Public Transit Mapping on Multi-Modal Networks in MATSim

Flavio Poletti

Master thesis

Institute for Transport Planning and Systems **July 2016**

Institut für Verkehrsplanung und Transportsysteme
Institute for Transport Planning and Systems

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Contents

List of Figures

List of Tables

Acronyms

GPS Global Positioning System

GTFS General Transit Feed Specification

HAFAS HaCon Fahrplan-Auskunfts-System

JOSM Java OpenStreetMap Editor

OSM OpenStreetMap

MATSim Multi-Agent Transport Simulation

railML Railway Markup Language

SBB Schweizerische Bundesbahnen

Master thesis

Public Transit Mapping on Multi-Modal Networks in MATSim

Flavio Poletti
IVT
ETH Zürich

CH-8093 Zürich

polettif@ethz.ch

July 2016

Abstract

This thesis proposes an algorithm to find the path a public transit vehicle takes on a network based on its stop sequence. This path is required to simulate mixed traffic in a transport simulation and thus to observe effects of the interaction between public transit vehicles and private car traffic in a transport simulation. The algorithm has been implemented for MATSim, an agent-based transport simulation framework. The implementation also provides tools to convert HaCon Fahrplan-Auskunfts-System (HAFAS), General Transit Feed Specification (GTFS) and OpenStreetMap (OSM) data to MATSim transit schedules as well as a tool to generate a multimodal MATSim network from OSM. The output of the package consists of a multi-modal network and a transit schedule that contains the link sequences for all transit routes. To find, show and fix implausibilities in the schedule and the network, basic tools are provided. The implemented algorithm has been tested for the Zurich area with a schedule generated from a GTFS data source provided by ZVV.

Keywords

Mapping, Map-Matching, Public Transit, Transit Routes, MATSim

Preferred citation style

Poletti, F. (2016) Public Transit Mapping on Multi-Modal Networks in MATSim, Master thesis, Institute for Transport Planning and Systems, ETH Zurich, Zurich.

Masterarbeit

Public Transit Mapping on Multi-Modal Networks in MATSim

Flavio Poletti
IVT
ETH Zürich

CH-8093 Zürich

polettif@ethz.ch

Juli 2016

Zusammenfassung

Um Auswirkungen der Interaktion von Bussen und dem motorisierten Individualverkehr in einer Verkehrssimulation zu sehen, muss für jedes ÖV-Fahrzeug bekannt sein, welchen Weg es nimmt. In dieser Masterarbeit wird ein Algorithmus vorgestellt, mit dem dieser Weg auf einem Netzwerk nur anhand der Haltestellenabfolge und -koordinaten gefunden werden kann. Der Algorithums wurde für MATSim, eine agentenbasierte Verkehrssimulation, umgesetzt. Die Implementierung bietet auch Tools um HAFAS, GTFS und OSM Daten zu MATSim Fahrplänen zu konvertieren und um ein multimodales MATSim Netzwerk anhand von OSM zu erzeugen. Der Algorithmus erstellt ein multimodales Netzwerk und einen MATSim Fahrplan, welche Wege für alle Fahrzeuge enthält. Es werden Werkzeuge zur Verfügung gestellt, um Unplausibilitäten in diesem Fahrplan zu finden und zu beheben. Der implementierte Algorithmus wurde mit einem Fahrplan und einem Netzwerk für die Region Zürich getestet.

Schlüsselwörter

Mapping, Öffentlicher Verkehr, MATSim

Bevorzugter Zitierstil

Poletti, F. (2016) Public Transit Mapping on Multi-Modal Networks in MATSim, Masterarbeit, Institut für Verkehrsplanung und Transportsysteme, ETH Zürich, Zürich.

Acknowledgement

I would like to thank Patrick Bösch for his support and inputs on all kinds of problems that came up while working on this thesis. Especially the discussions were very helpful and usually lead to solutions or new approaches. I would also like to thank Prof. Dr. Kay W. Axhausen for his inputs on possible approaches.

1 Introduction

Public transit vehicles such as buses interact with private traffic. They can be stuck in traffic which leads to delays or they can cause traffic jams if they stop on a road. To see such effects in a transport simulation like MATSim, the path a public transit vehicle takes has to be known. These paths are normally not available from public transit data sources and have to be generated. Defining this path (mapping) is not trivial as schedules normally only provide the stop sequence of a transit route. In addition, depending on the input data, precise stop locations are often not available, i.e. multiple stop locations on different roads are combined to one parent stop with the same name in the schedule. Map matching algorithms using Global Positioning System (GPS) points to find a path on a network are well documented (Quddus et al., 2007). In contrast, literature on mapping public transit routes to a network without using GPS data is rather sparse. Algorithms to find paths have been proposed by Bösch and Ciari (2015) who only use an unmapped schedule as input and Ordonez and Erath (2011) who include GPS data to find the path. Another approach described by geops (2014) uses OSM data to improve the data quality.

MATSim is a multimodal agent based transport simulation which models the travel behaviour of a synthetic population on a network during one simulated day (Horni et al., 2016). MATSim also provides public transit simulation (Rieser, 2010). Up to now, public transit vehicles in MATSim used a separate network from private cars. Simulations could be quickly set up that way and transforming raw schedule data to a separate network is easy (Nagel et al., 2016b). MATSim supports multimodal networks where all transport modes use the same network. However, for each public transit route, a path has to be specified which has to be followed precisely by the vehicle driver agent (Rieser, 2016a). Implementations for MATSim to find those paths have been proposed by Bösch and Ciari (2015) and Ordonez and Erath (2011). Transit schedules for MATSim are normally not generated from scratch but converted from schedule data exchange formats. Widely used formats are GTFS (Google, 2016) and HAFAS (HaCon, 2016). Tools to convert such data to a MATSim transit schedule have been developed by Zilske and Kühnel (2016) and Ordonez and Erath (2011) for GTFS and by Bösch and Ciari (2015) for HAFAS. These converters generate an unmapped MATSim transit schedule. Unmapped means that the stop sequence and departure times are defined but not the path and the reference link for each stop facility. Networks in MATSim are normally generated from OSM (Nagel et al., 2016a). Kühnel and Zilske (2016) have developed a plugin for JOSM (Java OpenStreetMap Editor) which allows to edit MATSim networks and transit schedules. The plugin also allows to generate a MATSim network from OSM.

This thesis proposes an algorithm to find the network path for a transit route given its stop

sequence. The algorithm uses an abstract graph to calculate the least cost path from the transit route's first to its last stop with the constraint that the path must contain a so called link candidate for every stop. The algorithm has been implemented in Java for MATSim. The implementation also provides tools to convert HAFAS, GTFS and OSM data to MATSim transit schedules and to generate a multimodal network from OSM. The output of the package consists of a mapped MATSim transit schedule and a modified network. Tools to find, show and fix implausibilities in the schedule and the network mapping are provided as well. The implemented algorithm has been tested for the Zurich area with a schedule generated from GTFS.

Concerning the structure of this report, Section 2 describes different transit schedule data formats. The problem of mapping a public transit schedule to a network, existing approaches and the proposed so called pseudo routing algorithm are presented in Section 3. The implementation of the algorithm and the aforementioned converters is described in Section 4. In Section 5 the methods to test the implementation as well as the test results of the Zurich data set are described.

2 Public Transit Data Formats

2.1 Definitions

While working with public transit data, one comes across different names and definitions for elements of a public transit schedule. Table 1 shows the definitions used within this thesis and their counterparts in other data formats. The basic element of a schedule is a trip. It is defined by a sequence of stops, the arrival and departure time at each stop and the path on a network for a vehicle. Trips with the same stop sequence and path can be combined to a transit route. A transit route has one or more departures. A departure is given by the departure time at the first stop. In real life, similar transit routes are often grouped to a transit line (e.g. one transit route for each direction). Transit lines are normally referred to with a number or colour (e.g. line 3 or red line). Fig. 1 shows an example network containing three transit lines.

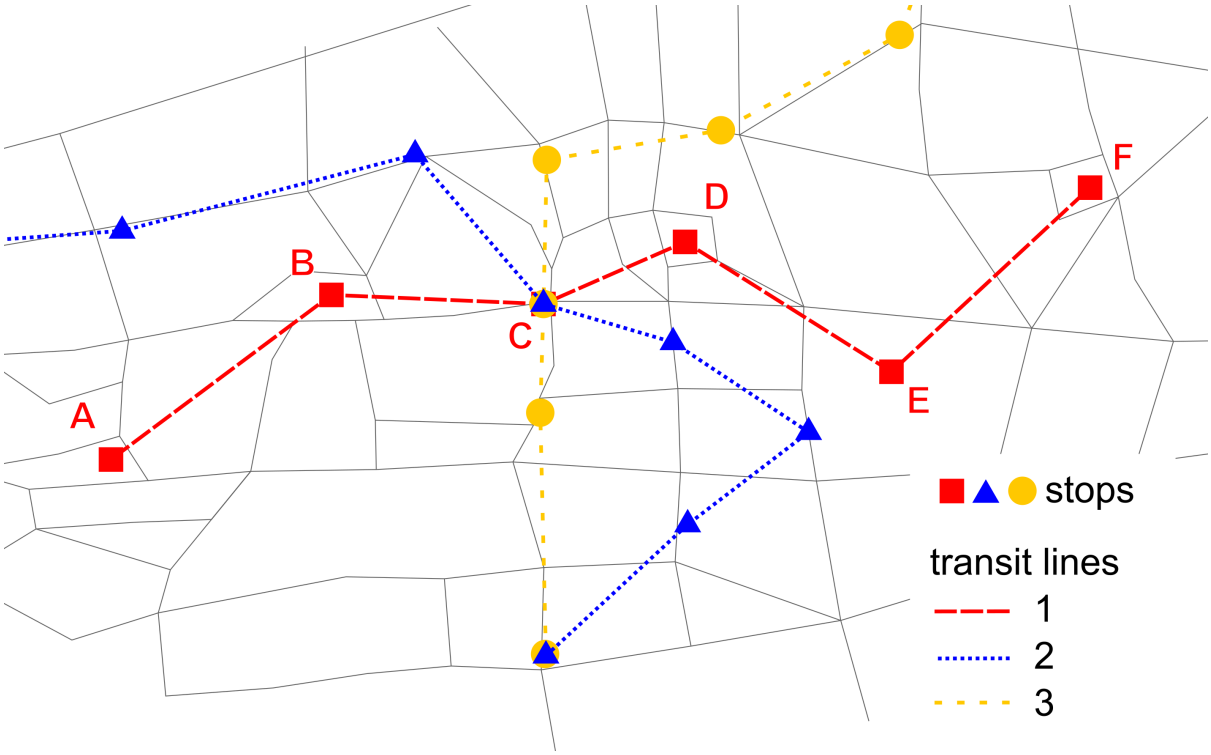
There are two types of stops that can appear in a stop sequence: stop locations or parent stops. Both are referenced with point coordinates. Fig. 2 shows stop C from Fig. 1 which is located at an intersection and used by three bus lines. Each bus line has two transit routes. The stop has four stop locations, where the buses halt and pick up or drop off passengers. All four stop locations usually have the same name and can be grouped to a parent stop. The stops in a stop sequence can be either stop locations or parent stops, depending on the data source.

2.2 Public transit in MATSim

In the first implementation of public transit in MATSim, agents were teleported. They were removed from one location and placed at another location after the estimated travel time. This approach is still available in MATSim. Today's standard however is simulating public transit in QSim (the standard mobility simulation) where public transit vehicles serve stops along a fixed route with a given schedule (Rieser, 2016a).

Public transit is based on two input files, `transitVehicles.xml` and `transitSchedule.xml`. The first file describes the vehicles serving a transit route. Every vehicle has a vehicle type (e.g. a small bus, a normal bus, trolleybus, a train or light rail vehicles) which defines the vehicle's length, capacity and other attributes. The transit schedule consists of two parts (Fig. 3 shows the entity relationship model of the data structure). First there is a list of *stop facilities*. Each stop facility has an id, coordinates, a referenced link and optionally a name. The referenced

Figure 1: An example network containing three public transit lines: Line 1 (dashed red), line 2 (dotted blue) and line 3 (dashed yellow). Line 1 is used to illustrate the mapping algorithm in Section 3. All transit lines have two transit routes, one going from the first to the last stop and one transit route back. Stop C is used by all three transit lines.



link defines where transit vehicles stop and agents board or leave the vehicle. Stop facilities define the position where passengers wait for a vehicle. This means that MATSim stop facilities should be regarded as stop locations. Parent stops cannot be modelled directly (Rieser, 2016a). This means that for a simple bus stop with one platform for each direction, two stop facilities are needed since the referenced links of the two platforms differ. There is an optional attribute *stop area* for a stop facility which might be used to group facilities. However, this attribute is currently not widely used.

The second part of the MATSim transit schedule consists of the transit lines. A `transit line` consists of one or more `transit routes`. Normally a line has two transit routes, one from the first stop to the last and one back. Additional routes might be used for service routes (coming or going to a depot) or partial routes during peak hours. Each transit route contains a stop sequence (`routeProfile`), a path the vehicles passes (`route`) as a sequence of links and a list of departures that define when a vehicle starts at the first route stop. Each stop contains the arrival and departure time as offset from the departure at the first stop (Rieser, 2010). It is possible to define a transport mode for a transit route. It is also possible to assign allowed transport modes

Figure 2: Difference between parent stop and stop locations. Stop C (see Fig. 1) at an intersection is passed by three bus transit lines with two transit routes each.

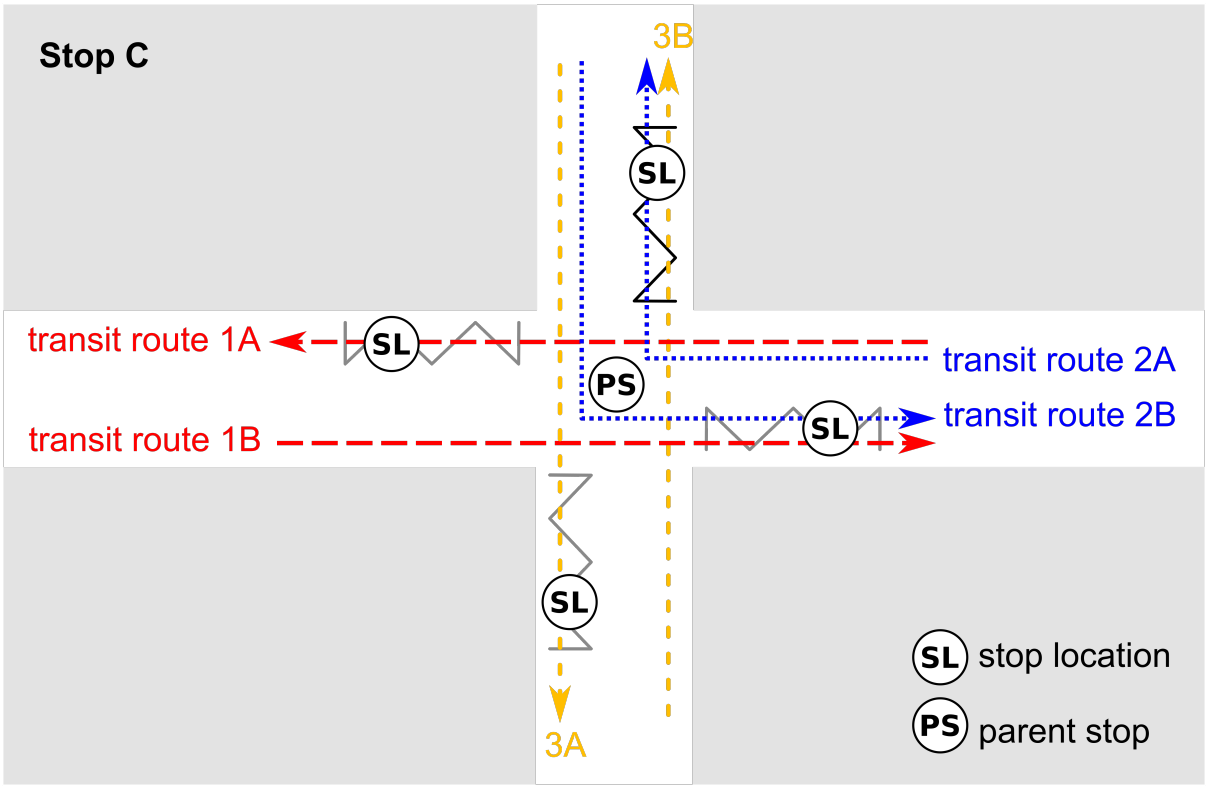
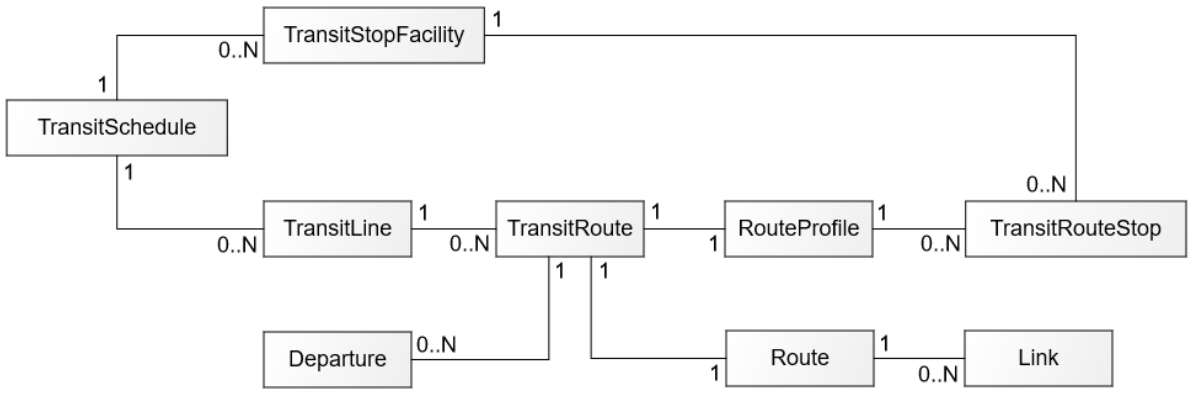


Figure 3: Entity relationship model of the MATSim transit schedule data structure.



Source: Based on Rieser (2010)

to a link in a MATSim network, creating a multimodal network. However, this information is currently not used in the simulation. All transit vehicles declare to use the network mode “car”, regardless of their actual type like train or bus (Rieser, 2016b).

Table 1: The following names for elements of a transit schedule are used within this thesis.

Element	**MATSim**	**GTFS**
stop	TransitRouteStop	Stop
- stop location	Transit Stop Facility	Stop
- parent stop	-	Stop, Parent station (optional)
stop sequence	Route Profile	defined in `stop_times.txt` for each trip
path	Route	Shape (optional)
arrival time	Arrival Offset (in relation to the first departure time)	Defined in `stop_times.txt` for each trip. Alternatively defined as headway during a given time period in `frequencies.txt`
departure time	Departure Offset (in relation to the first departure time)	Same as arrival
departure	Departure	-
trip	-	Trip
transit route	TransitRoute	-
transit line	TransitLine	Route

2.3 GTFS (General Transit Feed Specification)

A common format for public transit schedules and geographic information is the General Transit Feed Specification, GTFS (Google, 2016). It has been developed by Google as the *Google Transit Feed Specification* but has been renamed in 2010 since the specification started being used by applications of third parties. GTFS feeds are published by several hundred public transit agencies worldwide (see Transitland (2016)) which makes it an useful source for MATSim transit schedules.

A GTFS feed consists of several comma separated files (saved as text files in a zipped folder) that contain information about transit lines, stop locations, timetables and optional data such as transfer times or fares. Table 2 lists all files that are specified by GTFS. The approaches proposed in this thesis use transport modes based on the eight transport modes supported by GTFS (Table 3).

Table 2: The files defined by GTFS

File	Status	Description
agency.txt	**Required**	One or more transit agencies that provide the data in this feed.
stops.txt	**Required**	Individual locations where vehicles pick up or drop off passengers.
routes.txt	**Required**	Transit routes. A route is a group of trips that are displayed to riders as a single service.
trips.txt	**Required**	Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
stop_times.txt	**Required**	Times that a vehicle arrives at and departs from individual stops for each trip.
calendar.txt	**Required**	Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.
calendar_dates.txt	Optional	Exceptions for the service IDs defined in the calendar.txt file. If calendar_dates.txt includes ALL dates of service, this file may be specified instead of calendar.txt.
fare_attributes.txt	Optional	Fare information for a transit organization's routes.
fare_rules.txt	Optional	Rules for applying fare information for a transit organization's routes.
shapes.txt	Optional	Rules for drawing lines on a map to represent a transit organization's routes.
frequencies.txt	Optional	Headway (time between trips) for routes with variable frequency of service.
transfers.txt	Optional	Rules for making connections at transfer points between routes.
feed_info.txt	Optional	Additional information about the feed itself, including publisher, version, and expiration information.

Source: Google (2016)

Table 3: The transport modes available in GTFS.

ID	Mode	Description
0	Tram, Streetcar, Light rail	Any light rail or street level system within a metropolitan area.
1	Subway, Metro	Any underground rail system within a metropolitan area
2	Rail	Used for intercity or long-distance travel
3	Bus	Used for short- and long-distance bus routes
4	Ferry	Used for short- and long-distance boat service
5	Cable car	Used for street-level cable cars where the cable runs beneath the car
6	Gondola, Suspended cable car	Typically used for aerial cable cars where the car is suspended from a cable
7	Funicular	Any rail system designed for steep inclines

Source: Google (2016)

2.4 HAFAS (HaCon Fahrplan-Auskunfts-System)

HAFAS (HaCon, 2016) is a data format used by Swiss public transit agencies (particularly Schweizerische Bundesbahnen (SBB)) and other agencies in Europe (e.g. Deutsche Bahn). As Rieser (2010) states, information about the "HAFAS Rohdatenformat" (HAFAS raw data format) is scarse. Data is given in ASCII text files that contain information about operators, stops, vehicles and stop sequences, stop times and additional information. HAFAS data for Switzerland is publicly available and provided by SBB (Offizielles Kursbuch: Fahrplandaten, 2016). A report that comes with this exported data provides information on the structure of the files (SBB, 2016).

2.5 OSM (OpenStreetMap)

MATSim networks are normally generated from OSM (Nagel et al., 2016a). OSM is a free and editable map of the world, released with an open-content license (OpenStreetMap Wiki, 2016a). According to Zilske et al. (2011), OSM data has proven to be a valuable data source for network creation. Other sources for network creation (e.g. Vector25 from Bundesamt für Landestopografie swisstopo (2016)) are not discussed further within this thesis. Their often proprietary nature make it difficult to share the generated networks and it is usually necessary to

combine multiple data sets to reach the level of information OSM offers in one source. Very high point coordinate accuracy, a strength of commercial data sources, is normally not needed for MATSim applications.

OSM uses the elements node, way and relation, structured in an xml-format. A node is a single point in space defined by its latitude, longitude and id. A way connects two nodes, but ways are not only used for roads but all line elements present on a map. Relations group ways and nodes to logical entities such as buildings, rivers, forests, borders or public transit routes. OSM offers tags to define spatial public transit data such as routes and stops. Time information (i.e. departure times for each stop) cannot be tagged. Using OSM data in a fully automated process to create public transit schedules is not yet feasible. There are several ways to tag and implement a transit route, and the level of detail varies. In addition, there are no conventions to name or identify routes which makes it difficult to join the data set with other schedule sources. OSM provides the means to define the actual path a vehicle takes. However, data and tagging are very inconsistent (Talk-transit mailing list, 2016). Often parts of a route are missing or misplaced, making automatic processing of route information difficult. Stop locations are normally accurate. Unfortunately, often a logical connection to routes or streets is missing which makes it difficult to build a transit schedule using only OSM.

There are tools available to convert GTFS data to OSM, for example GO-Sync (Tran et al., 2013). There are currently no approaches to merge mapped MATSim transit schedule data to OSM. While the API to modifiy OSM is available (OpenStreetMap Wiki, 2016b), it probably will not be easy to automate such a process. The data formats differ and there is currently no consensus within the OSM community how to map public transit data (Talk-transit mailing list, 2016). In addition, the problem with differing ids and names arises as well. Trafimage Webkarten (2016) shows the difference in id and location of stops from different sources in Switzerland (including HAFAS and OSM).

2.6 railML

Another standard to exchange public transit data is Railway Markup Language (railML). It is an open source data exchange format for railroad data and based on XML (Hansen, 2010). It is mainly used by railway agencies in Europe, for example ÖBB, Deutsche Bahn and SBB (railML.org, 2016). Since the railML standard focuses on railway services and is not intended for roadbound public transit, railML has not been further used within this thesis.

2.7 PTV VISUM

PTV VISUM uses stops as elements of a stop sequence. Stop areas and stop points locate the stop on a network. *Lines* are made up of one or more *line routes*. The path on the network for a line route is stored as a sequence of *route points*. These route points are either stops or network nodes the line route passes (PTV AG, 2014, 44). VISUM separates routes and time profiles. It stores data in ASCII tables (Rieser, 2010) and allows data export in HAFAS and railML (among other data exchange formats) (PTV AG, 2016).

3 Mapping a public transit schedule to a network

3.1 Problem definition

The path a public transit vehicle takes can be used in a transport simulation to see effects of the interaction between public transit vehicles and the rest of the traffic or simply for visualisation purposes. This path needs to be generated from transit schedule data. However, schedule data typically does not provide vehicle paths and even if they are available, using them to generate link sequences for a MATSim transit schedule comes with its own difficulties.

The problem can be defined as: A reference link for each stop and the path between those referenced links have to be identified. To generate the paths, the following input data is usually available for each transit route:

- stop sequence
- stops (either as stop locations or parent stops)
- a network as a directed graph

The algorithm should work without using any additional GPS data. Such data is rarely available for all routes of a schedule and gathering it for large areas is expensive.

It is assumed that a public transit vehicle can access a stop by passing a link which is referenced to the stop. A problem which arises frequently is schedule data sources using parent stops instead of stop locations which makes mapping with bus hubs or larger rail stations complicated. When two lines use a stop at an intersection there might be one to four actual stop locations.

3.2 Literature

Map matching using GPS points is well documented (Quddus et al., 2007) but literature on mapping public transit routes to a network without using GPS data is rather sparse.

Bösch and Ciari (2015) provide such an algorithm. It looks for the closest node from a stop facility and then the nearest outlink. This link is set as the referenced link for this stop facility. Then the shortest path between all referenced links is calculated. If there are no nodes within a given search radius, a new node is created at the stop facility's location. This node is connected

with an artificial link to the previous stop link's end node. The artificial link is set as the stop facility's reference link. This ensures that all stop facilities can be accessed and thus a valid schedule can be created.

Ordonez and Erath (2011) propose a semi-automatic procedure, still using only one link per stop. An automatic map-matching algorithm is performed on a route. The algorithm goes through all stops and identifies the shortest path from the previous stop's link to the current stop's link. If a stop does not have a link referenced, a set of link candidates is created. The shortest path is calculated from the previous stop link to each defined candidate. The shortest path algorithm includes travel time and distance to the GPS points for link costs. The path with the lowest cost is part of the solution and its last link is selected and assigned to the stop. The reference link for the first stop is calculated similarly. Once a link is referenced to a stop facility, all other transit routes using this stop must use this link, which makes the order of the transit routes assignment crucial. The created path is verified automatically and errors can be fixed manually using a GUI (Graphical User Interface).

Brosi (2014) suggests some ideas on how to map public transit trajectories to a network, for example iteratively computing shortest paths between stops. Pursuing this approach, geops (2014) describe an algorithm consisting of the following four steps. Contrary to the problem definition mentioned above, stops are referenced to nodes in the network instead of links.

1. Build a graph from rail or road geometries and insert stops from GTFS.
2. Look at every trip in the GTFS feed and calculate the shortest path between every two succeeding stops.
3. Check for plausibility.
4. Filter and compress shapes to avoid redundancy.

Even though geops (2014) use GTFS as input data, the algorithm might be applied to all data formats where the stop location and the stop sequence of a transit route are given. During the first step, GTFS and OSM data are combined to increase the accuracy of stop coordinates. Stop name or id notation in both formats do not follow any schema. The algorithm uses attributes like equality of station id, distance and similarity of station name to create a priority queue. The stop is referenced to the node with the highest priority. To enable shortest path search in the second step, a predefined number of node candidates is taken from the priority queue and connected with the best node. This is especially necessary for rail networks. The algorithm then calculates the least cost path between two succeeding stop nodes. The edge costs are calculated based on heuristics. Since node candidates have been connected, the least cost path algorithm is highly likely to find a path from each stop to the next. In the third step the calculated path from the first

to the last stop is checked for plausibility by comparing the route length with the beeline distance. Filter and compression ensure in the fourth step that paths that appear in multiple routes are stored only once.

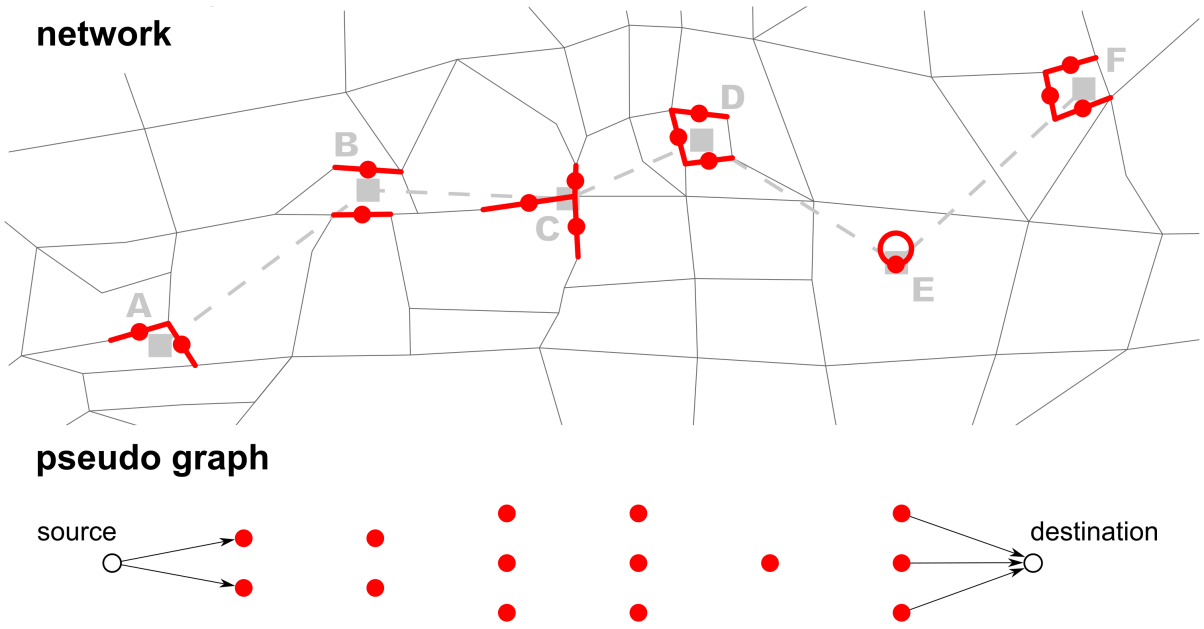
3.3 Public transit mapping algorithm

In this section a public transit mapping algorithm to find the path of a transit route is suggested. This "Pseudo Routing" algorithm requires a schedule in which each transit route has a sequence of stops defined. The coordinates of the stops have to be already defined as well. It is not required but useful if these stops represent stop locations instead of generalised parent stops. A network is not required as such. If no network is available, the pseudo routing algorithm simply creates an artificial, separated network for public transit. The implementation for MATSim is covered in Section 4.

An example setup for a schedule and network can be seen in Fig. 1. In difference to the previously described approaches, the pseudo routing algorithm not only looks at pairs of stops to define the best link for each stop but at the whole route. It is also possible that more than one optimal link can be found for a stop. The algorithm calculates the least cost path from the transit route's first to its last stop with the constraint that the path must contain a link candidate of every stop. For each transit route, the algorithm consists of the following steps:

1. Find link candidates for each stop.
2. Create a pseudo graph using the link candidates as nodes. Add a dummy source and destination node to the pseudo graph.
3. Calculate the least cost path between each link candidate pair. This path is represented by an edge in the pseudo graph, connecting two link candidate nodes. The edge weight is the path's travel cost plus half the travel cost of the two link candidates it connects.
4. Calculate the pseudo least cost path from the source node to the destination node in the pseudo graph. The resulting least cost path contains the best link candidate for each stop.
5. Create the link sequence. Each stop is referenced to a link (given by the best link candidate that is part of the pseudo least cost path). The least cost path on the real network between the referenced links is used to create the network path for the transit route.

Figure 4: Each link candidate (red) is represented as a node on the pseudo graph.



3.3.1 Finding link candidates

For each stop, a set of link candidates is gathered by selecting the nearest n links from the stop's coordinate. The link's transport modes need to match the transport mode used by the transit route. Sometimes, no link candidates can be found because there are no links within a predefined distance. In this case, an artificial loop link is created because all stop facilities need to be referenced to a link. This is done by adding a node to the network at the coordinates of the stop. Then a loop link that connects this node with itself is added. This loop link is the stop's only link candidate.

3.3.2 Creating a pseudo graph

In the next step, a pseudo graph is initialised with each link candidate represented by a node (Fig. 4). Note that these nodes do not have any actual coordinates. To efficiently calculate the least cost path on the pseudo graph, dummy source and destination nodes are needed. The source node is connected to all link candidate nodes of the first stop, the destination node to all nodes of the last stop. All these dummy edges have the same weight (e.g. 1).

3.3.3 Calculating least cost path between each link candidate pair

In the previous step, a set of link candidates for each stop has been created. These link candidates are represented as nodes in the pseudo graph. In this step the edges of the pseudo graph are added. For each pair of link candidates of two adjacent stops, the least cost path on the network is calculated. This path is represented by an edge on the pseudo graph (Fig. 5). The pseudo edge's weight is the path travel cost plus half the travel costs of the two link candidates the path connects. It is possible that two adjacent stops share a link candidate. Then, the travel cost between the "two" candidates (i.e. the travel cost on the link) is multiplied by four. This is done to impede the same link being referenced to two stops.

The travel cost on a network link is normally length or travel time. More complex travel cost calculations are also possible. Following Ordonez and Erath (2011), GPS point data or data from OSM could be included to in- or decrease the travel cost accordingly. If no path can be found between two link candidates, an artificial link is created and added as edges to the pseudo graph (see Figure 5 between stops C and D). This ensures that a path can be found in the pseudo graph and thus also in the network. Artificial links can also be created if the cost of a path is greater than a defined threshold. This prevents paths that have too high travel costs and are thus unlikely to be correct.

3.3.4 Calculate the pseudo least cost path

The shortest path from the source node to the destination node is calculated. Any shortest path algorithm can be used. The least cost path gives a sequence of link candidates that describes which link should be referenced to each stop of the transit route (see Fig. 6).

3.3.5 Create link sequence

After each stop has a link referenced, one can use the least cost path between each reference link pair to define the path (link sequence) the vehicle takes on the network (Fig. 6).

Figure 5: The least cost paths on the network between two link candidates (top) is represented as one edge on the pseudo graph (bottom). The pseudo edge's weight is the travel cost of the network path. Six paths and their corresponding edges on the pseudo graph are highlighted as examples.

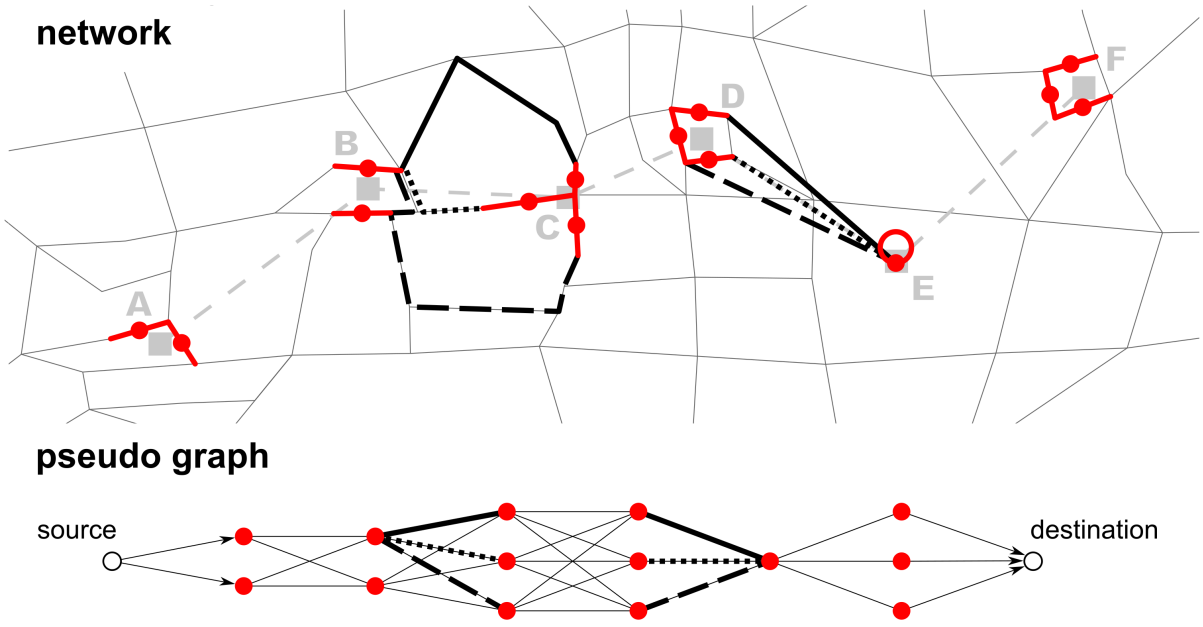
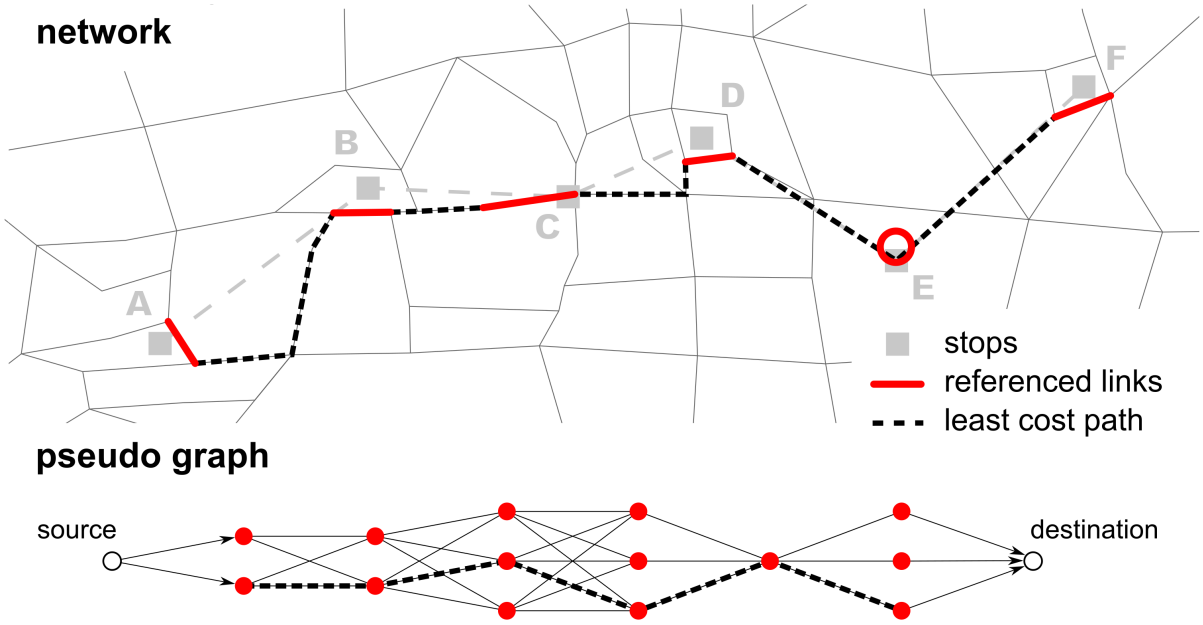


Figure 6: The least cost path from the source to the destination on the pseudo graph is calculated. The nodes of this path represent the link candidates for the stops on the network.



4 Implementation in MATSim

A package to convert input schedule data such as GTFS, HAFAS or OSM to a completely mapped schedule has been developed in Java. The implementation allows to create a multimodal network from OSM. Tools to validate and edit the mapped schedule have been implemented as well. Figure 7 presents the workflow.

4.1 Creating an unmapped transit schedule

4.1.1 GTFS to MATSim transit schedule converter

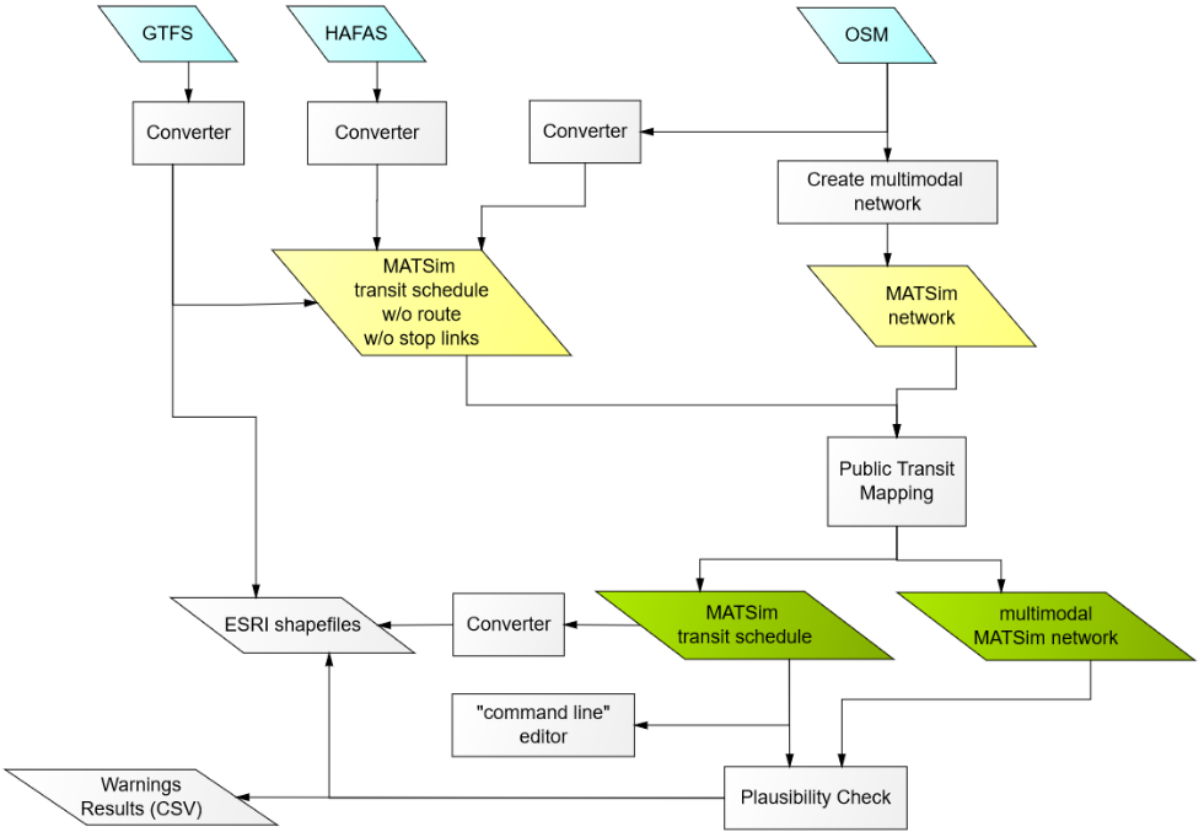
The implementation provides a script to convert a GTFS feed to an unmapped MATSim transit schedule. It is based on work by Ordonez and Erath (2011). While working on the converter, the MATSim contribution GTFS2MATSim has been released (Zilske and Kühnel, 2016). Since mapping and subsequent steps only need an unmapped schedule, it does not matter which converter is used. For future work, the more tested GTFS2MATSim converter is recommended.

The converter's main job is transforming GTFS trips to MATSim transit routes. Trips and transit routes are not equivalent: A trip defines the stop sequence and the departure time for a vehicle while a MATSim transit route defines the stop sequence but usually has more than one departure time. The time offsets between stops is stored in the `routeProfile` within the transit route in MATSim. GTFS offers two ways: Either with explicit time stamps in `stop_times.txt` or as frequencies in `frequencies.txt`. The converter can use both. Finally, routes in GTFS are like transit lines in MATSim. Stops are converted straight to stop facilities. GTFS offers no information that could be used as `isBlocking` value for MATSim stop facilities, so false is set by default.

For each trip, a service id defines on which days the trip is carried out. MATSim normally only simulates one day, therefore one reference day has to be extracted from the GTFS feed. Although possible, it is not recommended to use all trips in the feed. The following methods are suggested:

1. Define the date to extract all affected trips from it.
2. Use the day on which most services run.

Figure 7: The workflow to create a mapped transit schedule and the corresponding multimodal network from different data sources.



3. Use the day on which most trips are carried out.

The converter can create a default `transitVehicles.xml` file with one vehicle type per transport mode. If shapes for trips are specified in the feed, the trips used in the MATSim schedule can be converted to an ESRI shapefile (this is needed for the tests in Section 5).

4.1.2 HAFAS to MATSim transit schedule converter

The used HAFAS converter is based on Bösch and Ciari (2015). Stop facilities are created from the stops defined in the file `BFKOORD_GEO`. The transit lines, routes, departures and stop sequences are generated from `FPLAN`. The file `BETRIEB_DE` is used to add the agency's name to the transit line id. The reference day is extracted using `BITFELD`. For more details, see the documentation by Bösch and Ciari (2015). A default transport mode is assigned to each transit route depending on the vehicle defined in HAFAS. The transport modes correspond to one of the eight transport modes defined in GTFS (see Table 3). Optionally, a default vehicle file

is provided using predefined values[1] for different vehicle types (such as interregional trains, S-Bahn trains or trolley buses).

4.1.3 OSM to MATSim transit schedule converter

OSM offers tags to specify spatial public transit data on stop locations and transit routes. The converter provided as part of the package creates an unmapped transit schedule from OSM data. It creates stop facilities from OSM nodes with the tag `public_transport=stop_position`. Relations with the tag `route=*` are converted to transit routes. These transit routes contain only stop sequences. Link sequences are not converted even if they are available. The transport mode is set based on the respective `route=*` tag value of the relation.

The quality of the generated transit schedule for a region depends largely on the accuracy of the data in OSM. Often route data is either inconsistent or not even available. The lack of naming conventions further complicates using the data. In addition, OSM does not offer any tags to store temporal information. Thus, departure times and stop offsets have to be gathered from other sources.

4.2 Creating a multimodal network from OSM

The converter is based on the OSM network converter[2] available in the MATSim core. However, some changes have been made: The default values for links as well as parameters are stored in a `config.xml` file. This file allows to define whether an opposite link should be created and to set values for capacity, freespeed and the number of lanes for `highway=*` and `railway=*` tags. This also allows to convert rail links. In addition, ways that are only used by buses (i.e. are part of a relation tagged with `route=*` in OSM) are included in a network. If information on number of lanes or freespeed is available for an OSM way, it is used for the MATSim link. It is recommended to simplify the network by inserting only one link instead of multiple links between two intersections. The Euclidian distance between the new link's end nodes will less than the the summed up length of the replaced links but the original length can be stored in the length attribute of the new link.

[1] Default values are based on data provided by M. Rieser and P. Bösch.
[2] M. Rieser: `org.matsim.core.utils.io.OsmNetworkReader`

4.3 Public transit mapping implementation

The pseudo routing algorithm described in Section 3.3 has been implemented. All input parameters are defined in a config file (see appendix B). The whole schedule is mapped to the network. In cases where only subsets should be used (e.g. for a region), the schedule needs to be filtered beforehand.

4.3.1 Link candidates

First, a set of link candidates is created for each stop facility and schedule transport mode. For all nodes within `nodeSearchRadius` the in- and outlinks are fetched and sorted in ascending order by their distance from the stop facility. The other link candidates search parameters are defined for each schedule mode in the respective `linkCandidateCreator` parameterset. `maxNClosestLinks` defines how many links should be considered for each stop facility. This limit is not strictly enforced: When the limit is reached, the last link's opposite link is still added to the set. Further, after the limit has been reached, the distance of the farthest link to the stop facility is multiplied by `linkDistanceTolerance` and all additional links within this distance are added as well. This is used as a soft constraint to include links with almost the same distance from the stop facility. However, no links farther than `maxLinkCandidateDistance` from the stop facility are used.

The implementation allows to manually define link candidate beforehand in the config (in a `manualLinkCandidates` parameterset or a separate csv file (parameter `manualLinkCandidate CsvFile`, see Table 5 in appendix A.3 for an example). This helps mapping with complicated rail stations. Stop facilities with no link within `maxLinkCandidateDistance` are given a dummy loop link at their coordinates: A node is added at the coordinate and a dummy loop link is added to the network with the added node as source and destination. The loop link is referenced to the stop facility and is set as its only link candidate. `useArtificialLoopLink` defines if such an artificial loop link should be created regardless of the other parameters. Tram, subway, ferry, funicular and gondola routes are normally mapped with artificial links.

4.3.2 Creating mode dependent router

The config parameterset `modeRoutingAssignment` defines for a transport mode what links a transit route of this mode is allowed to use. For example, transit routes with schedule mode bus

can only use links with "bus" or "car" as modes. Similarly, all transit routes with the transport mode rail can only use rail links. If no assignment for a schedule transport mode is given, all transit routes using that mode are mapped artificially, with loop links for every stop facility and artificial links between them.

To calculate the least cost paths as part of pseudo routing, a router is needed for every transport mode. To create these routers, mode separated networks are generated. Then an A* router[3] for each of these networks is initialized. The networks are filtered according to the `modeRoutingAssignment`. The router uses one of two link travel costs: either `linkLength` or `travelTime`, defined in parameter `travelCostType`.

4.3.3 Pseudo routing

During this step the best sequence of link candidates for each transit route is calculated as described in Section 3.3. While routing on the network uses an A* router (Hart et al., 1968), least cost path search on the pseudo graph is done with a separate Dijkstra implementation (Dijkstra, 1959).

Artificial links connect the `toNode` of a link candidate with the `fromNode` of the next link candidate. Artificial links are added to the network in two cases: When no path on the network between two link candidates can be found or if the least cost path has costs greater than a threshold. This threshold is defined as `maxTravelCostFactor` times the minimal travel costs. The minimal travel costs depend on the parameter `travelCostType`: If it is `linkLength`, the beeline distance between the two stops is used. If it is `travelTime`, the minimal travel cost is equivalent to the travel time needed based on the arrival and departure offsets of the two stops. All artificial links (including loop links for stop facilities) and nodes have the prefix defined in `prefixArtificial`.

The step "PseudoRouting" creates `PseudoRoutes` for each transit route, each of which contains a sequence of `PseudoRouteStops`. A `PseudoRouteStop` contains information on the stop facility, the link candidate as well as departure and arrival offsets.

This pseudo routing step can be parallelized using multiple threads (`numOfThreads`). For each thread a queue of transit lines is handled. However, the search for the shortest path between link candidates uses the routing algorithms provided in the MATSim core which are not thread safe. Access to the mode separated routers had to be synchronized. Creating routers for each

[3] C. Dobler: `org.matsim.core.router.FastAStarLandmarks`

thread might increase performance but also causes substantial overhead. Additionally, some paths would likely be calculated more than once.

After this step, all artificial links are added to the network. These links have a very low freespeed and an increased link length. This prevents transit routes from using these links unless they have to. At this point, the link sequences for the transit routes have not yet been created.

4.3.4 Child stop facilities and route profiles

After all pseudoRoutes have been created, most likely there are multiple best links for a stop facility because different routes use different links. For each of these links a "child stop facility" is created. It has the same name and coordinate as its parent stop facility but it has a link referenced. The id of the child stop facility is generated by combining the parent stop id and the link id, connected by the string "`.link:`". For example the child stop facility of parent stop 64587 which is connected to the link 432 would get the id `64587.link:432`. Using the same connection string for all parts of the package allows to infer the parent stop id based on the given stop facility id.

Since it is now known which link candidates and thus child stop facility each transit route uses, route profiles (stop sequences) for all transit routes can be created.

4.3.5 Creating link sequences

Given the stop sequences where each stop facility is referenced to a link, the paths between those links are calculated for each transit route. Least cost path calculation is executed again because artificial links have been added to the network. Again, mode separated networks and A* routers are created. In theory, this step is not needed because the paths between the link candidates have already been calculated (Section 4.3.3) and could be stored. However, multiple threads might create artificial links and synchronized adding of links to the network was not implemented.

4.3.6 Pulling stop facilities together

It is possible that a link is used as a child stop facility reference link although the very next link in a link sequence is actually closer to the stop facility (and most likely was a link candidate as

well). The reason this happens is that the distance of the stop facility to the link candidate is not considered in the pseudo graph. This means that link candidates that are in sequence to each other are basically identical to the least cost path search algorithm. Which candidate is chosen then depends only on the internal order of link candidates and rounding of travel costs. This means that possibly too many child facilities are created. To solve this, each referenced link of a transit route is compared to is preceding and subsequent link. If one of the two is closer to the referenced stop facility, it is used as a child stop facility link. Since only three links are compared at once, this process is repeated until the closest link for every stop facility has been found.

It is certainly possible to include the stop facility distance in the pseudo graph edge weights. However, one might need to analyze possible effects on least cost path search since decreasing edge weights can lead to unexpected results during pseudo routing. This is the reason the approach using the stop facility distance was not pursued.

4.3.7 Cleaning schedule and network

Pseudo routing is finished after the previous steps. However, some artifacts remain in the schedule and the network. Since child stop facilities are created, it is likely that parent stop facilities and some child stop facilities (after pulling, Section 4.3.6) do not have any transit routes using them. By default, stop facilities that are not used by any transit route are removed (`removeNotUsedStopFacilities`). The length of artificial links is reset to the Euclidian distance. During pseudo routing, the freespeed of artificial links has been set very low (Section 4.3.3) and has to be increased again. It is set according to the schedule: For all transit routes the minimal necessary freespeed to ensure they are on schedule is calculated. The highest minimal freespeed of all transit routes of a link is used as the link's freespeed. This process can be done for other link transport modes as well (defined in `scheduleFreespeedModes`). It is recommended to do this for rail.

The transport mode of each transit route is assigned to its used links. Links that are not used by a transit route are removed. This can clean up and simplify rail networks. Links which have a mode defined in `modesToKeepOnCleanup` are kept regardless of public transit usage. Figures 8, 9 and 10 show examples of the completed mapping process. Fig. 11 shows line 33 in Zurich on the Hardbrücke, an example where the mapping did not work as desired. This probably happened because there are too many links on different levels around the stops which lead to a wrong selection of link candidates. It is also possible that the network has inconsistencies like missing links.

Figure 8: Mapping example for bus line 749 in Zurich.

(a) Before: Stop sequence connected with straight lines to visualize as input.

(b) After: The transit route is mapped to the street network.

4.3.8 Validation

Finally, the schedule is validated by the `TransitScheduleValidator`[4]. This checks if the link sequence can be driven along by a vehicle and if all stops appear in the correct order and have links referenced. A valid schedule means that it can be used for MATSim simulations.

[4] M. Rieser: `org.matsim.pt.utils.TransitScheduleValidator`

Figure 9: Mapping example for bus line 80 in Zurich.

(a) Before: Stop sequence connected with straight lines to visualize as input

(b) After: The path mapped to the network

Figure 10: Mapping example for a rail line: S5 from Pfäffikon to Zug.

(a) Before: Stop sequence connected with straight lines to visualize as input.

(b) After: The transit route is mapped to the rail network.

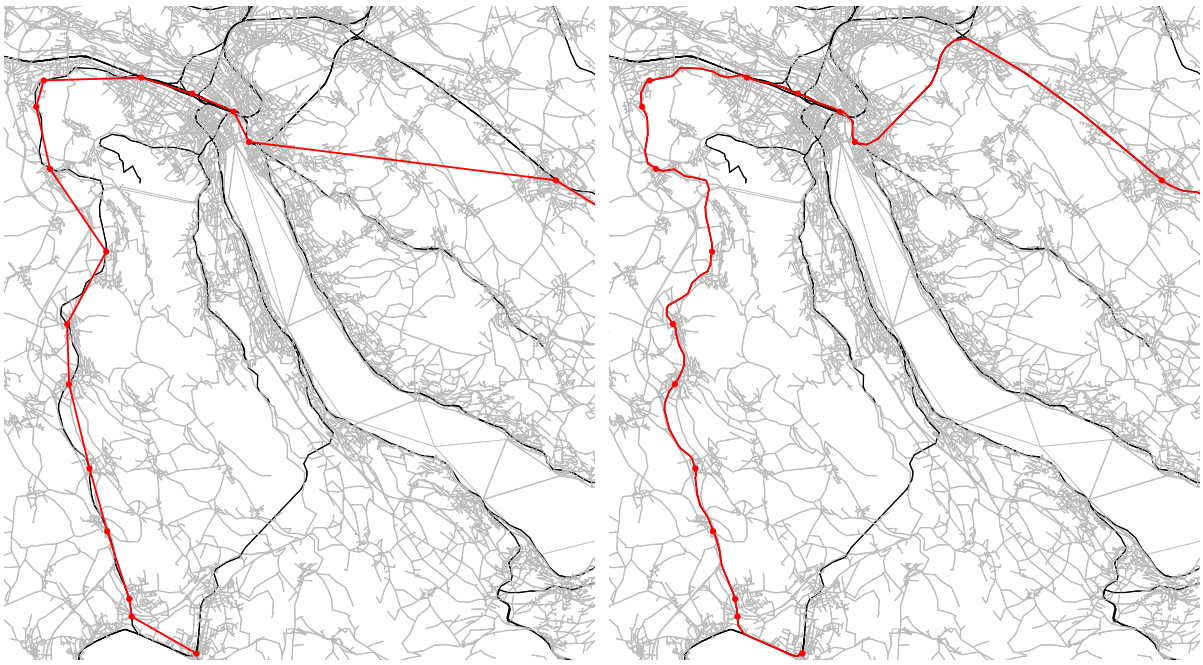
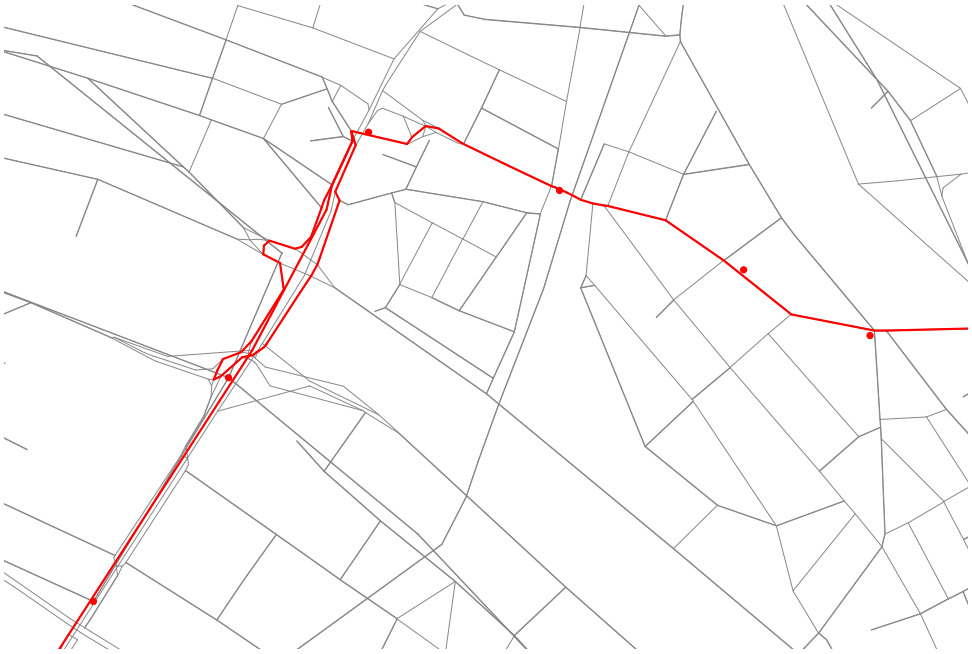


Figure 11: Example where mapping did not lead to the expected result (Line 33 in Zurich on the Hardbrücke).



4.4 Plausibility and editing

4.4.1 Plausibility check

While the `TransitScheduleValidator` is a good first indicator of the quality of a schedule, it does not check how plausible a transit route is. Possible implausibilities are for example if a route has loops, if sudden direction changes happen between links (u-turns) or if the travel times given by the schedule are cannot be met. If the travel time between two stops is higher than the scheduled time this points to either a too long (i.e. wrong) route or too low freespeed values on the network.

The package provides a plausibility check (class `PlausibilityCheck`, see appendix A.4) that looks for implausible parts of a route and warns accordingly. These warnings are written to a csv file as well as ESRI shapefiles.

4.4.2 Command line editor

A tool to edit MATSim networks and schedules is the JOSM-Plugin which is available as an external MATSim contribution (Kühnel and Zilske, 2016). However, using the plugin for large data sets (e.g. a network and schedule of the greater Zurich area) did not work due to memory and processor constraints. Another way to edit transit schedules is by hand using a text editor. This can become tedious for changing link sequences especially if multiple routes use similar link sequences. The package provides a way to edit large batches of a transit schedule via a csv file. The implementation provides methods to reroute parts of a transit route or change reference links for child stop facilities (see appendix A.5)

5 Analysis

The results of the implemented workflow have been validated by testing the accuracy of the mapping. It was also checked if a reasonable number of child stop facilities is created. As a reference, the mapping implementation by Bösch and Ciari (2015) has been tested as well.

5.1 Reference data

The tests have been been conducted with schedules based on a GTFS feed for the Zurich area (Offizielles Kursbuch: Fahrplandaten, 2016). The feed is provided by the Zürcher Verkehrsverbund (ZVV, Zurich traffic network) and covers all bus, tram, funicular and ferry routes of the Zurich area. There are two types of stops available: First, stop locations that are used by trips. Second, parent stops which also have point coordinates but are not accessed by any trips. Parent stops exist to group stop locations and are similar to the stops that are provided in the Swiss HAFAS data set. Fig. 12 shows an example from the feed for stop locations and parent stops. The feed also contains the shapes of the trips, i.e. a polyline representing the vehicle's path. These shapes can be used as a source to validate the schedule created by the pseudo routing algorithm. The shapes are available in a text file and can be converted to polylines. The feed has been converted to two unmapped MATSim transit schedules, one using stop locations as stop facilities and one using the parent stops as stop facilities. Table 4 contains descriptive information about the unmapped schedules. The unmapped schedules have been mapped to a network created from OSM[5]. Contrary to what is usually done, the links in the network were not simplified. To see differences to a previous approach, both schedules have also been mapped using the algorithm implemented by Bösch and Ciari (2015). Thus, four mapped schedules have been analysed.

5.2 Number of child stop facilities

The GTFS feed differentiates between parent stops and stop location. Thus, one can validate the approach of creating child stop facilities by comparing the original data with the result of the mapping. Fig. 13 shows the histogram of stops and the number of child stop facilities or the number of stops for the GTFS data respectively.

[5] Downloaded on 8.4.2016

Figure 12: Example for the stop locations and parent stops available in the GTFS feed for the zurich area.

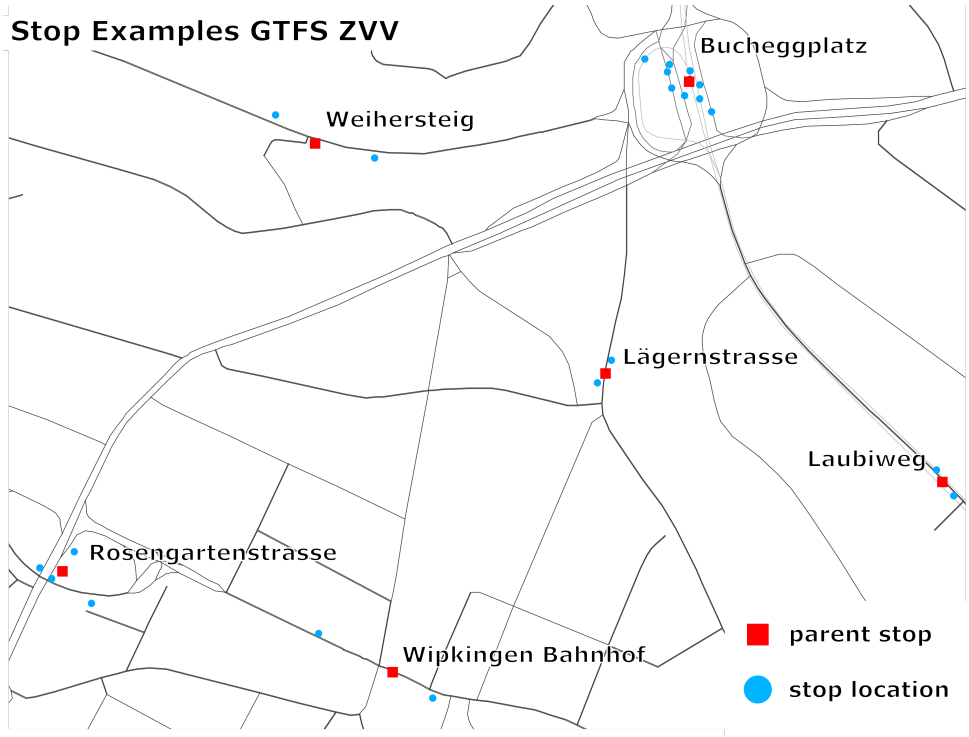


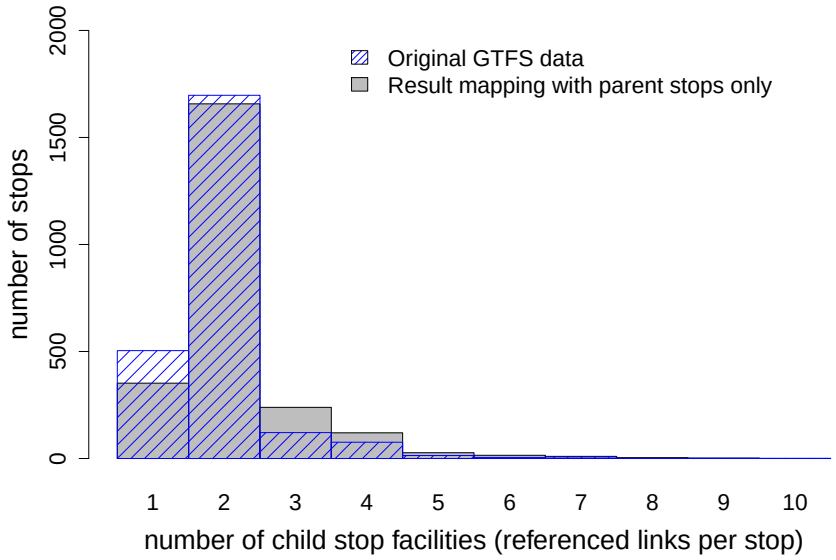
Table 4: Statistics on the unmapped schedules used for analysis.

	unmapped schedule *stop locations*	unmapped schedule *parent stops*
stop facilities	4'777	2'426
transit lines	346	346
transit routes	1'928	1'928

5.3 Mapping analysis

The second goal is to test the spatial validity and accuracy of the mapping by comparing the generated paths from the algorithm with the shapes given in the GTFS feed. The mapped schedule was converted to an ESRI shapefile, containing each route as a polyline. These polylines were compared to the trip polylines from the GTFS feed. Only bus routes were compared since train lines are not available and tram, ferry and funicular routes are normally mapped with artificial straight links between stops. Six transit routes have been removed because the GTFS shapes were incorrect (not covering the whole trip or straight lines from the first to the last stop).

Figure 13: The histograms show how many child stop facilities have been created per parent stop (grey) compared to the original data (blue).



In total, 1922 transit routes were compared to their corresponding GTFS shapes.

5.3.1 Maximal distance

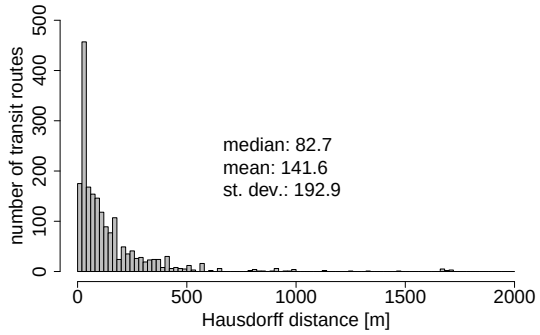
A standard way to compare two geometries is the Hausdorff distance (Huttenlocher et al., 1993) which represents the greatest of all the distances from a point in one line to the closest point in the other line. This single value represents the maximal distance between two lines. Fig. 14 shows the Hausdorff distance between the mapped path and the GTFS path for all four mapped schedules.

5.3.2 Accuracy score via buffer intersection

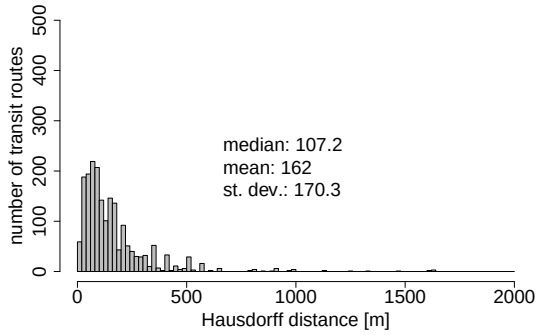
The Hausdorff distance does not actually quantify the similarity of two lines but is normally used to find the most similar line from a set of lines. When comparing the mapping result to the GTFS shapes, we are interested in how much of the mapped path is similar its GTFS counterpart. To further complicate the issue, even when the mapped path is perfect, differences are to be expected because the mapping result is based on a network from OSM which is different from the base map of the GTFS shapes. Thus, a buffer intersection test was applied. Each GTFS polyline was transformed to a polygon with a buffer of 5m or 10m. The polyline from the mapped MATSim

Figure 14: Histogram for Hausdorff distances between transit routes and GTFS shapes.

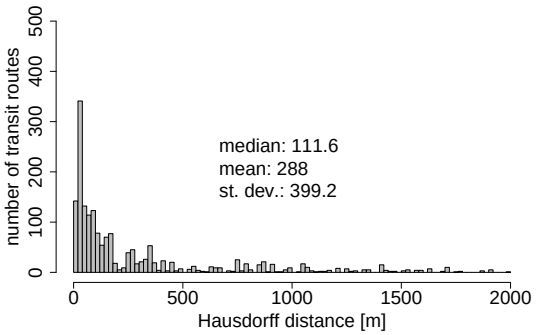
(a) Pseudo routing with stop locations



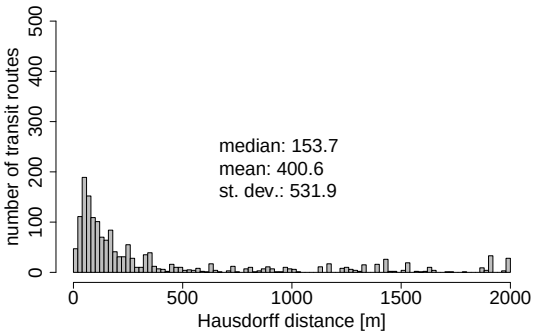
(b) Pseudo routing with parent stops



(c) Implementation Bösch and Ciari (2015) with stop locations



(d) Implementation Bösch and Ciari (2015) with parent stops



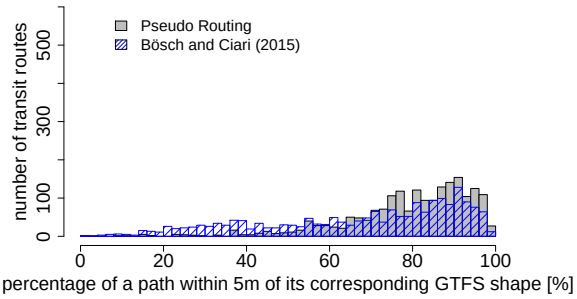
transit route was intersected with this buffer. This intersection defines the parts of a transit route that are within the buffer distance of the corresponding GTFS polyline. The accuracy score of a route is defined by the ratio of *mapped path length within buffer* and the *total mapped path length*. Figure 15 shows the score histograms for buffers of 5 and 10 meter.

5.3.3 Path length difference

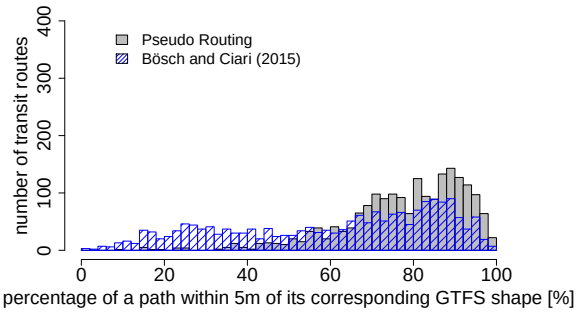
If a mapped transit route takes shortcuts, detours or has loops, its length will differ from the length of the GTFS line. The goal of this test is to compare the length of the GTFS path to the mapped path. Figure 16 shows the histograms for the differences between mapped path length and GTFS path length.

Figure 15: Histogram showing the number of transit routes for each accuracy score value.

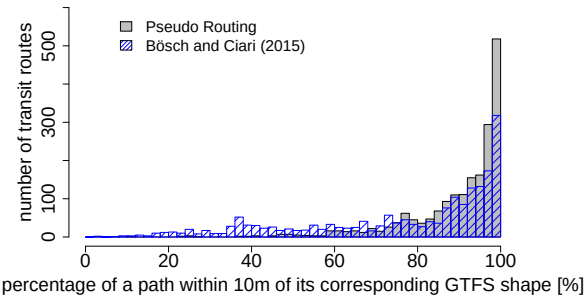
(a) Schedule with stop locations, buffer 5m



(b) Schedule with parent stops, buffer 5m



(c) Schedule with stop locations, buffer 10m



(d) Schedule with parent stops, buffer 10m

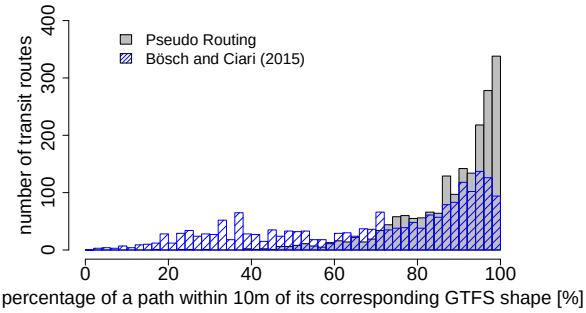
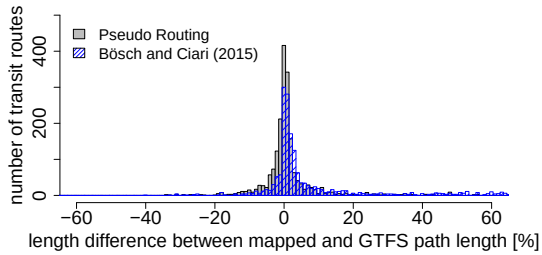
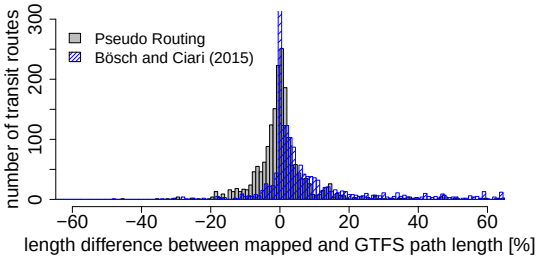


Figure 16: Histogram showing the number of transit routes for length differences in percent between the mapped transit path length and the length of the corresponding GTFS path.

(a) Mapping with stop locations



(b) Mapping with parent stops



6 Discussion

The test results in Section 5 show that the implemented pseudo routing algorithm does not create too many child stop facilities. The number of child stops for each parent stop is similar to the original GTFS data.

The mapping accuracy analysis shows that the overall majority of routes were mapped very similar to their corresponding GTFS shapes. The accuracy score for a buffer of 10 meters for both schedules (stop locations and parent stops) are satisfactory for an automatic algorithm. One should note that the test is rather simple and does not directly compare both lines but the mapped path with a general area where it should be. In addition, depending on the buffer distance, loops within the buffer cannot be detected. Manual analysis of paths with a low score showed that some were correctly mapped but had an offset due to the different map data sources. The Hausdorff distances for all four tested schedules are higher than expected. However, this distance represents the maximal distance between two lines and the accuracy scores are relatively good. The length difference between the mapped path and the GTFS shape is very small, again for both input schedules. The implemented pseudo routing algorithm tends to give better results than the implementation by Bösch and Ciari (2015) for both stop locations and parent stops.

Mapping results were not satisfying for routes whose stops are far apart from each other (e.g. express lines which skip some stops). If the network situation around a stop is complex (e.g. high number of links within a small radius or links on multiple levels), it is likely that not enough link candidates are selected. This leads to incorrect paths because the right link candidates were not part of the pseudo graph. In addition, the implementation might lead to unexpected results if there are too many stop facilities along a link, either because the density of stops is high or because the link is very long. Since the algorithm tries not to use the same link candidate for subsequent stops, this might lead to an invalid schedule. Finally, the mapping quality is largely dependent on a consistent and accurate network. If links are missing (especially bus lanes), the result is likely to be wrong.

Using travel time as travel costs tends to give better results than link length. However, when using a schedule based on Swiss HAFAS data, travel times might not work. The data only uses minutes for arrival and departure times which means that some stops have an offset of zero (i.e. no time difference between them).

7 Conclusion

7.1 Conclusion

This thesis proposes an algorithm to map a public transit schedule to a network. The pseudo routing algorithm calculates the path on a network for a transit route given its stop sequence. It calculates the least cost path from the first to the last stop with the constraint that the path must contain a link candidate from every stop. The algorithm has been implemented for MATSim. The test results show that the implementation leads to reasonable paths and that it is a viable way to automatically generate link sequences for transit routes. The package supports a multimodal approach, with bus routes mapped to streets and rail routes mapped to rail tracks. If no network links are available, artificial links are created.

The implemented Java package also contains tools to convert HAFAS, GTFS and OSM data to MATSim transit schedules. A multimodal network can be generated from OSM. Validation of the mapping process has shown that a majority of transit routes are mapped correctly. However, it is still necessary to check the resulting schedule manually. Tools to identify implausibilities in the schedule are provided with the package as well as a command line like editor to edit multiple link sequences at once.

7.2 Outlook

There are two suggested points to increase the quality of the mapping. Most problems with routes (loops or simply wrong routes) come from a wrong selection of link candidates. The proposed approach takes a number of links within a given radius. More complex link candidate search functions are conceivable, for example depending on the number of transit routes on a stop or on the type of stop. One could improve link candidate selection by including OSM data to order link candidates as proposed by geops (2014). Links that are close to a stop location identified in OSM could get a higher score. This does not even require to match data sets.

A second improvement step would be to develop more complex routers, thus improving mapping without changing the basic algorithm. The package allows two types of link travel costs: link length and travel time. Additional data like GPS could be included to calculate the travel cost. Links with GPS points next to them could have decreased travel costs. Again, OSM data could be included. For example, buses could have lower travel costs if they travel on links that are

tagged as bus routes in OSM. In addition, the network routers currently allow u-turns which should lead to a travel cost increase in further development steps.

Different MATSim simulation runs should be executed to further test and validate the implementation. First test runs have been conducted using the existing population from the IVT Baseline scenario (Bösch et al., 2016) and the Swiss HAFAS data (Offizielles Kursbuch: Fahrplandaten, 2016). However, due to time constraints, no analysis could be done so far.

OSM in general could provide much more information on networks even beyond public transit. For example, pedestrian and level crossings or traffic signal locations could be used to adapt link capacities or freespeed values.

Currently, data formats for public transit in MATSim cannot handle multiple stop facilities that belong to the same parent stop in a schedule. Even if the implementation presented in this thesis provides a workaround, the issue should probably be addressed on the core level. Additionally, the handling of transport modes by agents in MATSim is inconsistent and not intuitive. Transport modes defined for transit routes are ignored and all transit vehicles use the transport mode "car". However, the developers are aware of these issues. While converting GTFS data to a MATSim transit schedule, some information such as transfer times or fares cannot be converted and is lost. Last but not least, while working on this thesis, it became apparent that multiple groups work with different approaches to generate a mapped MATSim transit schedule from different sources. It might be useful to combine these efforts to a public transit pre-processing package.

8 References

Bösch, P., K. Müller and F. Ciari (2016) The IVT 2012 baseline scenario, paper presented at the *16th Swiss Transport Research Conference*, Ascona, April 2016.

Bösch, P. M. and F. Ciari (2015) A multi-modal network for MATSim, paper presented at the *15th Swiss Transport Research Conference*, Ascona, April 2015.

Brosi, P. (2014) Real-time movement visualization of public transit data, Master Thesis, University of Freiburg, Freiburg i. Br.

Bundesamt für Landestopografie swisstopo (2016) Vector25, webpage, June 2016, `http://www.swisstopo.admin.ch/internet/swisstopo/de/home/products/landscape/vector25.html`.

Dijkstra, E. W. (1959) A note on two problems in connexion with graphs, *Numerische mathematik*, **1** (1) 269–271.

geops (2014) Mapping public transit networks, webpage, December 2014, `http://geops.de/blog/mapping-public-transit-networks?language=en`.

Google (2016) What is GTFS?, webpage, June 2016, `https://developers.google.com/transit/gtfs/`.

HaCon (2016) Hafas, webpage, June 2016, `http://www.hacon.de/hafas/`.

Hansen, I. A. (ed.) (2010) *Timetable Planning and Information Quality*, WIT Press, Southampton.

Hart, P. E., N. J. Nilsson and B. Raphael (1968) A formal basis for the heuristic determination of minimum cost paths, *IEEE transactions on Systems Science and Cybernetics*, **4** (2) 100–107.

Horni, A., K. Nagel and K. W. Axhausen (eds.) (2016) *The Multi-Agent Transportation Simulation MATSim*, Ubiquity, London.

Huttenlocher, D. P., G. A. Klanderman and W. J. Rucklidge (1993) Comparing images using the hausdorff distance, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **15** (9) 850–863.

Kühnel, N. and M. Zilske (2016) A plug-in for josm, the openstreetmap editor, to save osm data in the matsim format, webpage, June 2016, `http://matsim.org/extension/josm-plugin`.

Nagel, K., M. Rieser and A. Horni (2016a) Generation of the initial MATSim input, in A. Horni, K. Nagel and K. W. Axhausen (eds.) *The Multi-Agent Transportation Simulation MATSim*, 67 – 69, Ubiquitiy, London.

Nagel, K., M. Rieser and A. Horni (2016b) MATSim data containers, in A. Horni, K. Nagel and K. W. Axhausen (eds.) *The Multi-Agent Transportation Simulation MATSim*, 62 – 66, Ubiquitiy, London.

Offizielles Kursbuch: Fahrplandaten (2016) Download der öffentlichen Fahrplansammlung der Schweiz, webpage, June 2016, `http://www.fahrplanfelder.ch/de/fahrplandaten.html`.

OpenStreetMap Wiki (2016a) About OpenStreetMap, webpage, June 2016, `http://wiki.openstreetmap.org/wiki/About_OpenStreetMap`.

OpenStreetMap Wiki (2016b) API, webpage, June 2016, `http://wiki.openstreetmap.org/w/index.php?title=API&oldid=1158215`.

Ordonez, S. and A. Erath (2011) Semi-automatic tool for map-matching bus routes on high-resolution navigation networks, *Technical Report*, Institut für Verkehrsplanung und Transportsysteme, Eidgenössische Technische Hochschule Zürich, Zürich.

PTV AG (2014) *PTV VISUM 14 Manual*, Karlsruhe.

PTV AG (2016) *Factsheet PTV Visum Modules Interfaces*, Karlsruhe, `http://vision-traffic.ptvgroup.com/fileadmin/files_ptvvision/Downloads_N/0_General/2_Products/1_PTV_Visum/EN_PTV_Visum_Modules.pdf`.

QGIS (2016) A free and open source geographic information system, webpage, June 2016, `http://qgis.org/en/site/`.

Quddus, M. A., W. Y. Ochieng and R. B. Noland (2007) Current map-matching algorithms for transport applications: State-of-the art and future research directions, *Transportation Research Part C: Emerging Technologies*, **15** (5) 312–328.

railML.org (2016) Partners, webpage, June 2016, `http://www.railml.org/en/introduction/partners.html`.

Rieser, M. (2010) Adding transit to an agent-based transportation simulation, Ph.D. Thesis, TU Berlin, Berlin.

Rieser, M. (2016a) Modeling public transport with MATSim, in A. Horni, K. Nagel and K. W. Axhausen (eds.) *The Multi-Agent Transportation Simulation MATSim*, 108 – 113, Ubiquitiy, London.

Rieser, M. (2016b) Transit vehicles all use network mode "car", webpage, June 2016, https://matsim.atlassian.net/browse/MATSIM-504.

SBB (2016) INFO+ HRDF-Exportschnittstelle 5.20.39, *Technical Report*, SBB.

senozon AG (2016) via – visualization and analysis tool, webpage, June 2016, http://via.senozon.com.

Talk-transit mailing list (2016) GTFS, tools and pt tags generally, webpage, June 2016, https://lists.openstreetmap.org/pipermail/talk-transit/2016-June/thread.html.

Trafimage Webkarten (2016) Vergleich Stationsdaten, webpage, June 2016, https://maps.trafimage.ch/#/ch.sbb.netzkarte?layers=ch.sbb.stationen.homog.hafas,ch.sbb.stationen.homog.osm,ch.sbb.stationen.homog.dfa,ch.sbb.stationen.homog.didok,ch.sbb.stationen.homog.haltestellen_oev.

Tran, K., S. Barbeau, E. Hillsman and M. A. Labrador (2013) Go_sync a framework to synchronize crowd-sourced mapping contributors from online communities and transit agency bus stop inventories, *International Journal of Intelligent Transportation Systems Research*, **11** (2) 54–64.

Transitland (2016) Feed registry, webpage, June 2016, https://transit.land/feed-registry/.

Zilske, M. and N. Kühnel (2016) GTFS2MATSim, webpage, June 2016, http://matsim.org/extension/gtfs2matsim.

Zilske, M., A. Neumann and K. Nagel (2011) OpenStreetMap for traffic simulation, paper presented at the *1st European State of the Map: OpenStreetMap conference*, 126–134, Wien.

A Implementation

A.1 Converters to MATSim transit schedule

A.1.1 GTFS to MATSim transit schedule

Main Class: `publicTransitMapping.gtfs.Gtfs2TransitSchedule`
The parameters can be set as arguments in the main function.

A.1.2 HAFAS to MATSim transit schedule

Main Class: `publicTransitMapping.hafas.Hafas2TransitSchedule`
The parameters can be set as arguments in the main function.

A.1.3 OSM to MATSim transit schedule

Main Class: `publicTransitMapping.osm.Osm2TransitSchedule`
The parameters can be set as arguments in the main function.

A.2 Create multimodal network from OSM

Main class: `publicTransitMapping.osm.Osm2MultimodalNetwork`
The converter needs a config file. A default config file can be created by running `CreateDefaultOsmConfig`.

Table 5: Example manual link candidates csv file. It can be set for the parameter `manualLinkCandidateCsvFile` in the public transit mapping config. Each line contains stopFacilityId, modes and linkIds. Multiple linkIds can be separated by comma.

```
879843;bus,tram;565,566,5489,5488,321,45
stop3238;bus,tram;893,894,900,901
station3135;rail;287,288,412,414,1129,1130
```

A.3 Public transit mapper

Main class: `publicTransitMapping.mapping.RunPublicTransitMapper`
The mapper implementation needs a PublicTransitMapperConfigGroup config. A default config can be created running `publicTransitMapping.config.CreateDefaultConfig`. An example config for Swiss HAFAS data can be seen in appendix B. It is possible to define manual link candidates, either as parameterset in the config or in a separate csv file. An example csv file can be seen in Table 5.

A.4 Plausibility check

Main class: `publicTransitMapping.plausibility.PlausibilityCheck`
The class performs a plausibility check on the given schedule and network. It checks for three implausibilities:

1. loops
2. travel time
3. direction changes

A coordinate system has to be given, it is recommended to use `EPSG=*` names since the created shapefile definitely works with those. The Swiss coordinate system LV1903+ is `EPSG:2056`. The following files are generated in the ouput folder:

- `allPlausibilityWarnings.csv` shows all plausibility warnings in a csv file
- `stopfacilities.csv` the number of child stop facilities for all stop facilities as csv
- `stopfacilities_histogram.png` a histogram as png showing the number of child stop facilities

- `shp/warnings/WarningsLoops.shp` Loop warnings as polyline shapefile
- `shp/warnings/WarningsTravelTime.shp` Travel time warnings as polyline shapefile
- `shp/warnings/WarningsDirectionChange.shp` Direction change warnings as polyline shapefile
- `shp/schedule/TransitRoutes.shp` Transit routes of the schedule as polyline shapefile
- `shp/schedule/StopFacilities.shp` Stop facilities as point shapefile
- `shp/schedule/StopFacilities_refLinks.shp` The stop facilities' reference links as polyline shapefile

Shapefiles can be viewed in a GIS (Geographic Information System), a recommended open source GIS is QGIS (QGIS, 2016). It is also possible to view them in senozon VIA (senozon AG, 2016). However, no line attributes can be displayed or viewed there.

A.5 Command line editor

Main class: `publicTransitMapping.editor.RunScheduleEditor`
The class loads a schedule and a network, then executes all commands in the commands csv file. The first cell each a line denotes the method to be applied. All subsequent cells denote the arguments for the method. The file has to use ";" as delimiter. An example csv file can be seen in Table 6. Alternatively `publicTransitMapping.editor.BasicScheduleEditor` can be used to call methods within Java. The following methods are available:

rerouteViaLink `(transitLineId, TransitRouteId, oldLinkId, newLinkId)`
Reroutes the section between two stops that passes the oldLink via the newLink.

rerouteFromStop `(transitLineId, TransitRouteId, fromStopId, newLinkId)`
Reroutes the section after fromStop via the given newLink.

changeRefLink `(stopFacilityId, newlinkId)`
Changes the referenced link of a stop facility for all routes. A new child stop facility with the reference link is created if it does not already exist.

changeRefLink `(transitLineId, transitRouteId, parentId, newlinkId)`
Changes the referenced link of the stop facility in the transit route (given by transitLineId and transitRouteId) which contains the `parentId` in its id.

changeRefLink `("allTransitRoutesOnLink", linkId, parentId, newlinkId)`
If a stop facility should be re-referenced for multiple transit routes, one can use the command "allTransitRoutesOnLink" with a linkId to select all those transitRoutes.

addLink `(newLinkId, fromNodeId, toNodeId, attributeLinkId)`
Adds a link to the network. Uses the attributes (freespeed, nr of lanes, transportModes) of the attributeLink.

refreshTransitRoute `(transitLineId, transitRouteId)`
Recalculates the link sequence of the given transitRoute.

After each step the schedule is "refreshed" and the link sequences of the changed transit routes are recalculated. Comments can be used by adding "//" at the start of the line.

Table 6: Example command line editor file.

```
// TransitLineId;TransitRouteId;oldLinkId;newLinkId
rerouteViaLink;line4;route41;3443;3464

// TransitLineId;TransitRouteId;fromStopId;newLinkId
rerouteFromStop;line4;route41;8975613;3464

// StopFacilityId;newlinkId
changeRefLink;8975613.link:3435;3464

// TransitLineId;TransitRouteId;ParentId;newlinkId
changeRefLink;line4;route41;3464

// linkId;ParentId;newlinkId
changeRefLink;allTransitRoutesOnLink;295341;8580522;586310
```

B Public transit mapping example config

Figure 17: Example config for Public Transit Mapper (part 1).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v2.dtd">
<config>
<module name="PublicTransitMapping" >

  <param name="networkFile" value="network/mm/switzerland.xml.gz" />
  <param name="scheduleFile" value="mts/fromHafas/ch.xml.gz" />

  <param name="outputNetworkFile" value="../output/ch_network.xml.gz" />
  <param name="outputScheduleFile" value="../output/ch_schedule.xml.gz" />

  <param name="numOfThreads" value="4" />

  <param name="nodeSearchRadius" value="1500.0" />
  <param name="travelCostType" value="linkLength" />
  <param name="maxTravelCostFactor" value="5.0" />

  <param name="scheduleFreespeedModes" value="rail,light_rail" />
  <param name="modesToKeepOnCleanUp" value="car" />
  <param name="addPtMode" value="true" />
  <param name="combinePtModes" value="false" />
  <param name="prefixArtificial" value="pt_" />

  <parameterset type="modeRoutingAssignment">
    <param name="scheduleMode" value="bus" />
    <param name="networkModes" value="car,bus" />
  </parameterset>
  <parameterset type="modeRoutingAssignment">
    <param name="scheduleMode" value="rail" />
    <param name="networkModes" value="rail,light_rail" />
  </parameterset>
```

Figure 18: Example config for Public Transit Mapper (part 2).

```
  <parameterset type="linkCandidateCreator" >
    <param name="linkDistanceTolerance" value="2" />
    <param name="maxLinkCandidateDistance" value="60.0" />
    <param name="maxNClosestLinks" value="6" />
    <param name="scheduleMode" value="bus" />
    <param name="networkModes" value="bus,car" />
  </parameterset>

  <parameterset type="linkCandidateCreator" >
    <param name="scheduleMode" value="rail" />
    <param name="linkDistanceTolerance" value="1.0" />
    <param name="maxLinkCandidateDistance" value="300.0" />
    <param name="maxNClosestLinks" value="35" />
    <param name="networkModes" value="rail,light_rail" />
  </parameterset>

  <parameterset type="linkCandidateCreator" >
    <param name="scheduleMode" value="tram" />
    <param name="useArtificialLoopLink" value="true" />
  </parameterset>

  <parameterset type="linkCandidateCreator" >
    <param name="scheduleMode" value="gondola" />
    <param name="useArtificialLoopLink" value="true" />
  </parameterset>

  <parameterset type="linkCandidateCreator" >
    <param name="scheduleMode" value="funicular" />
    <param name="useArtificialLoopLink" value="true" />
  </parameterset>

  <parameterset type="linkCandidateCreator" >
    <param name="scheduleMode" value="ferry" />
    <param name="useArtificialLoopLink" value="true" />
  </parameterset>

  <parameterset type="linkCandidateCreator" >
    <param name="scheduleMode" value="subway" />
    <param name="useArtificialLoopLink" value="true" />
  </parameterset>

</module>
</config>
```

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Public Transit Mapping on Multi-Modal Networks in MATSim

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Poletti	Flavio

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 4.7.16	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.