2016). "Let's Get Started". In: *The Multi-Agent Transport Simulation MATSim*. Ed. by A. Horni, K. Nagel, and K. W. Axhausen. Ubiquity, London. Chap. 2. DOI: 10.5334/baw.

Rieser-Schüssler, N., M. Balmer, and K. W. Axhausen (2012). "Route choice sets for very high-resolution data". In: *Transportmetrica* 9.9, p. 10021. DOI: 10.1080/18128602.2012.671383.

Scheiner, J. (2005). "Daily mobility in Berlin: On 'inner unity' and the explanation of travel behaviour". In: *European Journal of Transport and Infrastructure Research* 5, pp. 159–186.

Schneider, S. (2011). "A methodology for the extrapolation of trip chain data". PhD thesis. Technische Universität Berlin. URL: http://elib.dlr.de/74230/.

Schröder, S., M. Zilske, G. Liedtke, and K. Nagel (2012). *A computational framework for a multi-agent simulation of freight transport activities*. Annual Meeting Preprint 12-4152. Also VSP WP 11-19, see http://www.vsp.tu-berlin.de/publications. Washington D.C.: Transportation Research Board.

Schulz, Benjamin (2011). "Abbiegespuren und Kreuzungslayout in OpenStreetMap". Bachelor's Thesis. Technische Universität Berlin.

Strauch, Christof, Ultra-Large Scale Sites, and Walter Kriha (2011). "NoSQL databases". In: *Lecture Notes, Stuttgart Media University*.

Strippgen, D. (2009). "Investigating the Technical Possibilities of Real-time Interaction with Simulations of Mobile Intelligent Particles". PhD thesis. Technische Universität Berlin. DOI: 10.14279/depositonce-2272.

Tran, Khoa, Sean Barbeau, Edward Hillsman, and Miguel A Labrador (2013). "GO_Sync-A Framework to Synchronize Crowd-Sourced Mapping Contributors from Online Communities and Transit Agency Bus Stop Inventories". In: *International Journal of Intelligent Transportation Systems Research* 11.2, pp. 54–64.

Weaver, James, Weiqi Gao, Stephen Chin, Dean Iverson, and Johan Vos (2012). *Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology*. Apress.

Wesolowski, Amy, Nathan Eagle, Abdisalan M Noor, Robert W Snow, and Caroline O Buckee (2013). "The impact of biases in mobile phone ownership on estimates of human mobility". In: *Journal of The Royal Society Interface* 10.81, p. 20120986.

Wolf, J., S. Schönfelder, U. Samaga, M. Oliveira, and K. W. Axhausen (2004a). "Eighty weeks of global positioning system traces: approaches to enriching trip information". In: *Transportation Research Record: Journal of the Transportation Research Board* 1870, pp. 46–54.

Wolf, Jean, Randall Guensler, and William Bachman (2001). "Elimination of the Travel Diary: Experiment to Derive Trip Purpose from Global Positioning System Travel Data". In: *Transportation Research Record* 1768.1, pp. 125–134.

Wolf, Jean, Stacey Bricka, T Ashby, and C Gorugantua (2004b). "Advances in the application of GPS to household travel surveys". In: *National Household Travel Survey Conference, Washington DC*.

Zhang, Chao, Carolina Osorio, and Gunnar Flötteröd (2017). "Efficient calibration techniques for large-scale traffic simulators". In: *Transportation Research Part B: Methodological* 97, pp. 214–239.

Zielstra, Dennis and Alexander Zipf (2010). "A comparative study of proprietary geodata and volunteered geographic information for Germany". In: *13th AGILE international conference on geographic information science*. URL: http://agile2010.dsi.uminho.pt/pen/ShortPapers_PDF%5C142_DOC.pdf.

Zilske, M. and K. Nagel (2012). "Towards volunteered digital travel demand data". In: *7th International Conference on Geographic Information Science, Extended Abstracts*. URL: http://www.giscience.org/proceedings/abstracts/giscience2012_paper_119.pdf.

– (2013). "Building a minimal traffic model from mobile phone data". In: *NetMob 2013, 3rd International Conference on the Analysis of Mobile Phone Datasets, Special session on the D4D challenge*. Also VSP WP 13-03, see http://www.vsp.tu-berlin.de/publications, MIT (Cambridge, MA).

– (2014). "Studying the accuracy of demand generation from mobile phone trajectories with synthetic data". In: *Procedia Computer Science* 32, pp. 802–807. ISSN: 1877-0509. DOI: 10.1016/j.procs.2014.05.494.

– (2015). "A Simulation-based Approach for Constructing All-day Travel Chains from Mobile Phone Data". In: *Procedia Computer Science* 52, pp. 468–475. ISSN: 1877-0509. DOI: 10.1016/j.procs.2015.05.017.

– (2016). "Software architecture for a transparent and versatile traffic simulation". In: *International Workshop on Agent Based Modelling of Urban Systems. ABMUS 2016. Lecture Notes in Computer Science, vol 10051.* Ed. by Mohammad-Reza Namazi-Rad, Lin Padgham, Pascal Perez, Kai Nagel, and Ana Bazzan, pp. 73–87.

Zilske, Michael, Andreas Neumann, and Kai Nagel (2011). "OpenStreetMap for Traffic Simulation". In: *Proceedings of the 1st European State of the Map – OpenStreetMap conference*. Ed. by M Schmidt and G Gartner. Vienna, pp. 126–134.

Zilske, Michael, Stefan Schröder, Kai Nagel, and Gernot Liedtke (2012). *Adding freight traffic to MATSim*. VSP Working Paper 12-02. See `http://www.vsp.tu-berlin.de/publications`. TU Berlin, Transport Systems Planning and Transport Telematics.