Free University of Bozen/Bolzano
Faculty of Computer Science

Fakultät für Informatik
Facoltà di Scienze e Tecnologie informatiche
Faculty of Computer Science

Bachelor Thesis

A Time-Dependent Data Model for Multi-Modal Networks

by
Luca Bellettati

Supervisor: Johann Gamper
March 20th, 2015

Contents

Abstract

The concept of multi-modal spatial network is becoming very popular as the availability of data is exponentially increasing, and new web and mobile technologies are using them to build nice solutions for route searching or trip planning. Most of the times, the problem is how to model such data in a way that many types of networks can be used efficiently for a desired solution. In fact, multi-modal network data are of different type: pedestrian, bus, train, railway, bicycle, etc. all with different features. In particular, there are some of them, like bus or railway, that hold a time table which are to be considered when building a system.

In our work, we propose a time-dependent data model for multi-modal networks that build a representation of the information based on the different features of each mode inside the network. To achieve this, we clearly separate each mode from the others in order to obtain a detailed structure for that specific graph. To create the final multi-modal network, we introduce the concept of *links*, that connects each sub-graph without modifying the structure of the existing modes by using switch points inside the graph. Furthermore, the idea of separation is applied to public transportation system modes to create sub-networks based on routes. For such graphs, the time table is kept in a separated structure as well.

We provide a detailed description of our solution by describing the structure of each network together with an import tool that stores instances of real data into a PostgreSQL database, comparing the new model with the existing one.

Riassunto Breve

Il concetto di reti multi modali è diventato sempre più popolare e usato, grazie alla crescente disponibilità di dati, e alla tecnologie web e mobile che usano queste numerevoli informazioni per sviluppare soluzioni software riguardanti il *route searching* e *trip planning*. Nella quasi totalità dei casi, il problema maggiore è causato da come questi dati possono essere modellati e rappresentati. Ciò è dovuto al fatto che una rete multi modale contiene al suo interno diversi tipi di grafi: pedonale, ciclabile, bus, treni, etc. Alcuni di essi sono associati ad una lista di orari, che deve essere presa in considerazione nel momento in cui si vuole costruire un sistema che usi queste informazioni.

Nel nostro studio, proponiamo un modello di dati *time-dependent* che rappresenti i dati di una rete multi modale in base alla caratteristiche di ogni singolo grafo. Per ottenere tale modello, noi proponiamo di separare ogni singolo *mode*, in modo tale da avere una struttura dettagliata per ognuno di essi. Inoltre, per le reti che contengono diversi percorsi, come ad esempio i bus o i treni, viene applicata un'ulteriore separazione in base a questi percorsi, ottenendo dei sottografi. Un struttura aggiuntiva è usata per salvare gli orari per i *modes* nei quali sono presenti. La rete finale contenente tutte le altre, viene creata attraverso il concetto di *links*, i quali connettono sia i diversi *modes* tra di loro, sia i diversi sottografi, ove presenti. Tutto ciò avviene senza modificare la struttura iniziale.

In questo report, forniamo una descrizione dettagliata del nostro modello insieme ad una procedura che salva dati reali in un database PostgreSQL, comparando infine il nuovo modello con quello già esistente.

Kurzfassung

Als Folge des exponentiellen Wachstums der Verfügbarkeit von Informationen ist das Konzept des *multi-modal spatial network* sehr bekannt geworden. Neue web und mobile Technologien benutzen diese um Routensuche oder Tourenplanung Lösungen zu entwickeln. Diese müssen sich mit der vielfältigkeit der Netzwerkarten auseinandersetzen und dafür sorgen, dass die Datenverarbeitung in Beziehung der Netzwerke zu erfolgreichen Ergebnissen führt. *Multi-modal network* Daten teilen sich in unterschiedlichen Arten ein und besitzen unterschiedliche Eigenschaften: Fußgänger, Busse, Züge, Fahrräder usw. Im Falle der Busse und Züge muss bei der Entwicklung und bei dem Bau dieser insbesondere auf den Zeitplan geachtet werden.

Im Folgenden wird ein auf den unterschiedlichen Eigenschaften der einzigen Modalitäten des *multi-modal networks* basiertes Zeitabhängiges Darstellungsmodell angeboten. Um das Ziel zu erreichen wird jede Modalität einzeln graphisch dargestellt. Anschließend wird das *multi-modal network* mit der Einführung des Konzepts der *links* dargestellt. Das letztgenannte Konzept verknüpft jedes Sub-Graph ohne seine Struktur zu verändern indem er *switch points* im jeweiligen Graph. Dieses Trennungskonzept wird auch am System öffentlicher Verkehrsmitteln angewandt um auf Routen basierte Sub-Netzwerke zu entwickeln. Auch für ein solches Netzwerk wird der Zeitplan auf einer anderen Ebene gespeichert.

Durch die Implementierung der entwickelten Lösung dank echter Daten wird es möglich, eine detaillierte Beschreibung der Struktur unserer Lösung zu liefern. Zum Schluss, wird ein Vergleich zwischen den neue und den existierende Modell angezeigt.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Johann Gamper for his incredible support during the past months, in particular for his willingness and his encouraging suggestions that I received while working on this thesis.
I would like to mention and thank Markus Windegger, Kevin Wellezohn and Paolo Bolzoni for the small but important help they provided me at the beginning of my work.
I will never forget the memorable moments with Michael, Emanuela, Sara, Irene, Verena and Laura while working for SCUB and Snowdays: dinners, party, and movie nights that we enjoyed together are the best moments I had during my bachelor career.
I have met lots of people at UniBz and I cannot mentioned everybody, but I cannot forget Alex, Simone, David and Beatriz, my girlfriend, for their support in these years, in particular in the last months, while I was writing this thesis, and for the special moments we had during our university life.
I would like to thank also all the friends I still have at home, in particular Matteo, Anna, don Roberto, don Stefano, don Francesco and don Luciano because they were part of the most important moment of my life, always supporting me with the best advices. We did a lot together and looking back now, I am suprised to see the person I was and what I became thanks to all the experiences we enjoy together. In particular I want, to thank Matteo because he introduced me to the world of computer some years ago, and make me discover what I really like.
I cannot forget to mention Pater Otto for his happiness, his willingness and his ideas that made and make my time at Haus St. Benedikt an opportunity to know nice people and spend beautiful moments with them.
Last but not least, my family, my parent Eugenio and Maria Carla, my syster Irene, my uncle Marco and my grandmother Cosetta: you encourage me, you support me, and you were always available for me. All the experiences, all the bad and good moments would have not been possible without you and your sacrifices.

Chapter 1

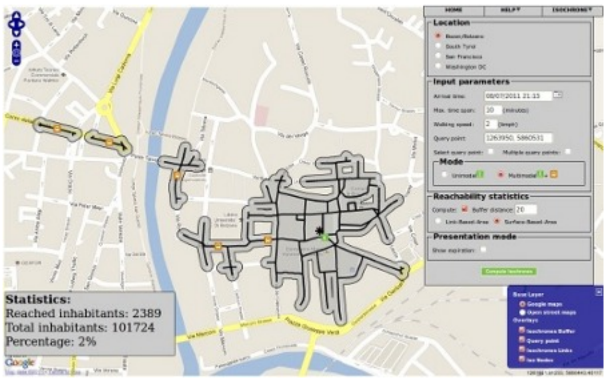
Introduction

1.1 Isochrones

The idea behind this thesis is based on the contributions of Innerebner et al. in [16] and [15], where they introduce the concept of *isochrone*. As defined in [15], given a multi-modal spatial graph and a query point *p*, an *isochrone* is the minimal sub-graph derived by merging all the paths from *p* to all reachable locations that are within a given time span, using any type of available networks (street, bus, train, bicyle, etc.).
To show the result of their theoretical studies, Innrebner et al. built a system called ISOGA, available at [4], which apply isochrones to some cities around the world.
Figure 1.1 displays the result of a query from the point "*". The panel on the right side of the picture contains the parameters the algorithms use to calculate the desired *isochrone*: time interval, preferred mode (street, bus, etc.) city data set. As a result of the calculation the system display the grey-coloured areas on the map, with some icons that denotes the mode used to reach a specific point, and some statistic about the covered portion of surface.
The data sets of ISOGA are stored in PostgreSQL database with PostGIS extension. The information saved are about street, bus and train networks as well as *points of interest*, like museums or restaurant, and buildings.

1.2 Motivation

During the maintenance of the database of [4] (see section 1.1), we discovered that the model used is not suitable to perform periodic efficient updates. The existing solution is a time-dependent model for multi-modal spatial networks that stores the data of every mode inside one graph, using shared points to connect the different modes. This is not a good solution since the data come from different sources and, in particular, have several structures.



Arrival time: 08/07/2011 21:15
Max. time span: 10 (minutes)
Walking speed: 2 (km/h)
Query point: 1263950, 5860531
Compute: Buffer distance: 20

Statistics:
Reached inhabitants: 2389
Total inhabitants: 101724
Percentage: 2%

Figure 1.1: ISOGA.

1.3 Contribution of the Thesis

We propose a new time-dependent data model that take into account the differences between various modes of a multi-modal network, in order to have a data set as real and efficient as possible. Our solution clearly separates each graph, in order to store only the required information for each mode, introducing *links* to connect all the network in a single final one. As a result of our work we have:

- a time-dependent data model for multi-modal spatial networks
- a data import tool to store in a PostgreSQL database a real implementation of the new model inside Linux systems
- a theoretical discussion on our solution
- a set of procedures to import the data using the old time-dependent model of ISOGA

1.4 Road map

In Chapter 2, we report references to previous works proposing solutions for modelling multi-modal networks. In Chapter 3, we describe our solution by specifying the data model as well as the database schema. In Chapter 4, we report the work we have done to implement a simple script that create and populate a database with the described model. In Chapter 5, we report a comparison between old and new solution. Finally, we summarize our work and we report future works that can be applied to the current version of the model.

Chapter 2

Related Work

In this chapter we briefly report the most important works about the topic of data modelling for multi-modal networks, in order to have a solid starting point for the next chapters, where we will explain in details our contribution and the results.

A simple version of *time-expanded* model is presented by Bast et al. [11] as a base for path discovering in public transportation networks. The solution uses three kinds of nodes, *departure node*, *arrival node*, and *transfer node* to implement the idea of multi-modality with mode change during a trip. This tries to minimize the number of transfer from mode to mode and from different routes of the same mode to achieve the so called *fast-direct connection* between a given source and destination points.
Booth et al. [12] described a detailed *time-dependent* data model for multi-modal transportation system, including the different networks occurring in a real world, together with some facilities related to each of them, like grocery, fast food, museum and other points of interest. Shared nodes are used instead of edges to connect different modes, i.e. to transfer from one network to another.
A distinction between private and public transport networks is presented by Zhang et al. [22] while proposing a data model which take into account multiple attributes like dynamic travel and time tables. In their work, a distinction is done between private and public transportation networks. The first one contains only physical nodes, i.e. real ones like bus stops, train or subway stations; on the contrary, the second one is described by both physical (real) and event (virtual) nodes, to consider also time table options at a given node. They also stress about the need of an explicit transfer edge or link in the network in order to change modality during a trip, where it is possible.
Mahrous [19] describes and discuss the challenges of multi-modal network modelling and proposes to clearly separate modes networks, creating one

dataset for each of them. What is proposing here is to perform an horizontal separation in order to avoid overlapping of network inside a dataset. Furthermore, he noticed that also inside bus mode the sub-networks of the routes are overlapping: he also proposed to vertically separate bus network by assigning the routes number to each line segments, i.e. to each edge of the network. Note that the vertical separation can be done in other type of networks where different routes are present. As transfer strategy between modes, he also proposed explicit transfer edges in the graph.
The work of Kircher [17] takes into account the *condensed model* where no time table information are present but only the physical structure of the network. He underlines: first, how the *time-expanded* model explicits the time table itself compared to the *time-dependent* version; second, the importance of transfer edges in order to have the model as real as possible.
Antsfeld and Walsh [10] [9] proposed a combination of *time-expanded* and *time-dependent* model by distinguishing two layers in multi-modal networks: the *station grap*h and the *events graph*. The first one contains *station links*, i.e. connections via a transport system, and *walking links*, i.e. street connections; the second one stores the information about events using three type of nodes (*arrival*, *departure*, *transfer*), and four type of links (*deaparture*, *continue*, *changing*, *waiting*). In addition, it used the concept of *hub nodes* to denote stations where several transport system edges from different nodes are entering. In this way *non-hub nodes* are discarded when searching for mode change.
Another different way of representing public transportation data was proposed by Liu [18], who used the concept of *switch point* to denote the place where a change of mode occurs. In this solution the cost of mode transfer is not associated with an edge but with a point, which also contains all the information about the modes entering and/or exiting to/from it, like time tables.
Delling et al. [14] concentrated on the optimization part of a *time-expanded* network. They discovered that the contraction technique, which simply bypasses nodes during path search, has a dramatic growth in number of edges. The solution they proposed is based on the assumption that if we found a path towards a given station A with arrival time t_a, there is no need to search for path that arrive at A later than t_a. Then, only a few number of transfer edges from arrival to departure nodes have to be added: it avoids to maintain the unuseful connections with certain transfer nodes.
An interesting solution specific for flight networks has been proposed in detail by Pajor [20]. He states that the structure of such a network is different in the real world compared to other types like, street, train or bus one. Hence, he developed a suitable model with different alternatives taking into account the characteristic of flight network: check-in, check-out, transfer action, actual flight time and class of flight (domestic or abroad). In addition, he proposed a general framework including time function that

handles the features described above.
Most of the work described in this thesis is based on the content of [21] and [13], where Pyrga, Delling et al. presents and compare their solution for *time-expanded* and *time-dependent* data model. They describe in detail how each proposed model is structured and how it behaves with the two main problem regarding transportation networks with time table, i.e. the *earliest arrival problem* and the *minimum number of transfer problem*. Since they take into account both simple and realistic version, the concepts of traffic days and link edges are added to handle respectively different time tables for different days and to explicitly connect the various modes inside the final multi-modal network. In particular, in [13] they proposed:

- an optimization for public transportation network by introducing the concept of *access nodes,* which allows to prune part of the search space by entering one network and stay on it until it is possible to avoid too many changes which would raise up the time

- a detailed description about how to use *traffic days* to check validity of the time table to built a realistic version of the network

Chapter 3

A Time-Dependent Data Model

The next pages will report in details the time-dependent data model that we study and develop. In section 3.1, the new data model is presented and described. In Section 3.2, the database schema of a real application of the new model is reported. In Section 3.3, we provide examples tables entries of the model.

3.1 Data Model

The main characteristic of the new data model is the separation of different modes to have a clear representation for each of them. To obtain this, we define a different structure for each different network included in the multi-modal graph. We consider basically two types: *street network* and *timetable networks*, like bus, train or tram networks.

3.1.1 Street Network

We define a street network as a directed graph G_s = (V_s, E_s), where V_s is the set of vertices or nodes, and E_s is the set of edges connecting pairs of nodes. Each node represents a physical point in the world where two or more streets intersect forming a crossroad. A street or part of it in the real world is stored as edge object, with source and destination nodes. This denotes the directed connection between a specified source point and its corresponding destination point. The reverse connection is expressed by a different edge object where source and destination are swapped. Nodes that are connected by at least one edge are said to have a neighbourhood relationship. For example, a node sn_j is said to be a neighbour of another node sn_k, if there exists an edge from sn_k to sn_j. The reverse is also true. Furthermor, the number of incoming and outgoing edges to and from a node sn_j represent respectively the *in-degree* and the *out-degree* of sn_j, which are essential information for isochrone calculation.

Figure 3.1 shows an instance of street network. Nodes sn_8, sn_9, sn_{10}, sn_3 are examples of street nodes, connected by pairs of street edges, one denoting the road segment for one direction, the other denoting the same road segment but in the reverse direction. Let us take node sn_8: it is directly connected to sn_{10}, sn_3 and sn_9. This means that it has three outgoing edges, respectively se_{24}, se_{19}, se_{21}, and three incoming edges, respectively se_{23}, se_{20}, se_{22}. Then, it has *in-degree* equal to three and an *out-degree* still equals to three.

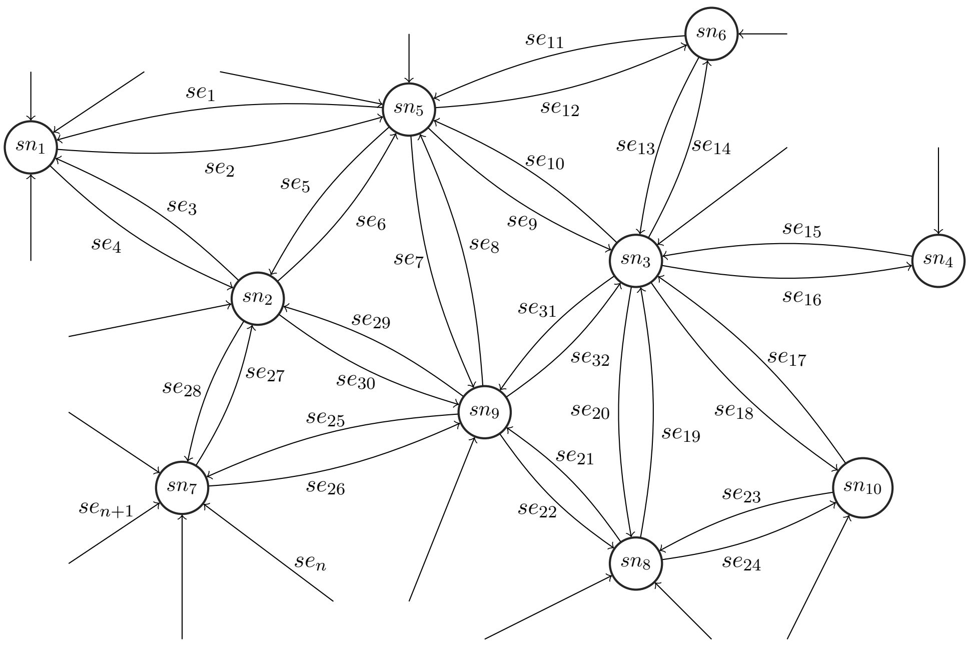


Figure 3.1: Street network.

3.1.2 Timetable Network

A time table network has an underlay graph G_t = (V_t, E_t), where V_t and E_t are respectively the set of nodes and edges. Each node tn_j in V_t denotes a stop or station in the real world and is connected to other nodes through an edge te_k in E_t. The different routes are considered to further separate the data. A sub-graph G_t^i = (V_t^i , E_t^i) is created for each route *i* over the graph G_t, where a node tn_j^i in V_t^i denotes a stop or station in real world for route *i*. Then, there is one node representing the same physical point in the real world for each different route using that stop or station, and there exists only one edge te_k^i in E_t^i that connects two consequent nodes tn_j^i and tn_m^i of route *i*.
In Figure 3.2, tn_1^5, tn_5^3, and tn_2^{10} represent all $station_1$ respectively for route 5, 3 and 10. In the same way, tn_3^3, tn_4^5, and tn_6^9 represent $station_2$ respec-

tively for route 3, 5 and 9. Since $station_1$ and $station_2$ are consequent nodes and both have route 5 and 3, there exists an edge te_1^5 to connect tn_1^5 and tn_4^5 for route 5, and another edge te_2^3 from tn_5^3 to tn_3^3 for route 3. Nodes tn_2^{10} and tn_6^9 are not connected since they represent two different routes, but there exist an edge te_m^{10} from tn_2^{10} to another node tn_j^{10} of $station_k$ and an edge te_n^9 from a node tn_h^9 of $station_l$ to tn_6^9.
The timetable is added to all G_t^i by associating to each edge te_j^i of route *i* a pair of time points. Each of such connections is referred to as trip sequence ts_k^i that store the departure time $time_d$ at source node tn_s^i and the arrival time $time_a$ at destination node tn_t^i. Then, to build a complete trip from the start node ts_{start}^i to the final node ts_{end}^i the corresponding trips sequences are grouped using two more attributes: the id of the trip ts_{id} and the a number ts_{seq} denoting the position of a specific connection ts_j^i inside the set of trip sequences with same id ts_{id}. This means that for an edge te_j^i of route *i* there is a trip sequence ts_k^i for each time connection over te_j^i. A further attribute is added to a trip sequence ts_k^i to denote its validity, i.e. on which days it actually runs. Then, a traffic day vector v_n is associated to ts_k^i which contains a sequence of 0s and 1s: a 1 at position *p* in v_n means that on day of the year *p*, ts_k^i is valid, i.e. runs; a 0, instead, means that it is not valid for day *p*. Thanks to v_n we have two advantages:

- the timetable is not equal for all days, i.e. it is more realistic
- we do not have to store the date for each ts_k^i, so that only one copy of it is actually stored and not a copy for all the day on which it is valid and running

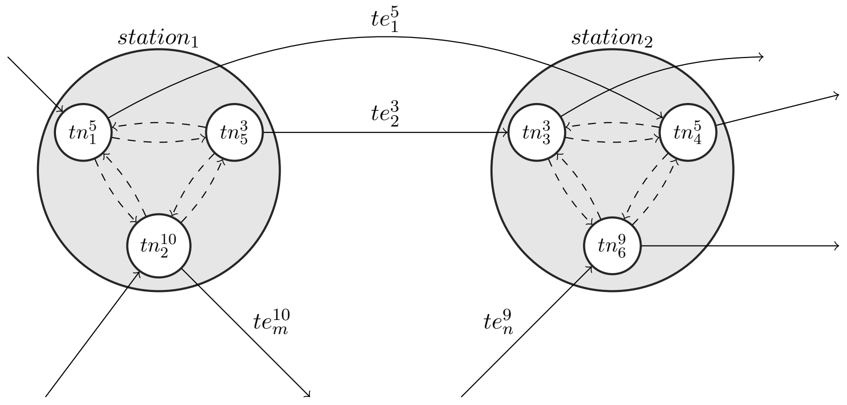


Figure 3.2: Timetable network.

3.1.3 Link Network

The final multi-modal network is constructed by connecting the two graphs G_s and G_t thorugh *link* edges, that represent the switching points between two modes. To obtain all the connections, there must be a mapping of the timetable network(s) on the street one, since there is no correspondence between nodes of G_s and G_t. This requires to further add new nodes to the street network which are the common shared point between G_s and G_t. For each pairs of different nodes (sn_j, tn_k^i), there will be one link l_v = *(*sn_j*,* tn_k^i*)* and one link l_u = *(*tn_k^i*,* sn_j*)*, both with zero cost, that denote a switch in the two directions: from street to timetable and vice versa.
In Figure 3.3, the complete multi-modal network is represented including G_s, G_t, and *link* edges (boldface lines). The nodes sn_1, sn_2 and sn_3 are all street nodes of G_s, but only sn_2 is a pure street node: sn_1 and sn_3 are added to G_s while mapping G_t on G_s, i.e. when $station_1$ is mapped to sn_1 and $station_2$ to sn_3. As consequence, each node of $station_1$ and $station_2$ are connected respectively to sn_1 and sn_3 with a pair of *link* edges: tn_1^5 has one incoming edge l_{24} and one outgoing edge l_{23} from and to ts_1, tn_3^3 has one incoming edge l_{19} and one outgoing edge l_{20} edge from and to ts_1, and so on for the remaining couples of nodes of G_s and G_t. In addition, all nodes of $station_1$ are connected through *link* edges l_1, l_2, l_3, etc.; the same is for all nodes $station_2$.

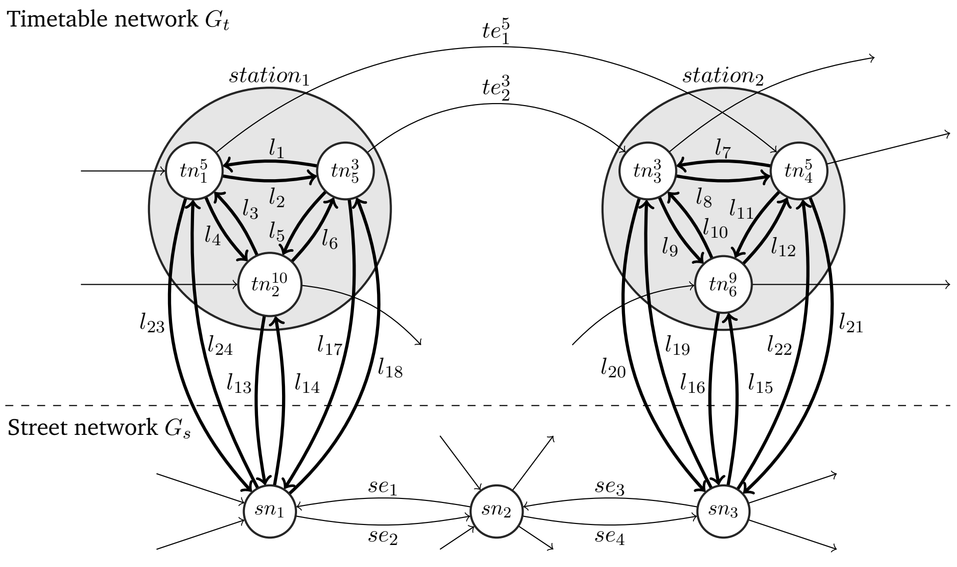


Figure 3.3: Multi-modal network with link edges in boldface.

When all *link* edges are added between G_s and G_t, the network is complete. Then, Figure 3.3 is an example of the final multi-modal network obtained by applying our time-dependent data model.

3.2 Database Schema

This paragraph is dedicated to described how the new data model is implemented in a database, by reporting the correspondent ER schema for both street and timetable network.

3.2.1 Street Network Schema

The street network as described in Section 3.1.1 is imported into a database by using one table for nodes and one for edges.
The data for street nodes are saved in the a relation with schema:

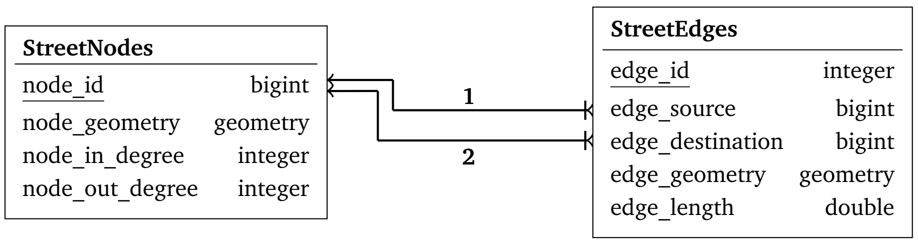
StreetNodes$(\underline{NodeID}, NodeGeom, NodeInDegree, NodeOutDegree)$.

Each entry in this table is uniquely identify by the *primary key* constraint on the *NodeID* field, *NodeGeom* contains the spatial information for the node, i.e. longitude, latitude and altitude values, *NodeInDeg* and *NodeOutDeg* store respectively the number of incoming and outgoing edges for the node, needed during the calculation of isochrones.
The street edges are defined in the database according to the following schema:

StreetEdges$(\underline{EdgeID}, EdgeSource, EdgeDest, EdgeGeom, EdgeLength)$

where *EdgeID* is the unique identifier for the tuple, *EdgeSource* and *EdgeDest* are the extremes of the street segment, *EdgeGeom* store spatial information for the line shape, and *EdgeLength* is the actual length in meters of the line. A *foreign key* constraint is applied to *EdgeSource* and *EdgeDest* with a reference to *NodeID* field of ***StreetNode*** relation. In fact, if *EdgeSource* or *EdgeDest* or both contains values that are not in *NodeID* field of ***StreetNodes***, the edge cannot exist as entry of ***StreetEdge***.
In Figure 3.4 we represent the two relations ***StreetNodes*** and ***StreetEdges*** as ER-diagram, including all the fields, their data type and the constraints previously described. In particular, we use the *bigint* type for *node_id*, *edge_source* and *edge_destination* because the source we use has big values for identifier and we keep such numbers in order to maintain the source and destination references for edges. This is not applied to *edge_id* field which is defined as an integer. Two are the reasons: first, there are no further references to identifier values of an edge; second, when *link* edges are built, the initial street connections are update, the identifier as well, and the values are not kept in any case. Then, starting a new enumeration for

this field is more convenient to not create confusion when adding new entries or during maintenance works. We use *double* type for *edge_length* to preserve precision on the measure of the segment. We denote *primary key* constraint by underlining *node_id* and *edge_id*; 1 and 2 represent the *foreign key* constraints respectively on *edge_source* and *edge_destination*.



Legend:
1: edge_source_fk
2: edge_destination_fk

Figure 3.4: Relational schema for street network.

3.2.2 Timetable Network Schema

The timetable network we describe in Section 3.1.2 includes relation for nodes and edges and three more to store the schedule.
The information for timetable nodes are save in a schema similar to ***StreetNodes*** with an additional field for the route. The relation is as follows:

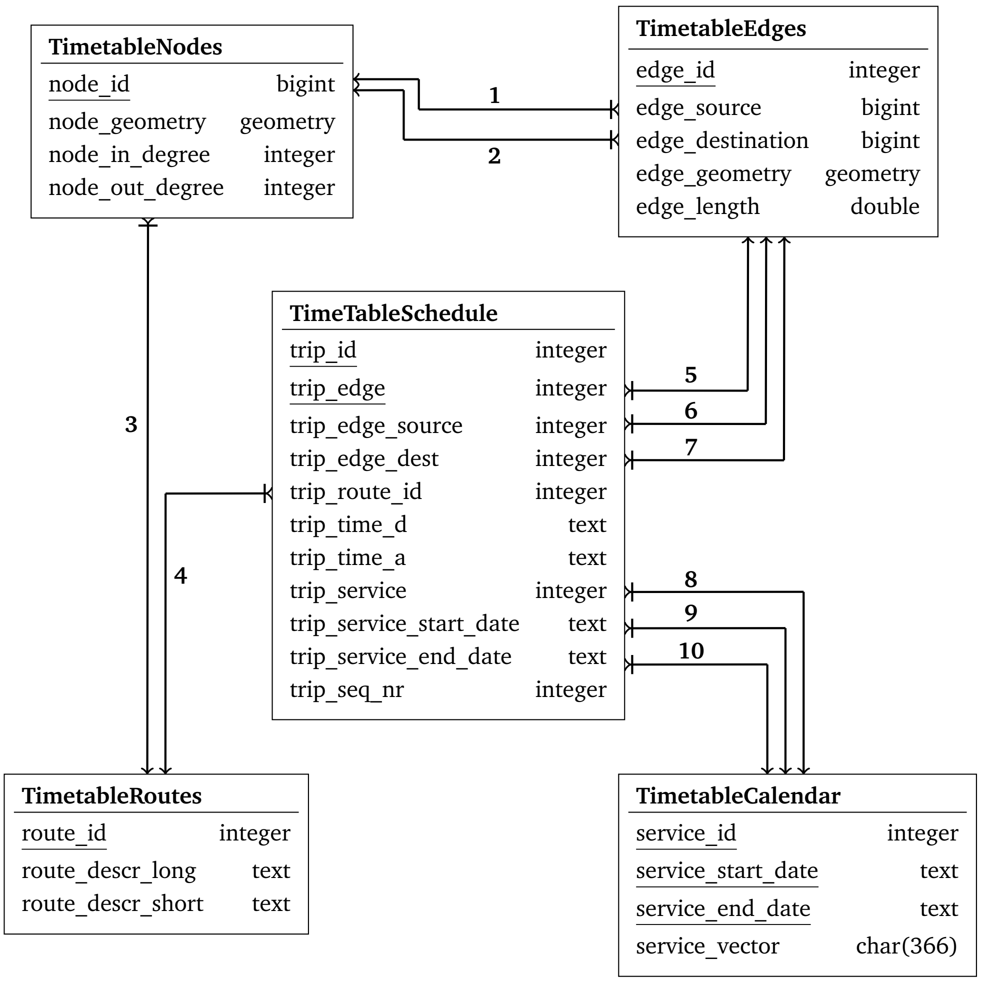
$$\textbf{\textit{TimetableNodes}}(\underline{NodeID}, NodeGeom, NodeRoute, NodeInDegree, NodeOutDegree)$$

The unique identifier is still *NodeID*, *NodeGeom* can have same value for several nodes, *NodeRoute* is a numeric value for the route associated to the node, *NodeInDegree* and *NodeOutDegree* represent respectively incoming and outgoing edges, both *link* and *timetable* edges. A *primary key* constraint is used to ensure the uniqueness of *NodeID*.
The schema for edges has less information with respect to ***StreetEdges*** relation. Its definition is:

$$\textbf{\textit{TimetableEdges}}(\underline{EdgeID}, EdgeSource, EdgeDest, EdgeRoute)$$

There are no attributes for geometry and length, but there is a field for the route. *EdgeID* is an unique value that identify the single connection between *EdgeSource* and *EdgeDest* for the route indicated in *EdgeRoute*. Since each edge represent a connection between two stop or station for a specific route, the values of *NodeRoute* of the corresponding entry in ***TimetableNodes*** for

EdgeSource and *EdgeDest* must be equal to the value in field *EdgeRoute* of this tuple. To ensure correctness of the data constraints are used: a *primary key* on *EdgeID* field and *foreign key* on *EdgeSource* and *EdgeDest* to ensure that such identifiers exist in ***TimetableNodes***.



Legend:

1: edge_source_fk	**5**: trip_edge_fk	**9**: trip_service_s_date
2: edge_destination_fk	**6**: trip_edge_source_fk	**10**: trip_service_e_date
3: node_route_fk	**7**: trip_edge_dest_fk	
4: trip_route_id_fk	**8**: trip_service_fk	

Figure 3.5: Relational schema for timetable network.

A simple schema is used for information about routes. It is as follows:

$$\textbf{\textit{TimetableRoutes}}(\underline{RouteID}, RouteDescrLong, RouteDescrShort)$$

The values for *RouteID* are unique, then a *primary key* constraint is applied on this field; *RouteDescrLong* and *RouteDescrShort* are string value to store respectively the long and short version of the name use to denote a route in real world.
The calendar containing the validity periods for trip sequences is store in this form:

$$\textbf{\textit{TimetableCalendar}}(\underline{ServiceID}, ServiceStartDate, \\ ServiceEndDate, ServiceVector)$$

To uniquely identify an entry we use a *primary key* constraint on *ServiceID*, *ServiceStartDate*, and *ServiceEndDate*, since same value for *ServiceID* can have different of *ServiceStartDate, ServiceEndDate*, and *ServiceVector*. This is a structures solution to keep exeception days, like Christmas or Easter, separated from the regular schedule. The value for *ServiceID* is a simple integer, *ServiceStartDate* and *ServiceEndDate* are string representing a date as "YYYYMMDD", *ServiceVector* is a sequence of 366 0s or 1s, each representing a day of the year. For example, if at position 31 we have a 1, it means that on the 1^{st} of February (beacause the enumeration starts from 0, not from 1) all the associated trip sequences of ***TimetableTripSchedule*** are valid.
The whole schedule for the timetable network is contained in one relation, whose schema is:

$$\textbf{\textit{TimetableTripSchedule}}(\underline{TripID}, \underline{TripEdge}, TripEdgeSource,) \\ TripEdgeDest, TripRouteID, TripTimeDep, TripTimeArr, \\ TripService, TripServiceStartDate, TripServiceEndDate, TripSeqNr)$$

To uniquely identify an entry we use both *TripID* and *TripEdge*. We cannot only use *TripID* since one tuple denotes just a piece of the trip and the same identifier must be used for the other segments. There is, instead, only one entry with a specific pair of *TripId* and *TripEdge*. *TripEdgeSource* and *TripEdgeDestination* seems to be not useful in this table but they are crucial at query time to avoid a join with ***TimetableEdge*** to find edge extremes. In fact, a typical query on ***TimetableTripSequences*** could be: give me all the connections exiting node tn_j^i after a specific time point given by $time_d$ on day *d*. Having the edge extremes in ***TimetableTripSchedule*** as well avoids the join of this table with ***TimetableEdges***, because the query will search all the entries of ***TimetableTripSequences*** whose *TripEdgeSource* is equal to tn_j^i and whose *TripTimeDep* is after $time_d$. The only join that needs to be performed is with ***TimetableCalendar*** to check that the service vector corresponding to *TripService* has a one in the position for day *d*. Of course, the tuples that are not valid on the specified day *d* are not included in the query

result. *TripRouteID* associates a trip sequence, and in turn a trip, to a route in ***TimetableRoute***. *TripSeqNr* is fundamental to build the complete chain of trip sequences from the beginning of a trip to the end: number 1 in this fields of an entry means that *TripEdgeSource* is the station where the trip starts, any number $n > 1$ denotes that *TripEdgeSource* is the n_{th} station, any other number $m > 1$, such that for any n in the chain of this trip $m > n$, means that *TripEdgeDest* is the terminus of the trip, i.e. the last station.
Figure 3.5 represents the whole schema of a timetable network, including all the *primary key* (underlined fields) and *foreign key*, this latters reported in the Legenda of the picture.

3.2.3 Link Network Schema

The schema for link network is very simple since it just creates pairs of connected node between the entries of ***StreetNodes*** and ***TimetableNodes***. Each tuple is of the form:

$$\textbf{\textit{Links}}(\underline{LinkID}, LinkSource, LinkSourceMode, \\ LinkDest, LinkDestMode)$$

LinkID is the unique identifier with a *primary key* constraint on it; *LinkSource* and *LinkDest* are respectively the source and destination extremes of the *link* edge; *LinkSourceMode* and *LinkDestMode* denote respectively the mode of the source and destination: 1 means that the correspondent node belongs to ***StreetNodes***, 0 that it is part of ***TimetableNodes***. We do not use a *foreign key* constraint on *LinkSource* and *LinkDest* since both can have an identifier coming from two or more different tables and a double references on a table field nor a *check* constraint with sub-queries cannot be done.
Figure 3.6 report the ER-model for *link* edges, which is a stand-alone table for the reasons we have just explained in the previous paragraph regarding the *foreign keys* constraints. In particular, we define *link_source* and *link_destination* as *bigint* type to fit both ***StreetNodes*** and ***TimetableNodes*** identifier.

Links	
<u>link_id</u>	integer
link_source	bigint
link_source_mode	integer
link_destination	bigint
link_destination_mode	integer

Figure 3.6: Relational schema for links.

3.3 Tables Data Examples

In this section we report some pictures about real table data for our data model as well as an explanation of the information displayed.

Street Nodes. In Figure 3.7 are data about ***street_nodes*** table. As you can see, values for *node_id* are large and can be even larger for other rows: for this reason we choose *bigint* type for this fields. The geometry is stored a Extended Well-Known Text (EWKT), one of the format used by PostGIS to represent point and line shape with alphanumeric values. *node_in_degree* and *node_out_degree* are simple integer and are always equals since the number of incoming and outgoing edges are the same.

row_id	node_id	node_geometry	node_in_degree	node_out_degree
1	9192415	0100123ED749...	2	2
2	29541604	0101000020E4...	1	1
3	60958599	0101030040B5...	3	3
...	...	...	...	...

Figure 3.7: Data example for street_nodes table.

Street Edges. Once we have the points of the network, we use them to build the edges. Figure 3.8 gives an idea of some rows store in the ***street_edges*** table. We note that *edge_id* contains smaller values with respect to *node_id*, since we do not keep the values of the source, but we restart the numeration, as already said in Section 3.2.1. Under *edge_length* field are *double precision* values with 12 decimal digit, obtained from the PostGIS function *ST_Length()* applied over *edge_geometry* attribute.

row_id	edge_id	edge_source	edge_destination	edge_geometry	edge_length
...	...	...	...	...	...
10	35	334643920	359085948	01200020E6...	173.102640511226
11	49814	3352099002	3352098999	0120549E08...	35.392863527394
12	24103	3259420147	3126879985	0120309F07...	64.896363827364
...	...	...	...	...	...

Figure 3.8: Data example for street_edges table.

Timetable Nodes. Points for timetable network are saved in the database similar to the street ones. In addition they have the *node_route* field, which references values in ***timetable_routes*** relation. Furthermore, *node_in_degree* and *node_out_degree* are still equal and are always equal to 1, as you can see in Figure 3.9. It could be the case that a variant of a route exists; then, there is a timetable node where the alternative starts: at that location the *node_out_degree* will be two or even more if more than one alternative path is present there.

row_id	node_id	node_geometry	node_route	node_in_degree	node_out_degree
...	...	...	...	...	...
89	54	01000A5F...	201	1	1
90	423	01002D8C...	5000	1	1
91	1035	0112E0F...	5	1	1
...	...	...	...	...	...

Figure 3.9: Data example for timetable_nodes.

Timetable Edges. The table for edges is lighter for a timetable network. In Figure 3.10 we have an example of such a table, where no fields for geometry and length are present. In addition we have the *edge_route* attribute to denotes the route the edge belongs to. For example, row 200 store edge 843 for route 6 that start at node 741 and ends at node 231. Also the next row store an edge for route 6, with extremes 25 and 54.

row_id	edge_id	edge_source	edge_destination	edge_route
...	...	...	...	...
200	2569	874	1547	11
201	843	741	231	6
202	1789	25	54	6
...	...	...	...	...

Figure 3.10: Data example for timetable_edges.

Timetable Route. Figure 3.11 shows some rows of ***timetable_routes***. The information in this table are few and for our model only the *route_id* is actually needed. *route_descr_long* and *route_descr_short* are useful for user interface purposes and we decided to keep them since our model referred to ISOGA, which need to show these information together with the computation results.

row_id	route_id	route_descr_long	route_descr_short
...	...	...	...
43	110	110 BZ	110
44	211	211 ME	21
45	1071	7A BZ	7A
...	...	...	...

Figure 3.11: Data example for timetable_routes.

Timetable Calendar. An example of ***timetable_calendar*** is Figure 3.12. As you can see, row 20 and 21 has the same *route_id* 1. This means that this specific service is valid for the time interval of both row 20 and row 21. Actually, the information that we need from this table is the value in the *service_vector* field, since it already contains the validity for a specific day *d*, denoted by the position *d* in the vector.

row_id	service_id	service_start_date	service_end_date	service_vector
...	...	...	...	...
20	1	20141224	20141224	0000000000000010000...
21	1	20141225	20141225	0000000000000001000...
22	8	20151229	20150106	000001001100100110...
...	...	...	...	...

Figure 3.12: Data example for timetable_calendar.

Timetable Schedule. The core of the timetable network is in ***timetable_schedule*** relation. Here we store each single connection at each different time in the schedule. This means that there will be more than one trip running on the same edge with different values for $time_d$ and $time_a$. For example, edge 529 is used by trip 15865 at 13:20:00 and by trip 9854 at 10:25:00 respectively with validity identifier 8 and 1. Furthermore, row 1000 is an example of a trip start point, since the *seq_nr* is 1, and row 1001 is its consequent edge, because the trip identifier is the same and the *seq_nr* is 2. This is how timetable is stored in the database: simple direct connections from that starts at *edge_s* at $time_d$ and arrives to *edge_d* at $time_a$.

row_id	trip_id	edge	edge_s	edge_d	edge_r	time_d	time_a	service	...	...	seq_nr
1000	15865	529	489	488	112	13:20:00	13:20:00	8	...	...	1
1001	15865	528	488	479	112	13:20:00	13:21:00	8	...	...	2
...	...	...	...	...	...	...	...	...	...	...	...
3567	9854	529	489	488	112	10:25:00	10:26:00	1	...	...	7
3568	9854	474	81	130	112	10:30:00	10:30:00	1	...	...	10
...	...	...	...	...	...	...	...	...	...	...	...
4010	5412	621	25	300	2	18:55:00	18:56:00	15	...	...	20
4011	2014	1254	789	900	2	07:14:00	07:15:00	5	...	...	10
...	...	...	...	...	...	...	...	...	...	...	...

Figure 3.13: Data example for timetable_schedule.

Links. The last table of the schema is ***links*** which is the most important in order to construct the final multi-modal network. In each entry, a *node_id* of ***street_ndoes*** is associated with one of ***timetable_nodes*** to build the specific *link* edge.
As Figure 3.14 shows, row 100 define the edge between node 899 with

mode 0, i.e. a timetable node, and node 3352098263 with mode 1, i.e. a street node. Since the mode change can happen in both directions, row 101 store another *link* edge where *link_source* is 3352098263 with mode 1, and *link_destination* is 899 with mode 0. Furthermore, *link* edges can also connect two different timetable nodes of different routes at the same station. This is the case of row 104 and row 105, where the bidirectional connection between node 101 and node 99 is expressed. In this situation, *link_source_mode* and *link_destination_mode* are both 0.

row_id	node_id	node_geometry	node_route	node_in_degree	node_out_degree
...	...	...	...	...	...
100	88	899	0	3352098263	1
101	89	3352098263	1	899	0
102	90	3215478953	0	992	1
103	91	992	1	3215478953	0
104	92	101	0	99	0
105	93	99	0	101	0
...	...	...	...	...	...

Figure 3.14: Data for links table.

Chapter 4

Implementation of a Data Import Tool

This chapter report the work done to implement the model described in the previous chapter. First of all, the sources for street and bus data are introduced; then, the created SQL, Java, and Bash script are briefly presented and discussed together with the actual process for importing the data.

4.1 Street Data

The complete network for streets is based on the public accessible data of OpenStreetMap, available at [6] in some different format like .pbf, .bz2, or shape file. For convenience, in our implementation we used the PBF format, which is based on Google Protocol Buffer, a mechanism to serialize structured data in order to read/write them from/to several streams using different languages. The serialization phase is performed using a so called .proto file, where the data are define as *messages*, a logical record which contains a list of key-value pairs, denoting the attributes of the object described by that specific record. In order to use the defined protocol buffer messages to parse a file with high-level languages like Python, Java or C++, a compiler is needed to create specific functions to access the fields as well as read/write raw bytes that store the actual data. A complete documentation as well as usage examples are available at [3]. Note that we do not build our own compiler, but we use an existing one available at [7].
Each .pbf file is organized in blocks containing both meta-data and actual data. In our implementation, we have to deal only with data block, which can contain the following objects:

- **node**: a physical point in real world representing entities like bus stops, street crosses, points of building perimeter, etc.

- **way**: ordered list of not more than 2000 nodes representing railways, street, footpaths, etc.
- **relation**: ordered list of nodes, ways and relations representing an entity like administrative boundaries or routes
- **tag**: sub-element common to nodes, ways and relations that store additional information as key-value pairs

More details about the actual parsing and data usage are presented in section 4.3.2.

4.2 Timetable Network

The implementation of the time table network part is based on the data coming from SASA, the local agency of the Province of Bozen-Bolzano responsible for urban bus system. These data are available at [5] and are represented following the data model created by the Verband Deutscher Verkehrsunternehmen (VDV) which offer a complete framework to handle many types of networks with time table. As this data model is very articulated and sometimes difficult to understand, due to the high number of entity it holds, we decided to switch to the General Transit Feed Specification representation, which has been developed by Google to give an interface for mapping public transportation systems into Google Maps database. This model is simpler than the one proposed by VDV and it can be quickly obtained through a tools developed by Google itself that convert files in VDV format to actual GTFS structure. As a result we obtain the following .txt files:

- **agency**: list of agency appearing in the data set, in our case only one entry for SASA
- **calendar and calendar_dates**: traffic days with exceptions
- **routes**: all the routes in the data set
- **stops**: a complete list of all bus stops
- **trips**: all the trips for the available routes with traffic day
- **stops_times**: the list of all simple connection between two stops for a given trip, with related arrival and departure times

In this way, we have a model that is more similar to the one that we implement. The big difference is that GTFS does not store a connection as an arc going from the source to the destination, but it stores the fact that at some node an arrival event is happening at arrival time, which will be followed by the departing event at departure time at the same node. As fig. 4.1 shows, in the GTFS solution there are no explicit arc that describe a direct connection between node n_{10} and noden_{20}: this has to be discovered by retrieving two different tuples. In our solution, this is not necessary because the arc is explicitly stored in one tuple. To have a complete overview about VDV and GTFS please refers respectively to [8] and [1].

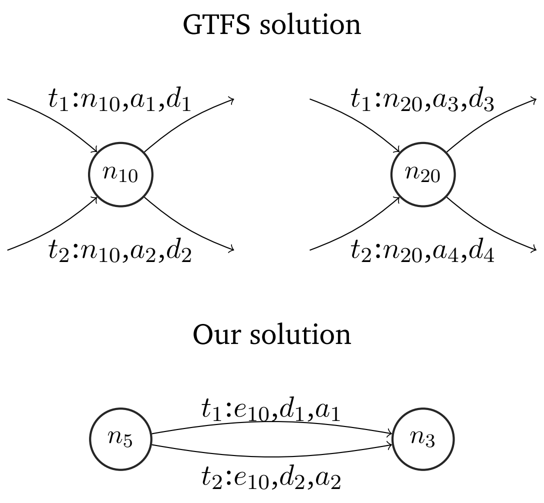

GTFS solution
t_1:n_{10},a_1,d_1
n_{10}
t_2:n_{10},a_2,d_2
t_1:n_{20},a_3,d_3
n_{20}
t_2:n_{20},a_4,d_4
Our solution
t_1:e_{10},d_1,a_1
n_5
n_3
t_2:e_{10},d_2,a_2

Figure 4.1: Implicit and explicit arc for trip sequence.

4.3 Data Import Tool

The import process to build our model has been built and tested under Linux Ubuntu 14.04 using Java 8 SE, PostgreSQL 9.3 and PostGIS 2.1. The whole flow of work is managed by bash scripts where Java, SQL and shell commands are executed to construct the desired networks. The complete work flow consists of an initialization phase, where the environment is set up, followed by an import phase, where the desired data are retrieved and stored in a database.

4.3.1 Initialization

As one is using the code for the first time, a pre-processing phase has to take place to set up the database environment with the required schemas and tables. This process create:

- a PostgreSQL user called *spatial*
- a PostgreSQL database called *spatial*
- a schema *time-dependent* that contains the implementation of our model
- a schema *time-dependent-old*, for testing purposes
- a schema *vdv-gts-tmp* to manage the import of time table networks
- directories to store temporal data

The whole initialization phase is managed by a shell script called *setup.sh*, where *psql* commands as well as bash commands are executed in order to create the items listed above. In Figure 4.2, we show the setup steps: first the database and the user are created by executing *db-create.sql*; then, *time-dependent*, *time-dependent-old* and *vdv-gts-tmp* are created thorugh *schema-create.sql*.
Most of the work will be concentrated on the *time-dependent* schema, as it implements our data model. However, one can also decide to import the old version of the model to *time-dependent-old* schema. This option is offered for testing purposes to compare the behaviour of some algorithms with same data modelled in different manner.

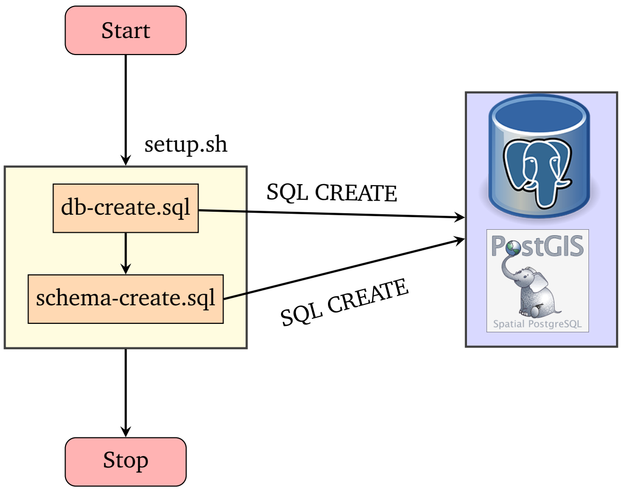


Figure 4.2: The initialization work flow.

4.3.2 Data Import

Once the setup phase has successfully terminated, the actual import process can take place by running the root script *net-builder.sh*, which create each network (street, bus, link, or test) separately, depending on the user choice. Note that we provide a list of cities for which the data are available; in the future,the idea is to offer also the possibility to the user to input the coordinates of the bounding box of a city, in order to import a specific data set not included in the list. Unfortunately, for bus network the data are more difficult to retrieve, due to the several agencies that manage local public transportation system even within a region.
Figure 4.3, shows the complete work flow for the data import. As already said, the whole process is controlled by the root bash script called *net-builder.sh*, which execute other bash files depending on the options that are used.
For street network, the option *-s* must be present; one can decide to download the latest version of the data from Openstreetmap by using the option *-u* in addition to *-s*, or start the parsing from already downloaded local data, if available. In case, the requested data are not available locally and the *-u* is not present, the script will automatically download them from [2]; then the bounding box for the desired city is obtain using a Java tool called *osmosis*, which return an OSM file; at the end, *osmconvert* transform the OSM file into a PBF one. At this moment, Java code is called to parse the data and insert them into the appropriate tables. If the data are already available locally, the bounding box extraction and the file conversion are skipped, and the parsing is directly executed.
For timetable network, the process is started with *-b* option. The VDV files are always downloaded from [5], they are transformed into GTFS through the tool from Google, and imported into the tables of *vdv-gtfs-tmp* schema. At this moment, The Java code is called and the bus information are inserted into the tables of our model.
If both *-s* and *-b* options have been used, the user can add also *-l* and the link network is created. Here only Java code is used to map the bus network to the street one, add the resulting links to ***Links*** table, and update the street edge after the process.
As we mentioned before, the data can also be imported using the old model. This is achieved in the script *-t* option which starts the execution of Java code that translate the data stored with the new schema into tables of the old one.

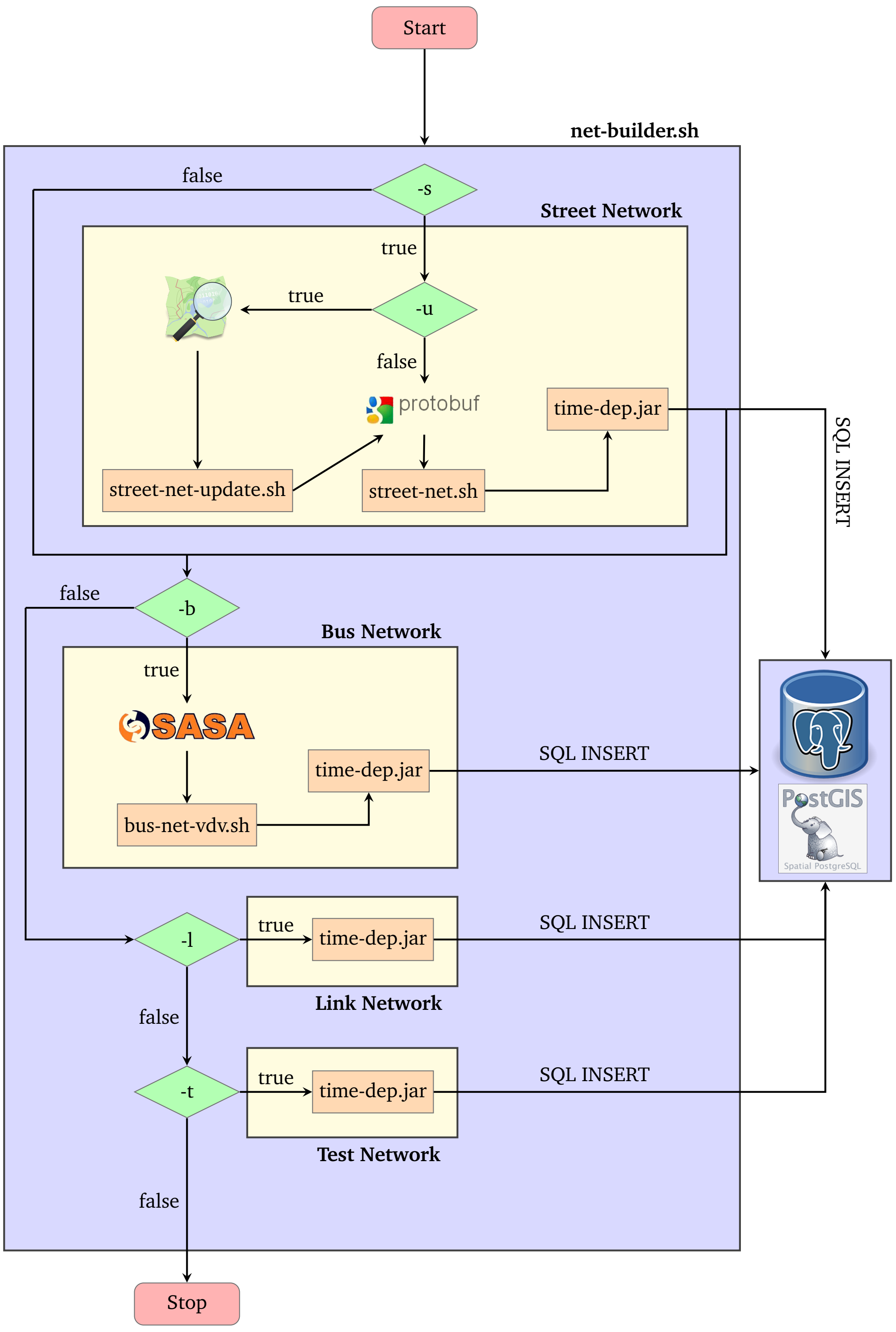


Figure 4.3: The complete work flow for data import.

Chapter 5

Discussion

In the next lines, we discuss our new data model, in particular we compare it with the one used for ISOGA, in order to underline the differences, the advantages and disadvantages of the two solutions.

Old Model. We present the old time-dependent data model, providing the details of its structure required for the comparison we show. The idea behind this solution does not separate the different modes from each other but stores them together in a single graph. Then, there is no explicit distinction between the street network and a timetable network. For this reason, the old model defines different schemas, in particular for nodes and edges, and it does not have a *links* table, since no further connections are required in addition to street and timetable segments.
Each node in this model is defined as:

$$\textbf{\textit{Nodes}}(\underline{ID}, Geom, Mode, Type, InDegree, OutDegree)$$

where *NodeType* and *NodeMode* provide both the mode of the point, respectively with a string among *PED* or *BUS* and with an integer among *1* and *0*. The other fields are defined as in our solution.
An edge contains much more information with respect to our idea. It has a schema like this:

$$\textbf{\textit{Edges}}(\underline{ID}, Source, Dest, Length, Route,\\ Mode, SourceMde, DestMode, Geom)$$

which contains all the fields to best suit both street an timetable edges. In addition, it stores also the a mode for the segment, for the source point and for the destination point. Then, we can have two types of connection: street and timetable. In the first one, the extremes can be of any mode, even both of *BUS*, since it may be the case that one go from a station on one side of a street to another station on the opposite side of the same street. In the second one, the situation is much simpler, since extremes must be

only of type *BUS*. Important is the *Route* value of an entry: as in our new model, there is one edge for each route connection between two stations in a timetable network, but the id of the extremes are always the same, since no route is associated with entries of ***Nodes*** table. Clearly, some fields are empty for both street and timetable connection: in the first case, no route is provided; in the second one, no geometry and length are present, since only the schedule has to be considered.
Information of schedules and routes are stored in the same way as we do; the calendar has a slightly different structure, which is as follows:

$$\textbf{\textit{Calendar}}(\underline{ID, StartDate, EndDate}, Monday, Tuesday, \\ Wednesday, Thursday, Friday, Saturday, Sunday)$$

where *ID*, *StartDate* and *EndDate* has the same meaning as in our solution; the rest of fields contains boolean values and replace the service vector that we use. An entry in such a table is means that from *StartDate* to *EndDate* a service is valid for days of week where the value is *TRUE*.

New-Old Model Comparison. We defined the old model and we proceed with a comparison of the two ideas, underlining the crucial aspects that distinguish them. First of all, the model we propose needs more space to store the data due to the separation we perform. In fact, for timetable networks the numbers of nodes is higher in comparison to the old model, since for each route that use a station, there is an entry in ***TimetableNodes*** that register this event. Furthermore, also the size of street nodes is greater, since in the old model the entries added after the mapping of the timetable network correspond exactly to the timetable nodes. In other words, if s_n, b_n, l_n, S_{old}, S_{new} are respectively the total number of street, timetable and link nodes (added to street graph after mapping), the size of all nodes of old model, and the size of all nodes for new model, we have that:

- $S_{old} = s_n + b_n$
- $S_{new} = s_n + b_n + l_n$; since $l_n = b_n$, $S_{new} = s_n + 2b_n$

As the formulas show, in the new model the separation cause the number of timetable nodes to be doubled, for each of such network. In terms of edges, nothing change for both street and timetable networks. For the timetable edges of old model, each segment connecting the same extremes stations is associated to a route, which means that it is used only for that specific connection on that route; however, the same pair of source and destination points can be used for many different edges. The situation is not the same when merging the street and timetable graph to build the final multi-modal network. In fact, in the new model for each timetable node tn_j^i, there are two entries in the ***Links*** tables for each node that represent the same station

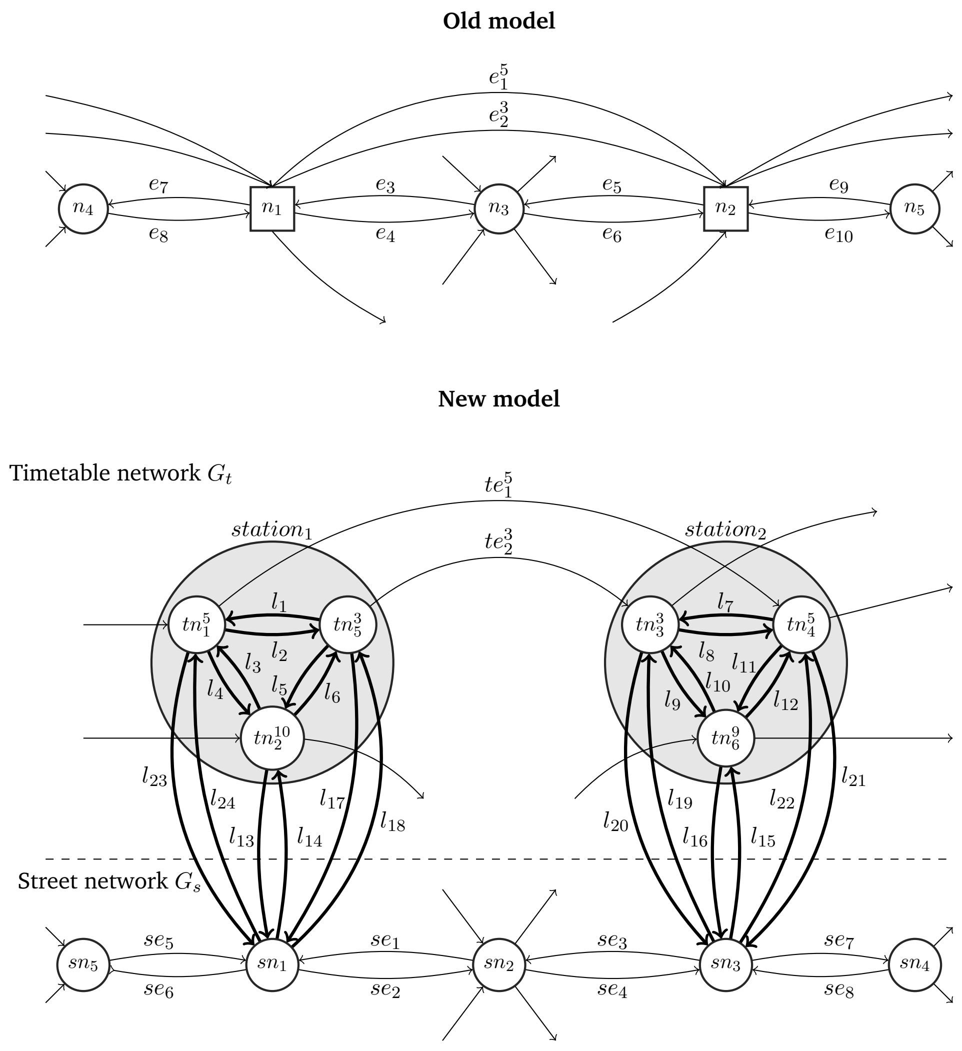


Figure 5.1: Old and new version of the time-dependent data model.

as *tn*^i_j but for different route *i*, and for the street node that map *tn*^i_j on the street graph. Then, each entry in ***StreetNodes*** has a pair of *link* connections for each node in ***TimetableNodes*** that corresponds to the station it maps. In case a new route is added or removed for one or more stations, the number of entries to add or removed from ***Links*** table are equal to $\binom{n}{1}$, where *n* is the number of existing nodes at that station.

In Figure 5.1, we provide a graphical representation of the comparison. In the old model version of the graph portion, node n_1 correspond to $station_1$ in the new model version, and in turn to sn_1. The same is for n_2, $station_2$ and sn_2. The remaining nodes n_3, n_4 and n_5 are street nodes that corresponds respectively to sn_2, sn_6 and sn_4 in the new model: in both situation

they are the same, with same number of edges entering and exiting them. The timetable edges as well do not change in size: line e_1^5 and e_2^3 corresponds respectively to te_1^5 and to te_2^3. The crucial difference between the two model is the number of timetable nodes and *link* edges. For example, if we add a new route 9 for $station_1$, we add a new timetable node tn_j^9 to represent it. In addition, we add an edge te_k^9 from tn_j^9 to tn_6^9. Then, to connect tn_j^9 to the other timetable nodes and to the correspondent street nodes, we have to add $\frac{2 \cdot 3!}{2!} = 6$ *link* edges. In the old model, instead, we just need to add just one edge e_h^9 in ***Edges*** table and we are done.
The situation we are interested in is when a single network has to be updated and the data are imported from scratch. Suppose we want to have the last version of a timetable network. In the old model, this requires to remove entry by entry all timetable nodes and edges from the correspondent tables plus the information about routes, calendar and schedule. After that, the new data can be imported paying attention to the consistency of the already existing information. In the new model, the update can be performed by completely remove the tables, re-create them and store the new data into them. In this way, the street network table is not modified until the *link* edges are inserted.

Conclusion

This last section is dedicated to report the inferences of our work, the advantages and disadvantages of the proposed model, as well as the future optimizations that can be performed on our initial proposal.
Regarding the actual modelling, the separation we adopted in our solution handles well the differences between various mode in a graph. This allows us to add or remove a new type of network without changing the structure of existing one. However, when adding a new mode, the existing networks has to be updated in order to include links: this means that new nodes are added that represent the switch points between modes.
In terms of performance and space, this model has some limitations, especially when applying graph discovering algorithm. On one side, the separation of the network and of the sub-networks (for graph with time table) store each single event at the same physical (real) point in a different object, as nodes or edges, which results in having many copies of the same location with different values for route (and also for id). Furthermore, when applying a graph search algorithm, the nodes and the edges to explore increased in number: this could be crucial when working with very big data sets. On the other side, the separation could help when working with system that uses preferences for modes, like ISOGA [4].
As future works, we propose some optimizations to enhance the reality of the data. First of all, the cost of the links can be take into account, since switching from one mode to another actually costs something in terms of time. Then, the street network can be further separated to obtain pedestrian, bicycle and car modes. In addition, if the data set of interest covers a big area, like country or continent, flight and ferry mode can also be included: they have still different structure in comparison with street or bus network. Since this model is thought for system like ISOGA [4] that works with isochrones, one could also think of including POIs to the model in order to have a complete scenario for a planning system.

Bibliography

[1] General Transfer Feed Specification. https://developers.google.com/transit/gtfs/reference.

[2] Geofabrik. http://download.geofabrik.de/.

[3] Google Protocol Buffer. https://developers.google.com/protocol-buffers/docs/overview.

[4] ISOGA. http://www.isochrones.inf.unibz.it/isoga.

[5] Opendata. http://sasabus.org/it/opendata.

[6] OpenStreetMap. http://http://www.openstreetmap.org.

[7] PBF parser. https://github.com/scrosby/OSM-binary.

[8] Verband Deutscher Verkehrsunternehmen. https://www.vdv.de/oepnv-datenmodell.aspx.

[9] Leonid Antsfeld and Toby Walsh. Finding Multi-Criteria Optimal Paths in Multi-Modal Public Transportation Networks using the T[r.

[10] Leonid Antsfeld and Toby Walsh. Finding Optimal Paths in Multi-Modal Public Transportation Networks using Hubs Nodes and TRANSIT algorithm.

[11] Hannah Bast, Erik Carlsson, Arno Eigenwillig, and Robert Geisberger. Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns. *ESA*, 2, 2010.

[12] Joel Booth, Prasad Sistia, Ouri Wolfson, and Isabel F. Cruz. A Data Model for Trip Planning in Multimodal Transportation System. 2003.

[13] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating Multi-Modal Route Planning by Access-Nodes.

[14] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Engineering Time-Expanded Graphs for Faster Timetable Information.

[15] Johann Gamper, Michael Boehlen, Willi Comett, and Markus Innerebneri. Defining Isochrones in Multimodal Spatial Networks. *CIKM*, 2011.

[16] Markus Innerebner. *Isochrones in Multimodal Spatial Network*. PhD thesis, 2013.

[17] Dominik Kircher. Efficient Routing on Multi-Modal Transportation Networks, 2013.

[18] Lu Liu. *Data Model and Algorithms for Multimodal Route Planning with Transportation Networks*. PhD thesis, 2010.

[19] Reem Fawzy Mahrous. Multimodal Transportation Systems: Modelling Challenges, 2012.

[20] Thomas Pajor. Multi-Modal Route Planning, 2009.

[21] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2), 2008.

[22] Jianwei Zhang, Theo Arentze, and Harry Timmermans. A Multimodal Transport Network Model for Advanced Traveler Information System. 2012.