Självständigt arbete på grundnivå

Independent degree project – first cycle

Datateknik
Computer engineering

Line Graph Utility
A software module for routing

Jonas Bergman

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MID SWEDEN UNIVERSITY
DSV

Examiner: Ulf Jennehag, ulf.jennehag@miun.se
Supervisor: Börje Hansson, borje.hansson@miun.se
Author: Jonas Bergman, jobe0900@student.miun.se
Degree programme: Programvaruteknik, 180 credits
Main field of study: Software technology
Semester, year: Fall, 2015

Abstract

This project was about building a line graph utility, a software module that should read map data from a PostGIS database and transform that information into a line graph (edge based graph) that the calling software could use to perform routing decisions. This outer calling application is part of a project (by an anonymized company) for flexible public transportation, that is meant to manage and direct a fleet of vehicles to where the customers actually are, instead of idling at bus stops. The software module should take different kinds of restrictions and conditions into account when building the line graph, to reflect the actual traffic situation. That can be turn restrictions, traffic signs, inclination, or conditions such as temporary hindrances, time of day. Some are static, but others vary dynamically and the state is to be found in the database.

This study has found a set of tools that aids in the transformation of OpenStreetMap data into a PostGIS database; for building the topology of the map; querying the database; and data structures for representing the graph and line graph.

The result of the project is a piece of working software that can return a line graph as a Boost graph with some restrictions taken into account, but it has not yet implemented them all, and more specifically, it does not handle conditional restrictions yet. There remains a good deal of work to implement all that complex logic.

Contents

1 Introduction

The work presented in this thesis is about flexible routing of public transportation. The result of the work is a software module that loads map data and converts it into in-memory data structures that can be used for routing decisions by exposing an *API*, (Application Programming Interface). This module is part of a bigger transportation optimization system that is meant to enable flexible public transportation solutions.

The module will be used for finding efficient routes in a dynamic traffic environment, i.e. the complete solution must take turn restriction, traffic lights and road signs into account. The outline of how to do this is by loading map data from *OpenStreetMap*[1] into a database (*PostgreSQL*[2] extended with *PostGIS*[3]). Upon a request directed to the API, the module should build (with soft real-time requirements) a data structure suitable for passing to a routing algorithm.

1.1 Background and problem motivation

The company (*anonymized*) aims at developing a solution for managing *flexible* public transportation, meaning no more buses standing idle and empty at bus stops, waiting just in case another bus fills up. The buses can be directed to where they are needed, and part of the solution is finding the best routes and give directions to the drivers where they should go. The public does not need to wait at bus stops, but can ask for pick-up via a mobile app.

There can obviously be huge benefits from such a transportation system. Less vehicles are needed, and better utilization of the vehicles, which should be good both for the environment and the finances of the operation. The public should also benefit from having access to public transportation where needed, and not from fixed locations.

Central for such a system is efficient routing of the vehicles, with almost instant updates on restrictions made available to the drivers needing directions. This project is a small piece in that puzzle.

1.2 Overall aim

This project should result in a working software module, fulfilling the requirements set by the company. There is needed some preliminary studying of graph theory, data structures, and research into what theories and solutions that already might exist, and if so, if they can be adapted and used in this project.

1.3 Scope

The scope of this project is to create the routing data structures representing the map data, not the routing algorithms, although they might affect one another, such that the choice of algorithm might affect what data structures are suitable.

1.4 Detailed problem statement

The software in this project is a module, exposing a function. When the function is called, it should load map data from a database, which has previously been loaded with OpenStreetMap-data, and build a connected graph to be used for routing decisions, and the data structure is returned to the caller so it can be used for routing. The building of the graph should happen in *soft real-time* so that it reflects all known restrictions in the database. For example if one

[1] http://www.openstreetmap.org
[2] http://www.postgresql.org
[3] http://postgis.net

road gets temporarily closed it should be marked as such, and that should be represented in the graph.

The requirements from the company states that the graph should be represented as a *line graph*, which is a basic technique for representing available turns at junctions. The software module shall be implemented in C++, using the *Boost Graph Library*[4] for the data structures. The software should be developed using *Behavior Driven Development* (BDD) or *Test Driven Development* (TDD) as methodologies, and otherwise adhere to the company's coding standards.

1.5 Outline

- Chapter 2 will present some background on graph theory, and research in map routing, regarding both theoretical foundations and some available implementations.
- Chapter 3 shows the methods and tools used.
- Chapter 4 is about the design and implementation of the software module.
- Chapter 5 presents the results from testing the implementation.
- Chapter 6 will include some discussion and conclusions made during this project.

1.6 Contributions

The work presented in this report is the sole work of the author.

[4] http://www.boost.org/doc/libs/1_54_0/libs/graph/doc/index.html

2 Related work

One of the first applications of *graph theory* was when *Leonhard Euler* considered the *Königsberg bridge problem*: Is there a way to walk over the seven bridges of Königsberg only once?

It is trivial to see that there is a close correlation between graphs and maps, as you can see in figure 2.1, with the roads and junctions in the map being lines and dots in the graph. A line is mostly called *edge* or *arc* and a connecting dot is called *vertex*, *node* or *point*; in this report it will mostly be *vertex* and *edge*, but to differentiate, another type of graphs called *line graph* will use the names *node* and *line* to distinguish. As one delves deeper into the theoretical material, one will find that there is good to know some *graph theory* and be familiar with some definitions.

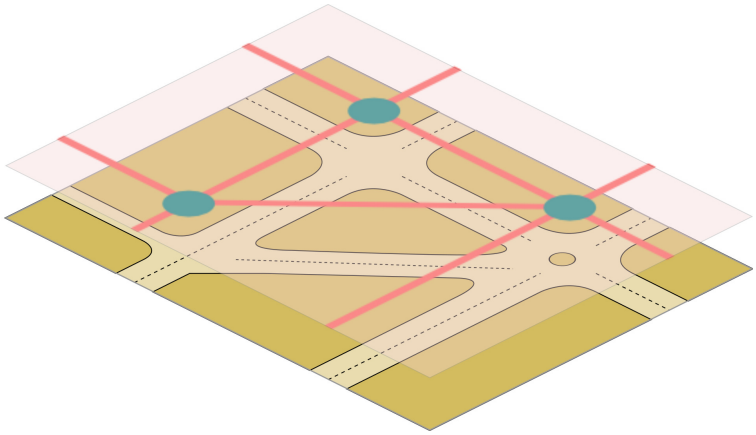
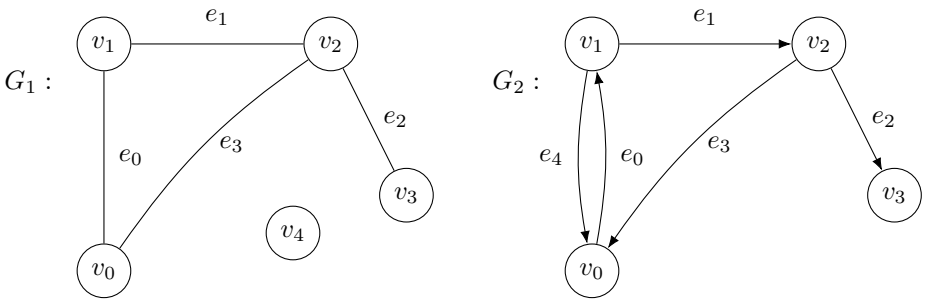


Figure 2.1: Graphs and road maps are a natural match.

2.1 Graph theory

There are good lecture notes such those from *Tampere University of Technology* [1] and *Univerity of Turku* [2] (both happen to be Finnish) and a good text book by *Reinhard Diestel* [3] are available to get into this subject. One does not need to understand all the concepts, but be familiar with some basic definitions and notations.

A *graph* is made up by *vertices* and *edges*, see figure 2.2.



(a) Undirected (b) Directed

Figure 2.2: A *graph* with *vertices* and *edges*.

So a graph G is a pair of sets, $G = (V, E)$ where $V = \{v_0, ...\}$ is the set of vertices and $E = \{e_0, ...\}$ is the set of edges. The edges can have their own labels as in the figure, or they can be denoted by the pair of vertices they connect: e_0 could be also named as (v_0, v_1) or v_0v_1. A graph can be *undirected* (figure 2.2a) if the edges have no sense of direction, or it can

be *directed* (figure 2.2b) if the direction of travel along an edge matters; e_0 is distinct from e_4 because the have different directions although they connect the same nodes v_0 and v_1. A directed graph can also be called a *digraph*.

To decide how "big" a graph is, one can count the number of vertices, $|V|$, to get the *order* or *cardinality* of the graph. If one counts the number of edges, $|E|$ one gets the *size* of the graph. In figure 2.2, G_1 has order 5, and size 4; and G_2 has order 4 and size 5.

Edges are *adjacent* if they share a common vertex, and vertices are adjacent if they are connected by an edge, one can also say that v is *incident* with e. In figure 2.2a, v_0 and v_1 are adjacent but not v_1 and v_3, and e_0 and e_1 are adjacent but not e_0 and e_2.

The number of edges connecting to a vertex is called the *degree* of the vertex, $d(v)$. In figure 2.2a, $d(v_2) = 3$, $d(v_3) = 1$ and $d(v_4) = 0$. A vertex of degree 1 is called a *pendant* vertex, or *leaf*, and a vertex of degree 0 is called *isolated*. If all *components* of a graph are *connected*, then the graph is a connected graph. In figure 2.2 graph G_2 is connected, but graph G_1 is not because it has an isolated vertex as a component.

A graph is *planar* if it is possible to draw without edges crossing each other. It is *Eulerian* if one can travel over every edge in the graph only once (as in the *Königsberg bridge problem*). A graph is called *Hamiltonian* if one can visit every vertex in the graph only once (as in the *Travelling salesman problem*).

Travels in graphs can be called different names. Ruohonen [1] has the most general name *walk* for travel from vertex to vertex along edges. A walk is *open* if it ends on a different vertex than it started, or *closed* if it ends on the same vertex. If an edge is traversed only once, the walk is called a *trail*. If any vertex is visited only once then the trail is a *path*. If the walk is a path but with the start and ending vertices being the same, then the walk is a *circuit*.

One can partition a graph into *subgraphs* if on places a cut in a vertex (*cut vertex*) or over a set of edges (*cut set*). In figure 2.2a a cut vertex could be v_2 and a cut set could be $\{e_1, e_3\}$.

2.1.1 Graph representation

There are different ways of representing graphs. We have so far used

- Graph diagram
- Set definitions, $V(G) = \{v_0, v_1, v_2, v_3, ...\}, \quad E(G) = \{e_0, e_1, e_2, e_3, ...\}$

One can also use

- *Adjacency matrix*
- *Incidence matrix*
- *Adjacency list*

An *adjacency matrix* is a matrix that shows if vertices are adjacent or not. A value of 0 indicates that the vertices are not adjacent. For an *unweighted* graph, adjacency can be indicated with a 1, or if it is a *weighted graph*, it can be the value of the weight (e.g. edge length or cost). From figure 2.2a:

$$D = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} v_0 & v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{2.1}$$

An *incidence matrix* describes which vertices that are incident with which edges. In a directed graph it is *positive* if it is the *start* vertex of the edge, or *negative* if it is the *ending* vertex of

the edge. From figure 2.2b:

$$A = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccccc} e_0 & e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & -1 & -1 \\ -1 & 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix} \end{array} \tag{2.2}$$

A *dense* graph has almost all vertices connected to each other, i.e. there are few 0s in the adjacency matrix. In a *sparse* graph there are a lot fewer edges than there could be, so the adjacency matrix has a lot of 0s. To be space efficient, especially in computing, it can therefore be better to represent a graph as an *adjacency list*, which simply lists for each vertex which other vertices it is adjacent to. No 0s needs to be included. From figure 2.2a:

$$v_0 : (v_1, v_2), \quad v_1 : (v_0, v_2), \quad v_2 : (v_0, v_1, v_3), \quad v_3 : (v_2) \tag{2.3}$$

2.2 Map routing

For graphs as those described above, there exists basic algorithms such as *Dijkstra* and *bidirectional search*, or more goal directed such as $A*$, that tries to find the shortest path from vertex s (source) to vertex t (target). To do that, each edge needs to be associated with a length. That is, the *metric* is *distance*.

However, when it comes to map routing there can be other metrics that are more important than the shortest path. For example *time* (we want the shortest driving time); *road category* or *land use* (we don't want to route through a residential area with low speed limits, or avoid having to go by ferry); *turn cost* (turning slows driving down so prefer straight routes); *multimodal* (when going by public transport we want to minimize waiting and the number of exchanges); *via* (we want to travel via a specific road or city); and so on.

A really basic ingredient in map routing is of course also the fact that roads are directed, i.e. there can be one-way roads. It is also important to take into account that there can be turn restrictions, so that a turn is not allowed at a junction, although it looks like it on the map (and the graph). Even more complicating is the fact that different restrictions on roads might be permanent, or just temporary due to road work, accidents, etc, so there is a difference between *static routing*, where the metric costs are static, and *dynamic* routing where the costs fluctuate over time.

2.2.1 Overview

In an overview of route planning techniques from 2009 [4], it is stated that the starting point for a "horse race" in developing speed-up techniques started in 2005 (p.124), when continental sized road networks of Europe and USA were made publicly available. Before that, large map data had been proprietary and it was hard to compare different approaches. The last decade since then has seen a quick development in the area, so a new overview in a tech report in 2014 from researchers in German universities and Microsoft [5] stated that the previous report was now outdated. This last report is a great overview of route planning techniques from the basic *Dijkstra*, continuing to different families of techniques: *goal directed*, *separator based*, *hierarchical*, *bounded hop*. The report also describes combinations of different techniques and notes on *path retrieval* (getting a description of the shortest path, no just the cost), *dynamic networks* and *time dependence*.

The motivation for the speed-up techniques is to enable "instant" route planning in large networks. The Dijkstra algorithm might need some seconds to complete a query, while one with some preprocessing might be able to perform a query in milli- or even microseconds. This is done by dividing the work into two distinct phases: the *preprocessing* phase, and the *query* phase. The preprocessing phase takes the original graph and performs transformations and builds new data structures. This is a process that can take a lot of time, from seconds

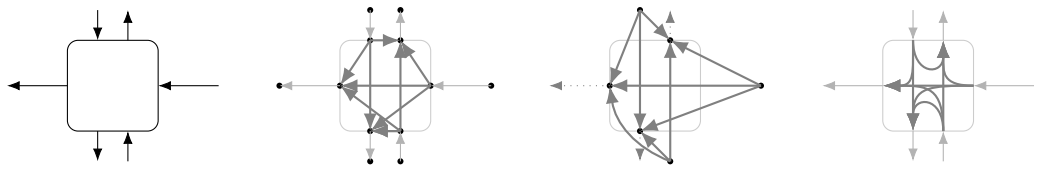
to hours and even days depending on algorithm, and the data the size of the data structures might multiply several times. The gain is that the query phase executes almost instantly.

A lot of the research has been conducted on simple models without turn restrictions, so it is easy to compare the speed gain to Dijkstra's algorithm, and one have thought that adding turn costs or restriction on top will not be so hard. However, it turns out that most algorithms with large gains in speed are quite inflexible and have trouble to incorporate changing restrictions and metrics without the need for running the preprocessing phase again [6, p.2]. A more flexible way would be to have a separation of *topology*, i.e. how the graph "looks" with vertices and edges, from the *metrics*, i.e. the cost for travel in the graph.

Those techniques with preprocessing can be characterized as *offline* techniques, while techniques that perform all processing in the query phase can be called *online*. As said before, a lot of the research has been done on continental scale maps. But if one restricts one self to a metropolitan map with a graph of a smaller size, then perhaps the queries perform fast enough without preprocessing, or the preprocessing phase is so fast that can be run online?

2.2.2 Map representation

To have a real-world application that performs route planning, it also needs to seriously take turn restrictions into account, and to be more useful also be able to handle *turn costs*. There exists several techniques for that, see figure 2.3. The most straight-forward technique might be to introduce some new vertices so that edges have a *head* and a *tail* vertex, and turns are modeled by connecting head and tail vertices; this is called a *full-blown* representation (figure 2.3b). One of the most used representations is by converting the original *vertex-based graph* to an *edge-based graph* (also called *arc-based graph* or *line graph*). It can be viewed as connecting tails to tails, see figure 2.3c. These techniques introduces several new edges and vertices, inflating the space needed for the data structures. A more compact representation is keeping a table for each junction with the associated turn and turn costs, see figure 2.3d.



(a) None (b) Full (c) Edge-based (d) Turn table

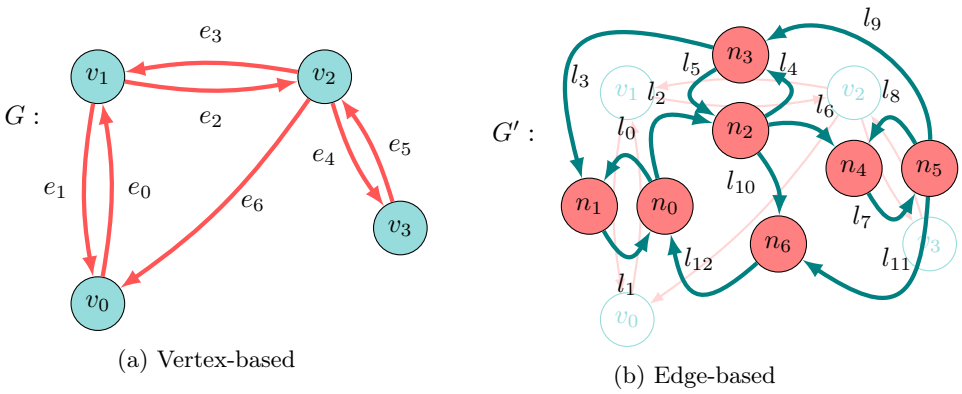
Figure 2.3: A closer look at a junction with two bidirectional and two unidirectional roads with different turn representations. *(After [6, p.8].)*

Edge-based graph

An *edge-based/arc-based/line* graph is a pretty straight forward transformation, where the edges of the original graph is turned into vertices in the transformed graph, and two vertices in the transformed graph is connected by an edge if a turn is allowed in the original graph. To make it simpler to distinguish beteween the vertex-based original graph and the new edge-based graph, we can call the new vertices *nodes*, and the new edges *lines*, i.e. "*road* = *node*", with nodes connected by lines, if a travel is allowed. This gives us a graph $G' = G_{edge-based} = (N, L)$ where N is the set of *nodes*, $N = \{n_0, n_1, ...\}, N = E$ and L is the set of *lines*, $L = \{l_0, l_1, ...\}$ connecting the nodes, see figure 2.4.

As one can see, the complexity and size of the graph grows in the transformation, what was $|V| = 4$ vertices and $|E| = 7$ edges became $|N| = 7$ nodes and $|L| = 13$ lines. The increase in size of the data structures is one drawback with this simple transformation, but on the positive side is the fact that one can apply ordinary algorithms such as Dijkstra to the edge-based graph just as easily as on the original graph. Another disadvantage might be that it lets the topology represent metrics, i.e. a turn restriction is hard coded into the topology, so how does one handle temporary restrictions?

Volker [7] has written a study on "Route Planning with Turn Costs" and uses edge-based



(a) Vertex-based

(b) Edge-based

Figure 2.4: Transformation of a graph G to an *edge-based* graph G'.

graphs as the foundation. He also introduces an *interface graph* as a link between a vertex-based and an edge-based graph. This new graph builds on an elaboration on what a turn and a junction actually is, with an incoming and an outgoing edge being adjacent on the junction vertex. One can thus see a turn T as either $T = (e_{in}, e_{out})$, i.e. an incoming and an outgoing edge, or as $T = (s, t, u)$ as going from vertex s to vertex u, *via* vertex t. This kind of finer look at what a turn is, is also used by others to build a more compact representation, see below, section 2.2.2.

Turn tables

One way of dealing with a more compact representation of turns and restrictions and costs associated with them is presented in "Efficient Routing in Road Networks with Turn Costs" [8], where they use a standard vertex-based graph (where roads are edges and junctions are vertices) on which they use speed-up techniques such as *Contraction Hierarchies*, but they associate each junction with a table that describes the costs of turns there, essentially describing all possible turns and restrictions. It turns out that a lot of junctions in a road map share the same characteristics and therefore can share the same *turn table*, so that on a map over Europe on average 18 vertices could share the same turn table. Thereby they managed to reduce preprocessing time by a factor 3.4 and space by a factor 2.4 with the same query times.

Another solution which uses the vertex-based graph with a turn table at the junctions is described in "Customizable Route Planning in Road Networks" [6, p.6]. Their solution uses a *separator-based* speed-up technique, which has been viewed as slower than hierarchical techniques, but they argue that this is the most flexible solution with a clean separation of *topology* and *metrics*, with two preprocessing stages before the query stage; one slow *metric-independent* that works on the topology, and a faster *metric customization* stage that can be run for each metric (takes about a second). This solution also uses the fact that many junctions share the same characteristics and therefore turn tables can be shared.

Bidirectional edges

So far we have thought of the directed original graph as having an edge for each direction of travel between two vertices, as with edges e_0 and e_1 in figure 2.4a. But this can be more compactly represented with one edge having a couple of flags indicating directions, meaning that for most roads we need not have two complete edge structures, but only one with two extra bits indicating the direction. This is however not something that can be done in an edge-based graph as the *lines* in it are not bidirectional.

2.3 Map data

One needs to have good map data as the source for building these data structures and apply smart algorithms on. With poor data it does not matter if one has smart algorithms. A said earlier, a race in route planning started with the public release of previously proprietary data. About the same time *OpenStreetMap* also began, which is an *crowd-sourced* project, meaning that anyone interested can be a cartographer and contribute with map data. Over the years the project has grown to an impressive size, and is used as the base for many applications. However, it turns out that the map data actually lacks a lot of turn restrictions. They might be hard to enter, and they are impossible to spot when comparing aerial photos with maps. Efentakis et al. states that for Athens with 277 thousand vertices only 214 restrictions were entered. They propose an automated remedy by comparing GPS-traces to the maps and deducing that turns seldom made are actually banned and could be marked as restricted in the map data [9].

All the same, a lot of high quality applications exists built on *OSM* (OpenStreetMap) data and this project aims at that to.

2.3.1 Projections

Generally speaking, the globe is spherical, but a map is flat. That means that one somehow needs to *project* the spherical data on a flat surface. There exists a lot of different *projections* that tries to do it best. To keep track of which projection one is working in, one can identify it by its *SRID – Spatial Reference System Identifier* that uniquely identifies which projection and which kind of coordinate system one works with.

2.3.2 Topology

The *topology* is about the relationship between objects in a map [10]. If one only thinks of a map as a collection of lines, it is hard to make something out of that information. It becomes useful when we understand the topology, that "this line is connected to that line at this point". The we have a relationship between the lines and can understand how to travel on the map.

Analyzing the topology also makes it possible to correct errors made while adding items to the map data, such as if two lines don't actually meet. Then there is a gap and there is no connection. When analyzing, one can opt to connect lines that are within a small distance of one another, thereby correcting mapping errors.

2.4 Available applications

Some of the research described earlier in this chapter is used actual working applications. For example *CRP (Customizable Route Planning)* [6] is used in Bing Maps, and *CH (Contraction Hierarchies)* are used in for example GraphHopper[1] and OSRM[2], and they are open source routing applications. So the source code is available so one can study how the data structures are implemented and how the algorithms work. There exists a lot of other solutions built on OSM as well[3], using Contraction Hierarchies or other speed-up techniques.

2.5 Memory or database

All research referred to so far has been about building data structures to be held *in memory* so algorithms can operate on them. But as we speak of *queries*, on might think that databases and query languages might be useful as well. There is some research, and a technique called *HLDB* is interesting [11], [12]. It is fast enough, and very flexible, permitting to query for *alternative routes* and *points of interest*.

[1] https://graphhopper.com/
[2] http://project-osrm.org/
[3] http://wiki.openstreetmap.org/wiki/Applications_of_OpenStreetMap

pgRouting is an open source database extension to *PostgreSQL*, often used for holding OpenStreetMap-data. It has a function called `pgr_trsp`[4] that looks for the shortest path with turn restrictions, so obviously standard relational databases can be part of a solution.

[4] http://docs.pgrouting.org/2.0/en/src/trsp/doc/index.html

3 Methodology

The overall methodology for the development is a combination a of *Behaviour Driven Development, BDD* and *Test Driven Development, TDD*, meaning all features of the module have either a *scenario* (BDD) or a *test case* (TDD) written.

As for tools, some are given in the specifications (see Appendix A), while others have been chosen during a selection process. Below is presented the tools chosen, and in some cases what alternatives that were also considered and tested. The categories are:

- Behaviour and Test Driven Development.
- Database.
  - Database and extensions.
  - Loading OpenStreetMap into database.
  - Build topology.
  - Examining map data.
  - Connecting to database from application.
- Reading configurations from `json`-file.
- Building graph.

3.1 Behaviour and Test Driven Development

Behaviour Driven Development tests usually have the structure: *Scenario* → *Given* → *When* → *Then*, written with words to describe the steps. An example in the *Gherkin* language is shown in listing 3.1.

```
Scenario: vectors can be sized and resized
     Given: A vector with some items
     When:  the size is increased
     Then:  the size and capacity change
```

Listing 3.1: Example of a BDD scenario in *Gherkin*.

So when developing BDD style one has to think through different scenarios and write them down, which can be helpful when thinking about what one tries to accomplish.

3.1.1 Tools, installation and usage

The testing library for this project is Catch[1], which is a small library for both BDD and TDD, where the BDD *"scenario"* corresponds to a TDD *"test-case"*, and *"given"*, *"when"*, *"then"* corresponds to *"section"*, meaning one can choose the development style one wishes. *Catch* was chosen because it is header only, and there is no need for complicated building of libraries and setting up paths; one can just include the header in the project and go.

Simply download the file `catch.hpp` and put it either in your project tree or in your path for includes.

Include the header in the source for your test, and get Catch to provide a `main`-method. See listing 3.2 for an example of how to implement the above stated "feature".

[1] http://www.catch-lib.net

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"
#include <vector>

SCENARIO ("Vectors can be sized and resized", "[vector]") {
    GIVEN ("A vector with some items") {
        std::vector<int> v(5);

        REQUIRE (v.size() == 5);
        REQUIRE (v.capacity() >= 5);

        WHEN ("the size is increased") {
            v.resize(10);

            THEN ("the size and capacity change") {
                REQUIRE (v.size() == 10);
                REQUIRE (v.capacity() >= 10);
            }
        }
    }
}
```

Listing 3.2: A basic BDD scenario with Catch

3.1.2 Alternatives

The BDD style of developing seems not to have caught on in c++ so much. There are a few libraries. Cucumber-Cpp[2] was investigated as it is an implementation for c++ of the Cucumber tool, which is widespread in many programming languages, so one could write the test for features in the ordinary `.feature`-files in the *Gherkin* language, that are common for writing features for tests. But I could not get Cucumber-Cpp to build correctly with CMake and the dependencies.

3.1.3 Remarks

It should not be a very difficult task to write a script that reads a `.feature`-file and outputs a template in c++, using the Catch syntax.

If one were to *not* go for BDD-style of testing, then one could go for TDD testing using Boost Test, if one would want to keep using Boost for most parts of the project.

3.2 Database

The database of choice, and in the requirements of the project, is PostgreSQL[3], with the extension PostGIS[4] which gives the database *spatial* and *geographic* capabilities, which are needed to simplify working with maps and such, for example when needing to measure distances in different projections. How to set up the database with users and passwords and such are not given in this report, but it is not so hard. When setting up databases one can interact via either the commandline or a *graphical user interface, GUI* such as *pgAdmin3*.

3.2.1 Tools, installation and usage

The tool set was given in the requirements, as mentioned before. On my Debian/Ubuntu system they can be installed as shown in listing 3.3.

[2] https://github.com/cucumber/cucumber-cpp
[3] http://www.postgresql.org/
[4] http://postgis.net/

```
$ sudo apt-get install postgresql postgresql-contrib-9.3 postgis postgresql-9.3-postgis-2.1 pgadmin3
↪   osm2pgsql
```

Listing 3.3: Installation of database tools

Listing 3.4 shows how to create a new database called `mikh_db` with a user "jonas" (that is already set up as a user with rights to create databases), and enabling the needed spatial extensions to work with map data.

```
$ createdb mikh_db -U jonas
$ psql -U jonas -d mikh_db -c "CREATE extension postgis;"
$ psql -U jonas -d mikh_db -c "CREATE extension postgis_topology;"
$ psql -U jonas -d mikh_db -c "CREATE extension hstore;"
$ psql -U jonas -d mikh_db -c "SET search_path=topology, public;"
```

Listing 3.4: Create database and enable spatial extensions.

3.2.2 Loading map data

To get the `.osm`-file, which is actually in `xml`, into the database one needs a conversion tool to parse the file and populate some tables with data.

Tools, installation and usage

There exists several tools for importing `OSM` data into a database. It was hard to know which one to pick and different options were tried, but the chosen tool is osm2pgsql[5]. It was installed in listing 3.3.

```
$ osm2pgsql -U jonas -d mikh_db -k -s mikhailovsk.osm
```

Listing 3.5: Usage of `osm2pgsql`.

Listing 3.5 reads an `.osm`-file in the current directory and populates the database `mikh_db`. The flags `-k` tells to use "hstore" for tags and, `-s` to make a "slim" conversion. Two different `.osm`-files have been provided for testing, "mikhailovsk.osm" and "partille.osm", hence the usage of "mikhailovsk.osm".

One might specify other flags as well. Among the options is to chose a different projection than the default `900913`. It is also possible to specify a `.style`-file which is a configuration over which tags to import. It is possible to use this file to decide which tags to import into the database and which tags to discard.

Alternatives

There exists a bunch of other tools that can convert OpenStreetMap files into database tables, such as *Osmosis, Imposm, osm2po, osm2pgrouting*, and others; all with different strengths and weaknesses, such as being good and free, but not open source.

3.2.3 Building topology

With the data in the database, it is time to build a topology of the map data, saying how the vertices and edges are connected, to make it possible to build a routable graph. A lot of the "nodes" in the `osm`-data are only useful for describing the geometry, while what is interesting when routing are the nodes that connects edges; that is the junctions at which roads meet. Therefore it is essential to analyze the data an build tables that contain information about the topology.

[5] http://wiki.openstreetmap.org/wiki/Osm2pgsql

One can have different thoughts of when to do this. It would be possible to do this at the preliminary step when loading the data into the database. That would be good if one was certain of that the topology is stable. If the network is more volatile, it would be better to build the topology on every query, to be certain that one always has the most up-to-date information. On the other hand; the topology for a road network should be stable, and temporary closures and other changing conditions will be better reflected in tags that can be queried when calculating costs for routing. That is the path taken in this project.

Tools, installation and usage

The choice for this project is PostGIS' topology extension. It is a part of PostGIS, which is already installed.

The `osm`-data from `osm2pgsql` has a table for all the lines in the map, called `planet_osm_line`, but in addition to roads it contains lines for railways, waterways, borders, buildings etc. So to build routing data we need to extract the lines only representing the roads, and put it in a new table. Listing 3.6 shows that.

```
$ psql -U jonas -d mikh_db -c "CREATE TABLE roads AS SELECT * FROM planet_osm_line WHERE highway IS
↪  NOT NULL;"
```

Listing 3.6: Creating a table with only roads in it.

Then one can build a topology of the roads, as shown in listing 3.7. The first line creates a new schema called `roads_topo` which will hold the topology data in the projection `900913` (the projection used when loading the database). The second line adds a column called `topo_geom` to the table `roads` in the public schema. The third line connects that column with the newly built corresponding topology in the `roads_topo` schema. The topology is built with a tolerance of 1.0 units. The unit for this projection is meters, so it means that if there are several nodes within 1.0 meters or a node within 1.0 meters from a road, they are joined. This can be essential to building a routable network. When running the validation tool in JOSM on the `mikhailovsk.osm`-file, it reported 16 suspect cases with nodes close but not connected, see figure 3.1.

```
$ psql -U jonas -d mikh_db -c "SELECT topology.CreateTopology('roads_topo', 900913);"
$ psql -U jonas -d mikh_db -c "SELECT topology.AddTopoGeometryColumn('roads_topo', 'public',
↪  'roads', 'topo_geom', 'LINESTRING');"
$ psql -U jonas -d mikh_db -c "UPDATE roads SET topo_geom = topology.toTopoGeom(way, 'roads_topo',
↪  1, 1.0);"
```

Listing 3.7: Building a topology with PostGIS.

Alternatives

One might load the database, and build a topology with osm2pgrouting[6], and the PostgreSQL extension pgRouting[7]. That solution is pretty smooth, and might heal the topology with a tolerance, but it seems it only builds the topology and does not give access to tags and other information usable when calculating costs.

Another attempt, was to run a topology building SQL function (as in `http://blog.loudhush.ro/2011/10/using-pgrouting-on-osm-database.html`), and then run another function to remove all nodes without topological meaning. But that lead to the problem shown in figure 3.1, as there had been no "healing" of nodes first. One solution could of course to write antoher function for that, or to fix the `.osm`-file manually in JOSM before loading it into the database. But the solution with the PostGIS topology seems like a better way to go.

[6] http://pgrouting.org/docs/tools/osm2pgrouting.html
[7] http://pgrouting.org/index.html

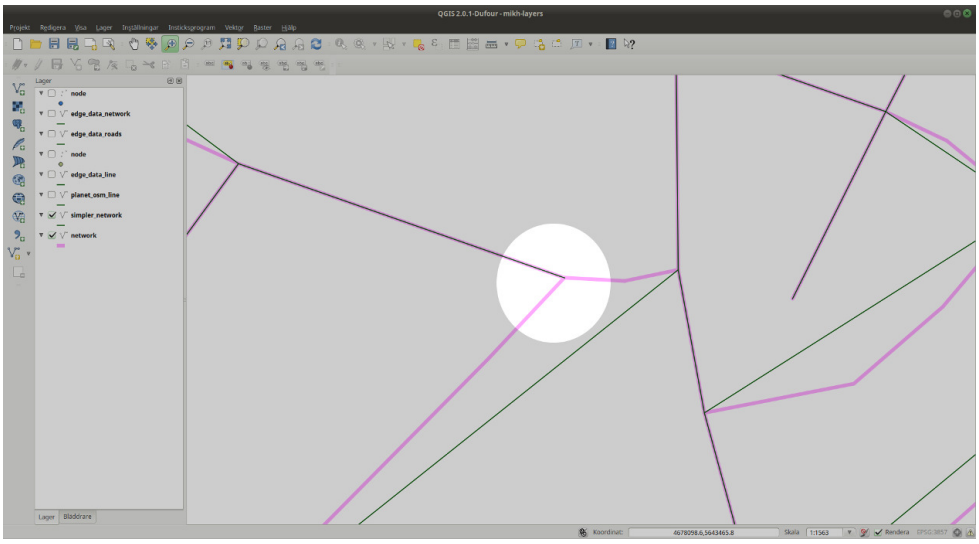


Figure 3.1: Error building topology with a node close but not connected.

3.2.4 Examining map data

Map data lends itself to visualization. And it is also useful to build a mental model of what one is working on, and to see the results.

JOSM

JOSM[8] is an editor for OpenStreetMap. It can open `.osm`-files and display them, inspect elements of the map, and it has tools for editing and validation, meaning one might be able to fix files that has problems. See figure 3.2.

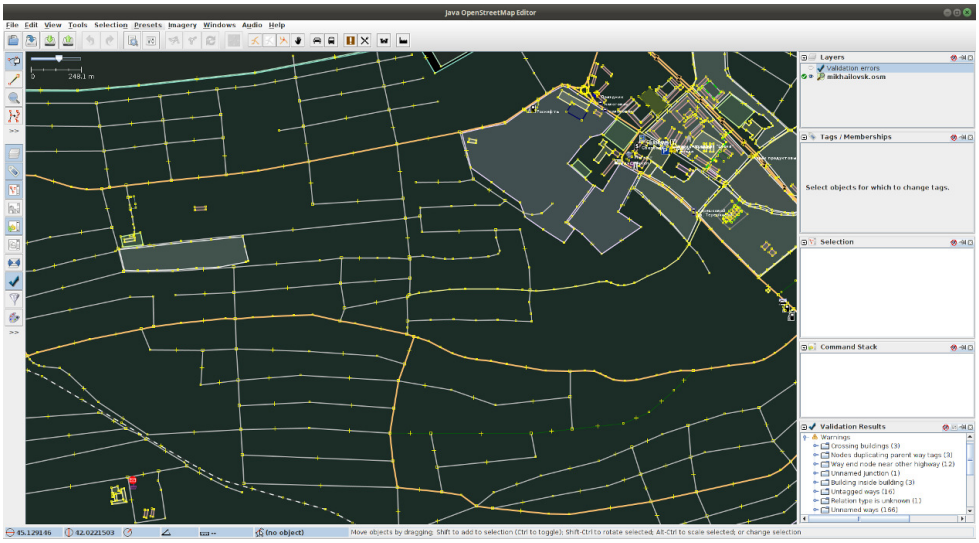


Figure 3.2: JOSM editor with Mikhailovsk map.

[8] https://josm.openstreetmap.de/

QGIS

QGIS[9] is a tool that can load spatial data from databases and display, as well as load for example `.osm`-files. It makes it good to visualize for example query results or transformations you have made in the database. See figure 3.3 for an example with layers of PostGIS-data of "Mikhailovsk" on top of each other.

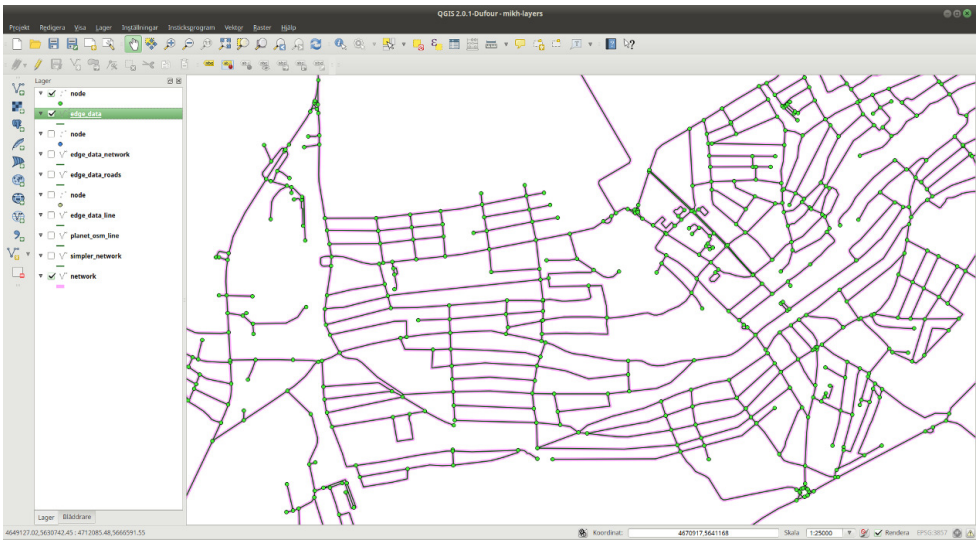


Figure 3.3: QGIS editor with Mikhailovsk map from PostGIS tables.

When loading data from PostGIS one might have to specify which projection to use for display. The default projection when loading `.osm`-files into the database using `osm2pgsql` is `SRID 900913`, and to display that correctly in QGIS one needs to use the projection `EPSG:3857`.

3.2.5 Connecting to database

After the module has read in the configuration, the next step is to connect to the database and perform some work on the map data before extracting relevant information.

The connection to the PostgreSQL database is handled by the library libpqxx[10], and while there exists a few alternatives, it is natural to go for the official alternative.

Tool, installation and usage

Installation of `libpqxx` is shown in listing 3.8.

```
$ sudo apt-get install libpqxx-4.0
```

Listing 3.8: Installing `libpqxx` on a Debian/Ubuntu system.

It is pretty straightforward to use: include the header, make connections and transactions. A snippet is shown in listing 3.9.

```
1 #include <pqxx/pqxx>
2 ...
3 pqxx::connection conn("dbname=testdb user=tester password=tester hostaddr=127.0.0.1 port=5432");
```

Listing 3.9: Include header and make a connection to the database.

When compiling, one must link with the libraries `pqxx` and `pq`, as shown in listing 3.10.

[9] http://www.qgis.org/
[10] http://pqxx.org/development/libpqxx/

```
$ g++ mytest.cpp -lpqxx -lpq -o mytest
```

Listing 3.10: Linking `libpqxx` at compile time.

3.3 Configuration

The module should be configured by a settings file, written as *json*. Settings can be related to the database such as host address, table names etc; or it can be configuration of costs for the routing such as speed limits, traffic lights, turn restrictions.

3.3.1 Tools, installation and usage

There is no meaning in writing a json-parser for this module as there exists lots of good libraries. The one chosen is Boost Property Tree[11], as the project uses other Boost libraries, and it simple enough to get started with.

As several Boost packages will be used in this project, it is just as good installing all of them (for a Debian/Ubuntu based system), see listing 3.11.

```
$ sudo apt-get install libboost-all-dev
```

Listing 3.11: Installation of Boost libraries.

An example to see how simple it is to parse a json-file is shown in listing 3.12.

```
1  #include <string>
2  #include <iostream>
3  #include <boost/property_tree/ptree.hpp>
4  #include <boost/property_tree/json_parser.hpp>
5
6  void readJsonFile(const std::string& filename) {
7      boost::property_tree::ptree pt;
8      boost::property_tree::read_json(filename, pt);
9      std::string host = pt.get<std::string>("host");
10     int port = pt.get<int>("port");
11     std::cout << "Host: " << host << ", port: " << port << std::endl;
12 }
```

Listing 3.12: Parsing a json-file.

3.3.2 Alternatives

One could go for a header-only solution here as well, such as jsoncons[12], which was also tested, but *Boost Property Tree* seemed nice and easy to get working if one already has the Boost libraries installed.

3.4 Build Graph

The requirements said that the *"Boost Graph Library (BGL)"* should be used for representing the graph and for returning the *line graph* structure for routing back to the calling application.

As discussed in section 2.1.1, the most space efficient way of representing a sparse graph is an *adjacency list*, and the *BGL* has such a data structure. Using template arguments one can configure what kind of data structures to use for the lists of *edges* and *vertices*, and the data structures to use for *edges* and *vertices*, and if the graph is *directed* or *undirected*.

If one has some properties of the edges and vertices that one wishes to keep in the graph (like the "cost" or some identifier of an edge), it is possible in several ways, either as *"interior"*

[11] http://www.boost.org/doc/libs/1_54_0/doc/html/property_tree.html
[12] https://github.com/danielaparker/jsoncons

or *"exterior"* properties, and `adjacency_lists` can use *interior* properties either as *"bundled properties"* or as *"property lists"*.

The *property lists* are external structures for some property that gets mapped to e.g. an edge in the graph.

The *bundled properties* are more intuitive, by using data structures as the *descriptors* of the *edges* and *vertices*, and with the properties as fields.

An example from the documentation for *bundled properties* [13] shows the difference clearly, in terms of how easy or hard it is to read or understand the code in the different approaches. See listing 3.13 showing the *bundled* approach and listing 3.14 showing the *property list* way.

```
// Vertices = Cities
struct City
{
  string name;
  int population;
  vector<int> zipcodes;
};

// Edges = Highways
struct Highway
{
  string name;
  double miles;
  int speed_limit;
  int lanes;
  bool divided;
};

// Map using `City` as vertex descriptor and `Highway` as edge descriptor.
typedef boost::adjacency_list<
    boost::listS, boost::vecS, boost::bidirectionalS,
    City, Highway>
  Map;
```

Listing 3.13: Bundled properties in a graph.

```
typedef boost::adjacency_list<
    boost::listS, boost::vecS, boost::bidirectionalS,
    // Vertex properties
    boost::property<boost::vertex_name_t, std::string,
    boost::property<population_t, int,
    boost::property<zipcodes_t, std::vector<int> > > >,
    // Edge properties
    boost::property<boost::edge_name_t, std::string,
    boost::property<boost::edge_weight_t, double,
    boost::property<edge_speed_limit_t, int,
    boost::property<edge_lanes_t, int,
    boost::property<edge_divided, bool> > > > > >
  Map;
```

Listing 3.14: Property lists in a graph.

In this project, the *bundled properties* were chosen for their ease of understanding and reading.

4 Implementation

As stated at the start of chapter 3, the software module called the *"Line Graph Utility", (LGU)*, that should be the outcome of this project, is sequential in nature. The complete specification is available in appendix A, but here is an outline of the main use case.

- The using application calls the LGU's `get_directed_line_graph()`.
- LGU queries the PostGIS database and builds a graph from the road network.
- LGU builds a directed line graph from the previous step (i.e. it converts nodes to edges and assigns weight to those edges based on road signs and other elements which are present in the nodes).
- `get_directed_line_graph()` returns a directed line graph structure which is based on a C Boost graph structure to the function caller.

This can be expanded to a series of steps. First comes a preliminary step, not actually part of the module, but essential during development and testing:

- Loading the map data into the database and build a topology.

The following steps are performed during development and usage of the tool:

- Load configurations from *json*-file.
- Get the relevant edges and vertices from the database; store the topology.
- Apply restrictions and costs on the topology.
- Build a graph structure from the topology, using *Boost Graph Library*.
- Transform the structure into a *line graph (edge-based/arc-based graph)*.
- Return the line graph.

4.1 Design

The sequential nature of the module, with a few easily identifiable objects, lead to no big design process was deemed necessary. Taking an object oriented approach, it is easy from the above list to identify *configuration (and configuration reader); edges; vertices; database; topology; restrictions; costs; graph (and graph builder and transformer); line graph.* All can be packaged up in a *Line Graph Utility*. The design therefore evolved gradually without a master plan more specific than this.

Another reason for this, was that this project was a discovery into not really well understood territory, despite some introductory research. It was necessary to learn the tools and concepts as the project proceeded, so the design and implementation grew incrementally. The incremental goals set during development, was to be able to build a graph from the map data, later extended to being able to build a line graph from that, to finally being able to apply restrictions and costs to the graphs.

A decision that was made early on, was to try not to pass pointers around, but instead use references, to reduce the complexity of memory handling. That means that a lot of functions gets passed in a reference to an object to fill in, rather than return a pointer to a newly constructed object. All the same, some pointers could not be avoided and raw pointers were used in those cases.

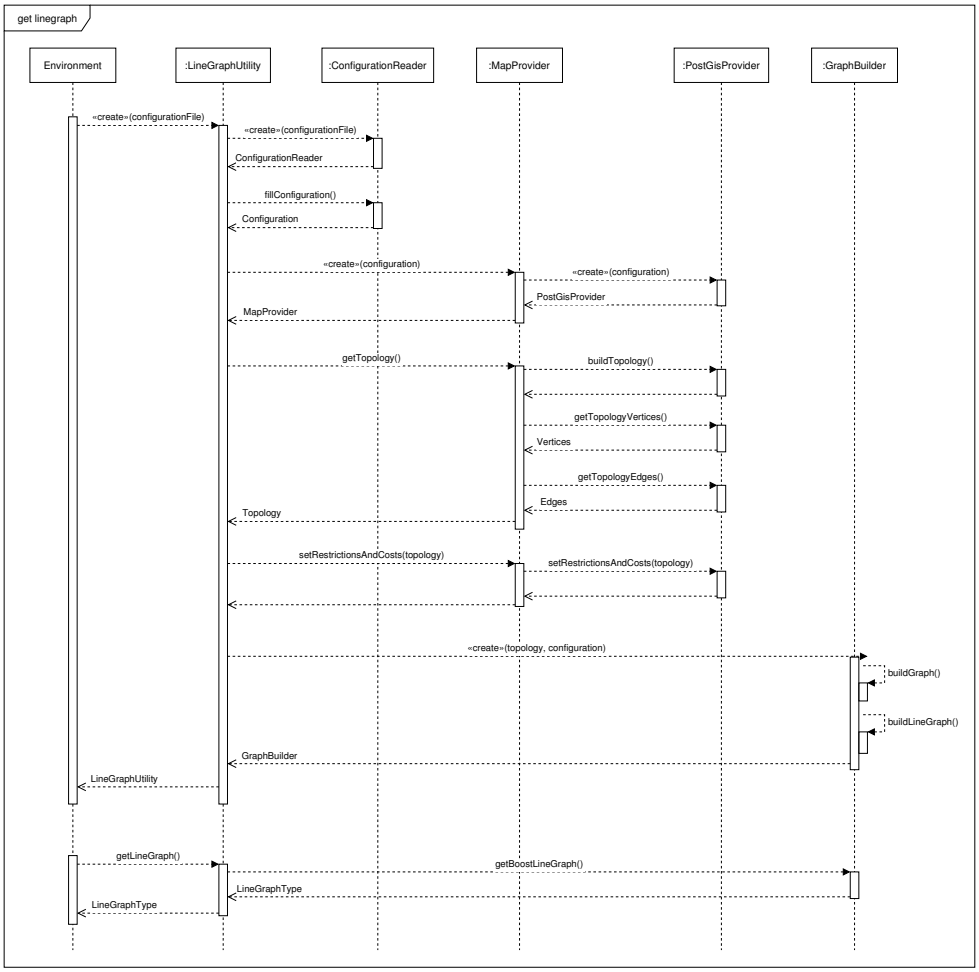


Figure 4.1: Sequence diagram of main use case to get a line graph.

4.1.1 Dynamic design

The sequence presented above has been refined into a design that can be shown in a sequence diagram, see figure 4.1 and appendix B.1.

The calling application *"Environment"* instantiates a `LineGraphUtility` object with the file name to a configuration file. The `LineGraphUtility` instantiates a `ConfigurationReader` that can be asked to fill in a `Configuration` object. The configuration contains among other things, a setting for which `MapProvider` to use. The idea is that one can read the *OpenStreetMap* data in several ways; for example parse the `.osm`-file, or use different databases or different tools to import `.osm`-files into the database. Hence the flexibility by using an abstract *map provider*. The only implementation in this project so far is the `PostGisProvider`, but others could be developed if it turns out there are better ways to access the map data.

So the actual work on retrieving the map data is performed by the `PostGisProvider`, that is fetching the `Topology` and applying *restrictions* and *costs* on the topology. The idea behind this separation is that the topology should be reasonably stable and constant, and that dynamic changes in the traffic, such as blocked roads, should be handled as restrictions and costs that are applied to the static topology. But it is also possible to perform an update on the topology if needed, for example if there has been built a new road. See figure 4.2 for a diagram of updating the restrictions and costs, and figure 4.3 for a diagram on updating the topology, (also in appendix B.2 and appendix B.3).

Back to figure 4.1, after having a restricted topology we instantiate a `GraphBuilder` object with the topology och configurations. This `GraphBuilder` builds a directed graph, and converts it to a *line graph*. If all went well the `LineGraphUtility` now is ready to serve the calling application a *line graph* any time it gets called.

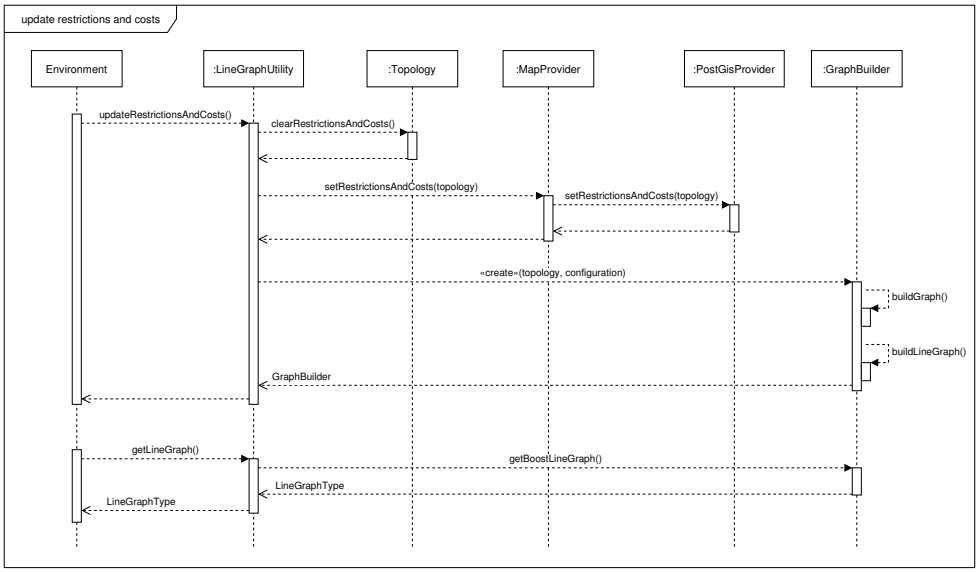


Figure 4.2: Sequence diagram of updating costs and restrictions on a topology.

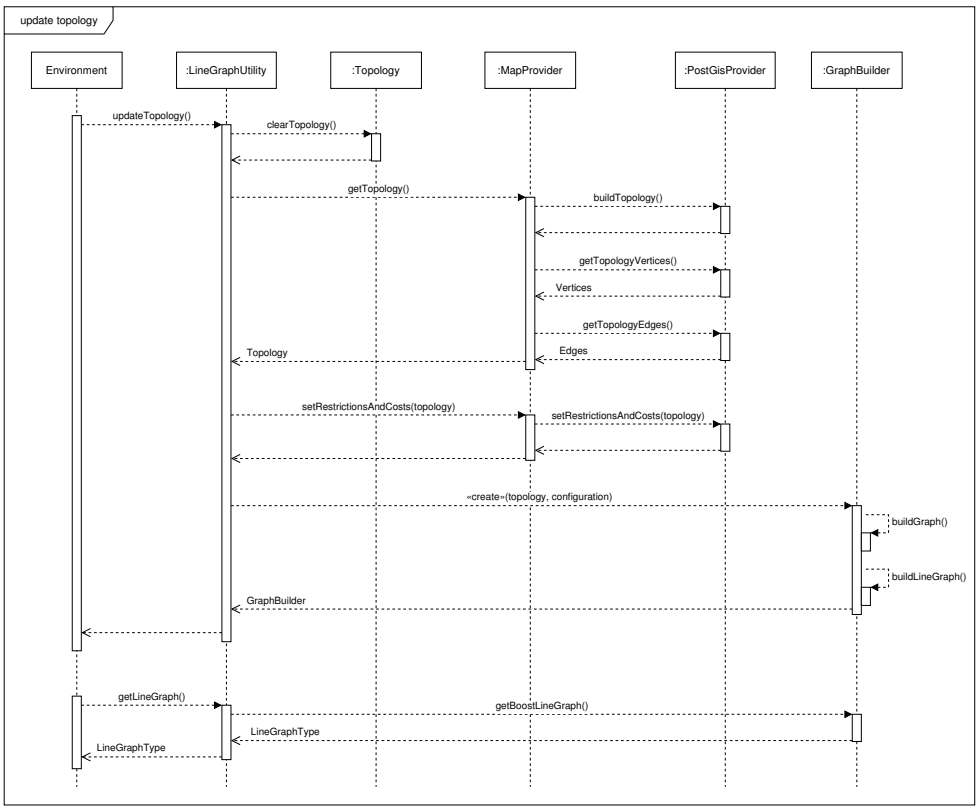


Figure 4.3: Sequence diagram of updating the topology.

4.1.2 Static design

A few classes were introduced in the sequence diagrams above, and a more complete view of the classes can be seen in the class diagram in figure 4.4 and in appendix B.4.

As can be seen, the application is divided in a few *"packages"*:

- `lgu`: The entry point to the `LineGraphUtility`.

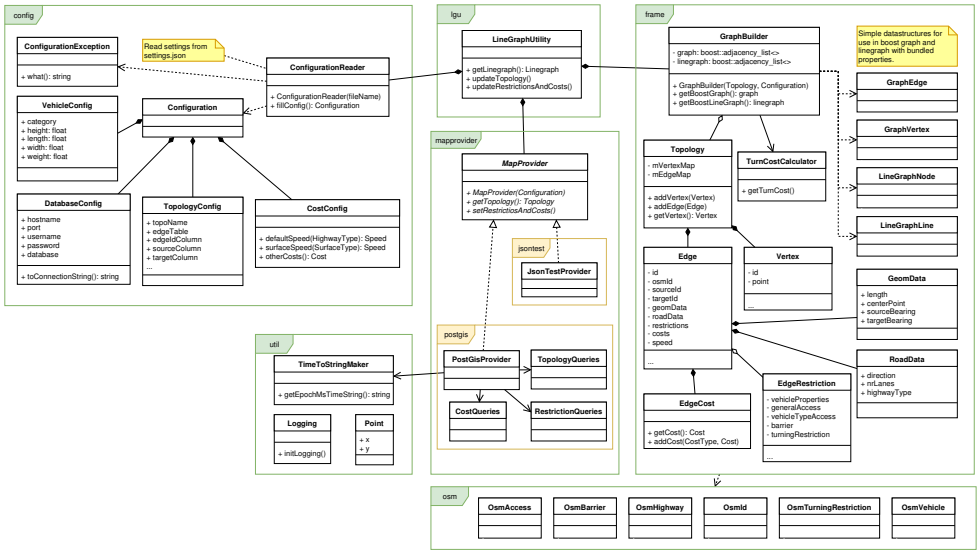


Figure 4.4: Class diagram of the *line graph utility*.

- `graph`: Classes related to graphs like `GraphBuilder`, `Topology`, `Edge`, `Vertex`.
- `mapprovider`: Classes related to providing map data.
- `config`: For handling configurations.
- `osm`: Helper classes for constants and concepts related to *OpenStreetMap* data.
- `util`: A few general helper classes.

This is an attempt to modularize the development process; to keep related classes in a specific area. It also makes navigating the code easier. Another attempt to make the packages coherent is that each package should have its specific *exception* class, that is the only public exception that gets thrown by the classes in the package. Other exception classes might be used internally but not exposed publicly.

4.2 Project structure

Apart from the packages above, there are directories in the project to support testing, setting up and documentation. This gives the project a basic directory structure as shown in listing 4.1.

```
./
├── catchtest/
├── config/
├── doc/
├── graph/
├── lgu/
├── mapprovider/
├── osm/
├── preparation/
├── util/
└── README.md
```

Listing 4.1: First level directory structure of the project

Each directory has a `README.md` file, a textfile in `markdown` mark up, that explains the

purpose of the directory. Each directory with code also should have a `catchtest` directory, where there are tests for the code in the directory/package.

4.2.1 catchtest

Most of the code developed in this project has been part of a test, and all *"packages"* have their own set of tests. *Catch* is a header-only framework, and that header resides in this root `catchtest` directory, and there is a source file that calls that header and functions as the entry point when testing, see appendix listing C.2. It also contains a sub-directory for configuration settings used during development.

In section 3.1.1 the *Catch* testing framework is introduced, and in listing 3.2 there is a small example of how to write a scenario. There one can see that the scenario is tagged with `"[vector]"`, and those tags can be used to determine which tests to run. If no tags are specified all tests are run, but if one specifies a tag, it only runs tests that matches that tag. One can also specify which tests *not* to run by prepending a tilde ('~') to the tag. See listing 4.2. If one wants to see the results of all tests and not only failed ones, one can add the flag `-s`. If running tests from inside and *integrated development environment, IDE* one can specify the arguments in a *"run configuration"* instead.

```
$ testapp ~[timing] -s
```

Listing 4.2: Running tests except those tagged with `[timing]`, showing all results.

4.2.2 config

See appendix listing C.3 for the contents of this package, whose purpose is handling configurations. The central part is a data structure `Configuration`, made up of data structures for *cost, database, topology* and *vehicle*. The configurations are filled in by the `ConfigurationReader` class which reads from a specified settings file.

The `CostConfig` are mainly concerned with keeping track of speeds for different categories of roads and surfaces. The types are specified in `OsmHighway` in the `osm` package, see section 4.2.7.

The `DataBaseConfig` is about connecting to the database.

The `TopologyConfig` is about which tables and columns in the database to use when getting the topology.

The `VehicleConfig` keeps characteristics about the vehicle we are routing through the map, such as weight, height, category (as specified in `OsmVehicle`, see section 4.2.7).

4.2.3 doc

Listing C.4 shows the contents of this package, that contains the documentation for the project. It has a directory for this report, and a directory for the UML diagrams.

The diagrams are not meant to be exact documentation, but rather give an idea of the concepts and the big picture, and therefore method names might be missing or spelled differently than in the actual code.

4.2.4 graph

The `graph` package (see appendix listing C.5) is really the central package, where the `Edge, Vertex, Topology` classes are, and the `GraphBuilder` resides. In addition there are the classes for restrictions and costs for edges (`EdgeCost, EdgeRestriction`); a helper class for calculating costs for turns (`TurnCostCalculator`); and a couple of essential simple types, `Cost` and `Speed` who are simply `typedefs`.

Edge

(See source code in appendix D.7.2 (header), D.7.3 (implementation)).
The `Edge` class represents an edge in the topology. So it keeps track of the *source* and *target* vertices, the original *OSM id* and of course its *id* in the topology. It also keeps track of properties of the underlying *road* (number of lanes; one-way; the road category), and its geometrical properties (length; centre point and *bearing* of the edge at the vertices to calculate turning angles).

Vertex

(See source code in appendix D.7.4 (header), D.7.5 (implementation)).
The `Vertex` class simply keeps the *id* it has in the topology and the *coordinates* of it, it does not keep track of an *OSM id* as it might not correspond to nodes in the *OSM* data, since the vertices were calculated when building the topology.

Topology

(See source code in appendix D.7.6 (header), D.7.7 (implementation)).
The `Topology` class is a collection of maps. One maps `{edgeId ↦ edge}`, that is it makes it possible to get to the topology's `Edge` object when one only has an *id* for it in the topology. The corresponding map exists for vertices, `{vertexId ↦ vertex}`. In addition there is a *multi-map* `{osmId ↦ edgeId}` that has an `osmId` as the key, that maps to several `edgeId`s. The reason for this is that an original road in the *OpenStreetMap* might be split into several edges when building the topology.

GraphBuilder

(See source code in appendix D.7.8 (header), D.7.9 (implementation)).
The `GraphBuilder` keeps a `Topology` and a `Configuration` as the base for building a *graph* and a *line graph*. The `GraphBuilder` header begins with defining a bunch of types, such as the data structures to be used for *edges* and *vertices* in the *Boost graph* (`GraphEdge`, `GraphVertex`). They keep track of the *id in the topology* and the corresponding *id in the graph*, so one can move from the one to the other. The `GraphBuilder` keeps a *map* for *vertex* such as `{topoVertexId ↦ graphVertexDescriptor}`, and a *multi-map* for *edges* such as `{topoEdgeId ↦ graphEdgeDescriptor}`. These maps make it possible to access the data in the data structures underlying the *descriptor*, e.g to get to the fields in the `GraphEdge` through the descriptor. The reason to use a *multi-map* for the edges are that the edge in the topology is undirected, but in the graph the edges are directed, so in the graph there should be a directed edge for each lane of the road, and thus for most roads there will be several *graph edges* for each *topology edge*.

The `GraphBuilder` also has the data structures that makes up the *nodes* and *lines* in the *Boost line graph*, and a map `{edgeId ↦ lineGraphNodeDescriptor}`, since that is the definition of a *line graph*: an edge turns into a node in the transformed graph.

There are also `typedef`s to make the code easier to work with, see listing 4.3.

```
1 typedef boost::adjacency_list
2    < boost::listS, boost::vecS, boost::directedS,
3     LineGraphNode, LineGraphLine >                    LineGraphType;
```

Listing 4.3: `typedef` a *line graph* to make the code more readable.

The operation when building the *graph* is that first all *vertices* in the *topology* gets added to the *Boost graph*, and then each *topology edge* gets examined to see if there are any restrictions that apply. If not, the correct number of edges (corresponding to the number of lanes) in each direction gets added to the graph.

When building the *line graph*, all the edges in the graph are added as *nodes* in the *Boost line graph*. To find which other nodes to connect to (= which travels are allowed), one has

to look at all *out-going* edges from the *end vertex* of the edge, functioning as a *via-vertex*. That *turn* (or travel) is really in three parts: *source edge* → *via vertex* → *target edge*. The *OpenStreetMap* data also gives the option to specify turns as travel via another edge instead of via a vertex, but that is complicated, see discussion in 4.2.7. When the *adjacent* edges have been identified, they are one by one checked if they are part of any *turning restriction*. If the edge is not part of such a restriction, then a *line* (that is a *line graph edge*) is constructed from the *source node* to this *target edge/node* and added to the *line graph*.

EdgeCost

(See source code in appendix D.7.12 (header), D.7.13 (implementation)).
The `EdgeCost` is a class for keeping track of different costs for edges. It has tree types of costs: *travel time; barriers; other*. The *travel time* cost represents the time it takes to travel the edge, and is thus dependent on the *length, speed limit or road category* and *surface*. The *barrier type* is for costs that comes from slowdowns imposed by barriers such as *speed bumps, gates* and such. The *other* cost are for slowing down for *signs, traffic signals, zebra crossings* and the likes.

A note on stop signs: they are associated with a road. But it is generally only applicable in one direction. For example the stop sign only affects the incoming edge in a junction, not the reverse direction of the same road going out of the junction. Therefore one needs to look at the position of the stop signs and find out which junction it really belongs to, and then only apply the cost to the affected edge. This is not implemented yet, so at the moment edges in both directions of roads with signs have costs added, which is faulty behavior.

TurnCostCalculator

(See source code in appendix D.7.16 (header), D.7.17 (implementation)).
When calculating the *costs* or *weights* for a *line* in the *line graph* it is the cost for the *source node/edge* plus the cost for the *turn*. This `TurnCostCalculator` helps with that. The calculations for this has been re-factored out to its own class as one can imagine wanting to include different properties when calculating the cost. Thus it would make sense to make this an interface and add different implementing classes, but this project just has this one implementing class for now, and therefore skipped the interface.

The inspiration for the calculations made by this calculator comes from [7], but not all factors in that paper are included here.

It is obvious that it is more costly to make a sharp turn, as one needs to decelerate coming in to the turn, and accelerate going out of the turn. The sharper the turn, the slower one needs to go. The deceleration and acceleration characteristics are properties of the routed vehicle. Also if one is coming from a lower ranking road category and is turning into a higher category, one needs to give way, which is also a cost.

EdgeRestriction

(See source code in appendix D.7.14 (header), D.7.15 (implementation)).
Restrictions for edges/roads can be somewhat complicated. Some regulates *general access*[1] with values such as *yes, no*, but in addition much more arbitrary values such as *permissive; designated; discouraged; customer*. Then other restricts access depending on the *vehicle type* such as banning *cars* but allowing *buses*. Then again, the restriction can depend on the *vehicle properties* such as *weight* or *width*. In some cases, such as *sump buster barrier*[2], it can ban access for a car, but only impose a cost on a bus. A road can also be tagged as *disused*, which is clear, but it is not so clear what to do with a road marked as *no-exit*.

An edge might no have a restriction by itself, but be part of a *turning restriction relationship*, so that one can not turn from one edge to another, although traffic is allowed on both edges. In addition, the specifications (see appendix A) said that *conditional restrictions*

[1] http://wiki.openstreetmap.org/wiki/Key:access
[2] http://wiki.openstreetmap.org/wiki/Tag:barrier%3Dsump_buster

should be respected, that is restrictions that only apply for example at a certain time, a certain day of the week, for vehicles with certain properties or of a certain category, see figure 4.5.

The *conditional restrictions* has not been implemented yet, and the whole class is marked by being developed incrementally while discovering how many separate and complex parts of *OpenStreetMap* represents some kinds of restrictions.

Figure 4.5: Conditional restrictions. [14]

4.2.5 `lgu`

The `lgu` package (see appendix listing C.6) is the *entry point* into the whole software module. The specification (appendix A) said that the module should be called from a function `get_directed_line_graph()`. This has not been written yet, so the entry point is by instantiating a `LineGraphUtility` object and call `get_line_graph()` on it, but it would be simple to write a wrapper to actually provide the specified function if needed.

The package is really only one class, `LineGraphUtility`, and how it works has been described in section 4.1.1.

4.2.6 `mapprovider`

This package (see appendix listing C.7), should contain sub-packages, as the `mapprovider` directory otherwise only contains an interface, `MapProvider`, and an exception class. The interface is the way to get map data from a source (such as a database) into the classes of the application.

There are two sub-packages in the project. One is `jsontest`, which in the initial phases of the project was used to load a small set of edges and vertices from a `json` file. It has been abandoned after loading from database was developed, but still hangs around.

The other sub-package is `postgis`, which is a map provider that uses a *PostGIS* database with the `postigs_topology` extension as the source for map data. This is where a lot of development has taken place during this project.

`postgis`

The `postgis` package uses the `libpqxx` to work with the *PostGIS* database. The `PostGisProvider` class gets passed in a `Topology` object to modify when asked for a *topology* or to set *restric-*

tions and costs. It also knows how to persist the *lines* and *nodes* of a *line graph* back to the database, which was desired functionality in the specification (see appendix A).

All the logic to work with the database and how to fill in the topology exists in this package. To make it more manageable, the `PostGisProvider` has four helper classes to actually perform the queries and handle the results from the database. They have names that describes their area of work: `CostQueries; LineGraphSaveQueries; RestrictionQueries; TopologyQueries`. They are all *static* classes and cannot be instantiated, one can only call the methods statically.

Some remarks about those classes:

The `TopologyQueries` simply fetches the relevant data for vertices and edges. For the latter case, it also performs some calculations in the *SQL* query to calculate the geometric data.

The `LineGraphSaveQueries` creates a new schema and table and inserts some basic information about the *nodes* and *lines*.

The `RestrictionQueries` has to extract all the different information for *edge restrictions* (see section 4.2.4). It uses an inner class for *turning restrictions* to work with those queries and to extract `OsmTurningRestriction` data (see section 4.2.7) so those restrictions can be resolved. Turning restrictions are not really attributes of edges, but *relations* in the *OpenStreetMap*, and the `osm2pgsql` tool for importing *osm* data into a *PostGis* database does not really handle relations so they can be used straightforwardly[3]. Therefore some workarounds have been made: In the process of initializing the database on creation a `turning_restrictions` is created and a couple of custom *sql* functions are installed (see appendix D.13.3 and D.13.2), that extract *turning restrictions* relations from the table `planet_osm_rels`, and parses what kind of restriction it is and the *osm ids* of the members (i.e. the edges and vertex involved). With those *ids* the involved *topology edges* are identified and stored as a string as the that is easier to make use of in the program than an array. The result are stored in the `turning_restrictions` table, and when running the `RestrictionQueries` for turning restrictions the topology ids are parsed and operation can continue.

4.2.7 `osm`

This package (see appendix listing C.8) deals with handling concepts and constants in *OpenStreetMap* data, such as enumerating the different categories of *accesses*[4], *barriers*[5], *highways*[6] and *vehicles*[7].

OsmTurningRestriction

In addition to those classes above, there is a class for dealing with the concept of *turning restrictions*, which are *relations* between *edges* and *vertices* in an *OpenStreetmap*. This class is an attempt to keep track of that information. In *OSM* a turning restriction is a relation of *(from → via → to)*. The 'via' part can be either a vertex (at a junction) or other edges, saying "travel from Here to There via roads This and That are not allowed". That kind of relationship is a lot trickier to represent, especially for this software module that only should build a *line graph* of the allowed turns, but has no routing information and thus cannot decide if a "via way" relation is allowed or not. It has therefore been disregarded in this project, and a routing application needs to decide that information some other way. The class `OsmTurningRestriction` has a field telling if it is a *via way* or a *via vertex* restriction.

4.2.8 `preparation`

Before anything else can be done, one needs to prepare the database. That means installing needed extensions to handle geometric and geographic data, and set up some tables and functions needed. Then one can add the map data to the database.

[3] http://wiki.openstreetmap.org/wiki/Osm2pgsql/schema
[4] http://wiki.openstreetmap.org/wiki/Key:access
[5] http://wiki.openstreetmap.org/wiki/Key:barrier
[6] http://wiki.openstreetmap.org/wiki/Key:highway
[7] http://wiki.openstreetmap.org/wiki/Key:vehicle

Appendix listing C.9 show the contents of this package. There is an `.sql` file (see listing D.13.3) for initializing extensions `postgis; postgis_topology; hstore` and installing functions for finding *turning restrictions*. And there is a `.sql` file to use when building the topology in advance. Then there is a file `LGU.style` which tells `osm2pgsql` which tags to create columns for in the tables, and which tags to ignore. Then there is also the original `.osm` files with map data for *Mikhailovsk* and *Partille* (they are not included in the appendix, but there are instructions how to download them there, see appendix listing D.1 and D.2).

The way to prepare the database is shown in listing 4.4, which sets up a new database `mikh_db` for the *Mikhailovsk* map data.

```
$ # 1. Create database
$ createdb mikh_db -U tester

$ # 2. Install extensions and functions
$ psql -U tester -d mikh_db -f init_osm2pgsql_postgis_topology.sql

$ # 3. Import OSM data
$ # Flags:  -s    Slim mode (add data to db, do not build all in memory)
$ #         -k    Keep tags in `hstore` if not in own column
$ #         -S    Style-file to use
$ osm2pgsql -U tester -d mikh_db -s -k -S LGU.style mikhailovsk.osm

$ # 4. Building topology (optional)
$ psql -U tester -d mikh_db -f build_postgis_topology.sql
```

Listing 4.4: Preparing a database with map data.

4.2.9 `util`

This package (see listing in appendix C.10) contains a few utility classes: one for *logging* (using *Boost logging*) to be used where needed in the application; one for a *coordinate point* and one for *producing strings from current timestamp* which is used when building temporary topologies.

4.3 Development environment

Development of the project and the coding has taken place in *Eclipse Luna 4.4.2*. The build system is the default in Eclipse on Linux, generating *makefiles*.

Settings:

- Compiler flags:
  - `std=c++11`
  - `DBOOST_LOG_DYN_LINK`
  - `O0`
  - `g3`
  - `Wall`
  - `c`
- Linker flags:
  - `lboost_log`
  - `lboost_log_setup`
  - `lboost_thread`
  - `lboost_system`
  - `lpthread`
  - `lpqxx`

– `lpq`

The coding was to follow a *coding standard* (see appendix A.7) which regulates the naming scheme and the layout of the files.

As for working with the database the main tool has been *pgAdmin3*.

5 Results

The software module developed in this project does not fulfill all requirements (see appendix A), in that it does not handle *conditional restrictions* at all, and not all implemented restrictions are handled correctly, see section 5.1 below.

But the software does build a *line graph* that can be fetched and stored in database for inspection with visual tools, see section 5.2 below.

The project has so far been about implementing things and have not had any focus on performance, but some performance tests have been run, see section 5.3.

5.1 Specification fulfillment

Table 5.1 shows how much of the specification that has been fulfilled.

Table 5.1: Fulfillment of specification.

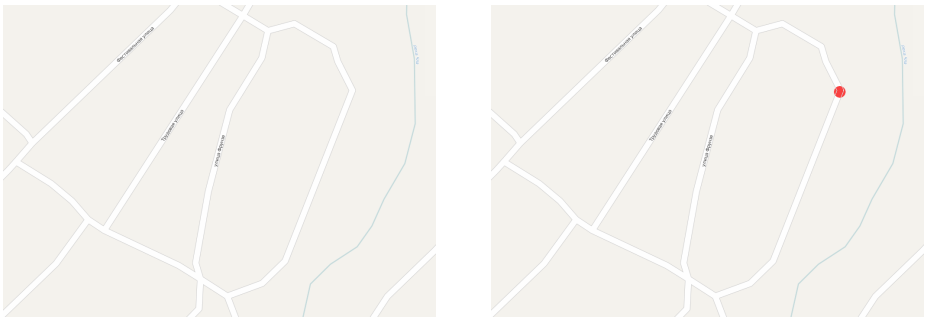
Section	Fulfills	Comment
1.2 Main use case		
1.2.1	X	As call to `LineGraphUtility::getLineGraph()`.
1.2.2	X	
1.2.3	X	
1.2.4	X	
1.3 Optional use case		
1.3.1	X	
1.3.2	X	
1.4 Functional requirements		
1.4.1		Lots of work remains to implement all restrictions.
1.4.2	X	
1.4.3	X	
1.4.4	X	Some small parts are hard coded.
1.5 Non-functional requirements		
1.5.1	X	Written in C++.
1.5.2	X	Did not find pgRouting really useable.
1.6 Testing requirements		
1.6	X	
1.7 Coding standard		
1.7	X	

5.2 Visual examination

Maps are easy to visualize, and a great number of tools exist to work with map data. Figure 5.1 shows a piece of a map over Mikhailovsk. In order to test if the handling of restrictions work, modified maps have been created. *JOSM*[1] is a tool for manipulating *OSM* data. In figure 5.1b it is indicated where a *bollard barrier* has been added in the middle of a road,

[1] https://josm.openstreetmap.de/

just to see if the restrictions work, and the new map is saved in its own `.osm` file, and a new database built for it.



(a) Original. (b) Modified with added barrier.

Figure 5.1: Map over part of Mikhailovsk. [15]

Using another tool, *QGIS*[2] can be used to load map data from for example a *PostGIS* database and viewed. In figure 5.2 the vertices and edges from the topology for that map has been layered on top of the image (with a slight misalignment). The topology is the same for both maps, it does not change with added barriers.

Figure 5.2: Edges (green) and vertices (blue) for the topology.

The interesting part is to see if the restriction has had any impact on the built *line graph*, see figure 5.3 for the original line graph, where the road is included in the line graph. It has a *node* in the middle and *lines* connecting to the adjacent *edges/nodes*.

Figure 5.4 shows the line graph after the restricting barrier has been added to the map. There one can see that the *edge* (road) has not been added as a *node* to the line graph, while all the other *lines* and *nodes* remain the same. This practically disables routing along that road.

[2] http://www.qgis.org

Figure 5.3: Original *line graph*. *Lines* in purple.

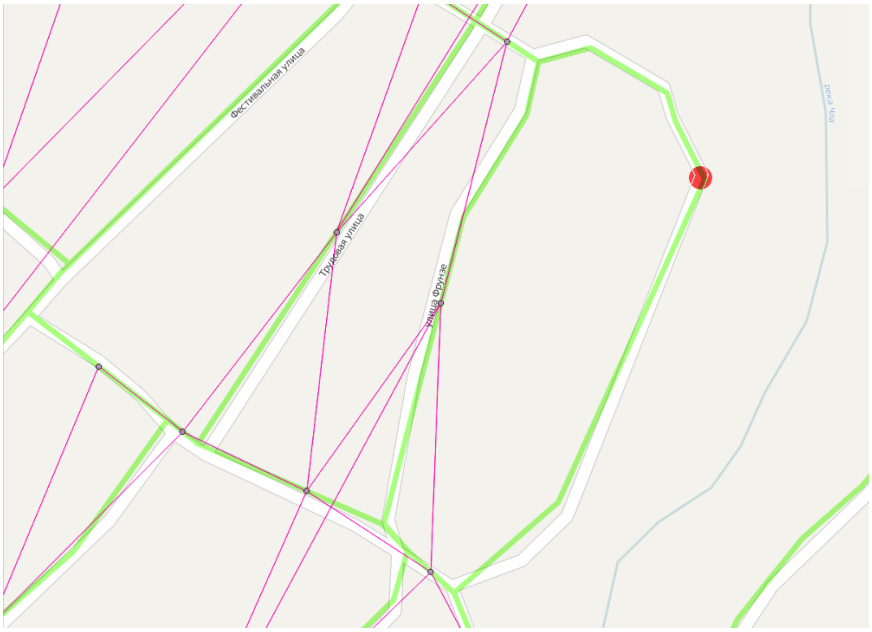


Figure 5.4: *Line graph* after added barrier. *Lines* in magenta.

5.3 Performance

There were *soft real time* requirements in the specification, but they were not specified more than that. But it is interesting so find out how long it takes to fetch a *line graph*, built on demand by the software module.

A few test cases were written in `LineGraphUtility_test.cc` that averages the number of *microseconds* it takes to instantiate a `LineGraphUtility` and fetch a *line graph*, over a given number of rounds.

The test runs on both a configuration with a pre-built topology, and a configuration that builds a temporary topology.

See table 5.2 for test results.

Table 5.2: Time in μs to fetch a *line graph*, with pre-built versus temporary topologies.

Test #		1	2	3	4	Sum
# of rounds		10	10	10	70	100
topology		avg (μs)	avg (μs)	avg (μs)	avg (μs)	avg (μs)
Mikhailovsk	normal	147859	149092	141782	133950	143171
	temporary	5026626	4924245	4917319	4875838	4936007
Partille	normal	180340	188405	179883	179978	182152
	temporary	10683194	10342420	10683873	10521535	10557756

The *size* (number of edges) and *order* (number of vertices) of the graphs are shown in table 5.3.

Table 5.3: Sizes of tested graphs

	Graph		Line graph	
	vertices	edges	nodes	lines
Mikhailovsk	654	1618	1618	4758
Partille	1645	2265	2265	5577

The results shows that, in order to meet *soft real time requirements* it is not possible to build temporary topologies at every instantiation of a `LineGraphUtility`, as the time increases dramatically. In the case of *Mikhailovsk*, the increase is from 0.14 s to 4.93 s, about 34 times as much. In the case of *Partille*, the increase is from 0.18 s to 10.55 s, nearly 58 times.

That fetching a line graph with a pre-built topology takes 0.15-0.2 s might fall within the requirements.

The test were conducted on a computer with 8 GB ram, processor Intel i7-4702MQ, running Linux Mint 17.1 with Linux kernel version 3.13.0-37-generic. The comiler flags were the same as for the rest of the project, i.e. no optimization.

6 Discussion

Presented below is my personal views of parts of the project and the outcome of it.

6.1 Research

6.1.1 Graph theory

Starting out on this project, I thought that one of the main obstacles would be no prior knowledge of *graph theory*, so I set out to allow for some time initially to get into the field. I am glad to have gained some fundamental knowledge of the area, but the time spent here could have been less.

6.1.2 Map routing

Reading about theory regarding map routing and graphs was really interesting, and a lot of research has been done in this area in later years. It initially gave me some ideas I thought I would like to try out, but once development got going, those theories vanished in favor of finding working solutions quickly.

6.1.3 Map data

OpenStreetMap is the source of map data for this project, and a lot of high quality projects. That puzzles me somewhat, because I have found it kind of messy. It is an *XML* application, but it has no official *schema*. That is, there is an informal consensus on which tags are good, but one can also make up ones own tags[1]. Another example of the messiness is the `maxspeed` tag, which can have the values *60; 50 mph; 10 knots*. That is, the default case is a unit of *km/h* and one can read the value as numeric. But one cannot be sure of that, because other units are allowed, and in that case one needs to parse the value as a string to find out which unit is used. It would surely have been better to let the unit be an attribute of the value, so one did not need to parse every value. In this project I decided to skip parsing, and just assume all values are *km/h*.

But, as said, a lot of good applications using *OSM* exists, see 2.4, so it is possible to work with. And it might also be unfair to say that *OpenStreetmap* is messy; it might be the case that it simply reflects the complex and difficult reality in the traffic, with lots of different rules and restrictions dependent on context and conditions.

6.1.4 Available applications

The fact that a lot of applications already existed, and some of them being *open source* and using *OpenStreetMap* as the the source for map data, made the direction of this project a little difficult. I proposed to the company that there are some good solutions out there that might just need some adaption to work, but they wanted their own thing. So the question for me was if I was to look at and copy features and concepts of those existing solutions anyway or just blindly go down my own path. In order to steer clear of issues with plagiarism I chose the latter, and that has surely impacted the project negatively. It would have been wiser to build upon the experience of others, developed through years.

6.2 Methodology

The main methodology for the project was supposed to be test driven (either BDD or TDD), but to be honest, most tests were written after the implementation of a feature, functioning

[1] http://wiki.openstreetmap.org/wiki/Map_Features

more as unit test, than driving the development. I don't think that has affected the outcome of the project negatively, it is more a matter which workflow feels best.

6.3 Design

Previously I have bee more into heavy design and modeling before starting coding, but in the last year I have tried to become more "agile", and start testing things out and be prepared to refactor and remodel when needed.

In this project perhaps it could have been good to design more, to have really thought through how the restrictions should work. On the other hand, a lot of the difficulties was discovered only when working on them, so it would be hard to have the full picture before. It is a balance in getting going and learning, and modeling before. What is clear, is that parts of the software as it stands now, should be re-modeled, specifically the *restrictions*.

6.4 Development

6.4.1 Coding standard

This was the first time I had to code to a standard. It was kind of awkward and unintuitive at first as it differs from my personal style, especially since having started to trying to practice *"Clean Code"*. and have less comments and visual dividers in the file. But after a short time the style became pretty easy to use. I don't think I have followed the standard completely, but it was too lng to read and get into before beginning to code.

6.4.2 Memory management

I tried to avoid pointers and only use references, but that turned out to be clumsy, so at times I reverted to using raw pointers. Eventually, I found out that it would have been a lot better to use the smart pointers from C++11 (or even Boost), but I did not want to spend the time needed for learning how to use them and redo the memory management completely.

6.4.3 Tools

OSM conversion

I tested and looked at a number of tools for converting *OpenStreetMap* data to a *PostGIS* database, and the choice fell on *osm2pgsql*. I am not certain that it was the right choice, as it has is shortcomings when working with restrictions. Fortunately, the developed software module is flexible so one can write a new *MapProvider* if one decides to work with another tool, that uses a different approach.

Database

The *pqxx* library was easy and straight-forward to work with.

Boost

This was the first time for me to use *Boost*. I have only used small parts of the library: obviously the *graph* package, the *property_tree* for parsing *json* and the *logging* package. There are some tricky concepts, but also a lot of useful stuff. Getting into all the long names and templates takes some getting used to, but it was OK.

Catch test

I really enjoyed the *Catch* testing library; small and easy to use. It didn't play so nicely with *Eclipse CDT*, marking errors throughout in the editor, but good enough.

6.4.4 OpenStreetMap

Restrictions

Turning restrictions are *relations*, and *osm2pgsql* does not really handle relations, so a lot of parsing was needed. And in the case of *conditional restrictions* I have not found out how to work with them. *osm2pgsql* can be instructed to put tags into separate columns in the database, but with conditional restrictions the tags changes 'looks' and the only way to find them is by parsing the *hstore* column.

Also, the *restriction* class in the application is kind of messy. It could do with some remodeling, partly to clean up, and partly to make it more extensible to incorporate *conditions*. The *OSM* syntax for *conditional restrictions*[2] is shown in listing 6.1, and could work as a model for developing a more generic restrictions class.

```
<restriction-type>[:<transportation mode>][:<direction>]:conditional
  = <restriction-value> @ <condition>[;<restriction-value> @ <condition>]
```

Listing 6.1: Syntax of conditional restrictions in OpenStreetMap.

6.5 Documentation

A lot of the time of the project has also been devoted to documentation and writing this report. I took the opportunity to learn how to write a report in LaTeX, using the excellent web service www.sharelatex.com. It has been a pleasure, and it feels really good not to depend on the shaky features with cross-referencing in word processors.

6.6 Results

As the project does not fulfill all requirements and did not finish on time, it was not all that successful. The reason for not meeting the specification is that I ran out of time, partly due to the specification was supplied more than two weeks late, and partly due to the complexities with handling restrictions.

It would be possible to continue development, most on finding good ways to handle conditional restrictions. From my horizon, I still think that the best solution would be to adapt an existing solution that has been developed and refined by many people through many years. Perhaps using *OSRM* together with a *PostGIS* database as demonstrated here: https://www.mapbox.com/blog/osrm-using-external-data/. But I do not have any overview of the greater project, and what it is trying to accomplish.

This project shows that there exists really good products, and that rolling ones own is not trivial. What first seemed like a straightforward sequential piece of software turned out to be tangled in complex handling of restrictions.

[2] http://wiki.openstreetmap.org/wiki/Conditional_restrictions

Bibliography

[1] Keijo Ruohonen. *Graph Theory*. Lecture notes. Tampere University of Technology, 2013. URL: http://math.tut.fi/~ruohonen/GT_English.pdf (visited on 2015-03-24).

[2] Tero Harju. *Graph Theory*. Lecture notes. University of Turku, Departement of Mathematics, 2012. URL: http://users.utu.fi/harju/graphtheory/graphtheory.pdf (visited on 2015-04-09).

[3] Reinhard Diestel. *Graph Theory*. 4th Electronic Edition 2010, Corrected Reprint 2012. 2012. URL: http://diestel-graph-theory.com/ (visited on 2015-03-25).

[4] Daniel Delling et al. "Engineering Route Planning Algorithms". English. In: *Algorithmics of Large and Complex Networks*. Ed. by Jürgen Lerner, Dorothea Wagner, and KatharinaA. Zweig. Vol. 5515. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 117–139. ISBN: 978-3-642-02093-3. DOI: 10.1007/978-3-642-02094-0_7. URL: http://i11www.iti.uni-karlsruhe.de/extra/publications/dssw-erpa-09.pdf (visited on 2015-04-12).

[5] Hannah Bast et al. *Route Planning in Transportation Networks*. Tech. rep. MSR-TR-2014-4. Jan. 2014. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=207102 (visited on 2015-04-11).

[6] Daniel Delling et al. "Customizable Route Planning in Road Networks". working paper, submitted for publication. 2013. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=198358 (visited on 2015-04-05).

[7] Lars Volker. *Route Planning in Road Networks with Turn Costs*. Student research project, Universität Karlsruhe (TH), supervised by P. Sanders and D. Schultes. Universität Karlsruhe, 2008. URL: http://algo2.iti.kit.edu/documents/routeplanning/volker_sa.pdf (visited on 2015-04-03).

[8] Robert Geisberger and Christian Vetter. "Efficient Routing in Road Networks with Turn Costs". In: *Proc. 23rd International Symposium on Distributed Computing (DISC, Elche, Spain, September 2009)*. Vol. 5805. Lecture Notes in Computer Science. Springer. URL: http://algo2.iti.kit.edu/download/turn_ch.pdf (visited on 2015-04-13).

[9] Alexandros Efentakis et al. "Crowdsourcing turning restrictions for OpenStreetMap." In: *EDBT/ICDT Workshops*. 2014, pp. 355–362. URL: http://ceur-ws.org/Vol-1133/paper-56.pdf (visited on 2015-04-14).

[10] QGIS. *Topology*. URL: https://docs.qgis.org/2.2/en/docs/gentle_gis_introduction/topology.html (visited on 2015-10-05).

[11] Ittai Abraham et al. "HLDB: Location-based services in databases". In: *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM. 2012, pp. 339–348. URL: http://research-srv.microsoft.com/pubs/172429/hldb_GIS12.pdf (visited on 2015-04-14).

[12] Ittai Abraham et al. *HLDB: Location-Based Services in Databases*. Tech. rep. MSR-TR-2012-59. June 2012. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=166857 (visited on 2015-04-14).

[13] Doug Gregor. *Bundled Properties*. URL: http://www.boost.org/doc/libs/1_54_0/libs/graph/doc/bundles.html (visited on 2015-10-08).

[14] "Achadwik". *File:Length and time restriction 2.jpg*. [License: Creative Commons Attribution-ShareAlike 2.0]. 2011. URL: http://wiki.openstreetmap.org/wiki/File:Length_and_time_restriction_2.jpg (visited on 2015-10-15).

[15] OpenStreetMap contributors ©. *OpenStreetMap*. [License: Creative Commons Attribution-ShareAlike 2.0]. 2015. URL: https://www.openstreetmap.org/#map=18/45.13903/42.03324 (visited on 2015-10-22).

A Specification

The complete specifications from the company.

A.1 General

Line Graph Utility, LGU is a software utility which can poll a PostGIS database for a road network and builds a directed line graph from that. The directed line graph is stored in memory and the call to `get_directed_line_graph()` returns a directed line graph stored in a C++ *Boost* graph structure. The directed line graph is built based on the time of the day, road signs, traffic lights and other conditions.

A.2 Main use case

A.2.1 Call `get_directed_line_graph()` from C++ code.

A.2.2 LGU queries the PostGIS database and builds a graph from the road network.

A.2.3 LGU builds a directed line graph from the previous step (i.e. it converts nodes to edges and assigns weight to those edges based on road signs and other elements which are present in the nodes).

A.2.4 `get_directed_line_graph()` returns a directed line graph structure which is based on a C Boost graph structure to the function caller.

A.3 Optional use case

A.3.1 All main use case steps.

A.3.2 Write the resulting directed line graph to a separate heterogeneous table in the PostGIS database so that the graph can be viewed in QGis.

A.4 Functional requirements

A.4.1 LGU should take into account the following elements when building a directed line graph and calculating a weight for each edge: road signs (including time scheduling for those), traffic lights, road type (OSM road types), time of the day, road marking (i.e. separate lanes should be treated as separate edges), crossing and roundabouts slowdown, slopes and downhills, one way streets, road conditions, 'closed road' attribute.

A.4.2 LGU should take into account restricted turns in the road network when building a directed line graph; i.e. it should not create edges between newly created nodes in a line graph.

A.4.3 LGU should only take road signs and other conditions which are already present in the PostGIS database, the database is the only source of data for LGU.

A.4.4 LGU should store all its settings in a `settings.json` file.

A.5 Non-functional requirements

A.5.1 LGU should be written in C/C++; or, Boost.Python can be used

A.5.2 LGU can re-use architecture and code from the pgRouting software, which has a very similar structure. Namely it can re-use the steps 1 and 2 of the pgRouting's source code:

- A C module that uses a query is passed in Postgresql in order to build a line graph.
- C++ modules that convert it into a boost graph, and launch the routing.
- Return a result into psql server (this step is not required)

A.6 Testing requirements

LGU should be tested with a road network map built from 2 `.osm` files, `partille.osm` and `mikhailovsk.osm`.

A.7 Coding standard

Not actually written down in this document, but noted in an earlier conversation was that the company uses a *coding standard* [1] that must be followed.

[1] http://www.possibility.com/Cpp/CppCodingStandard.html

B UML Diagrams

Sequence and class diagrams of the software module.

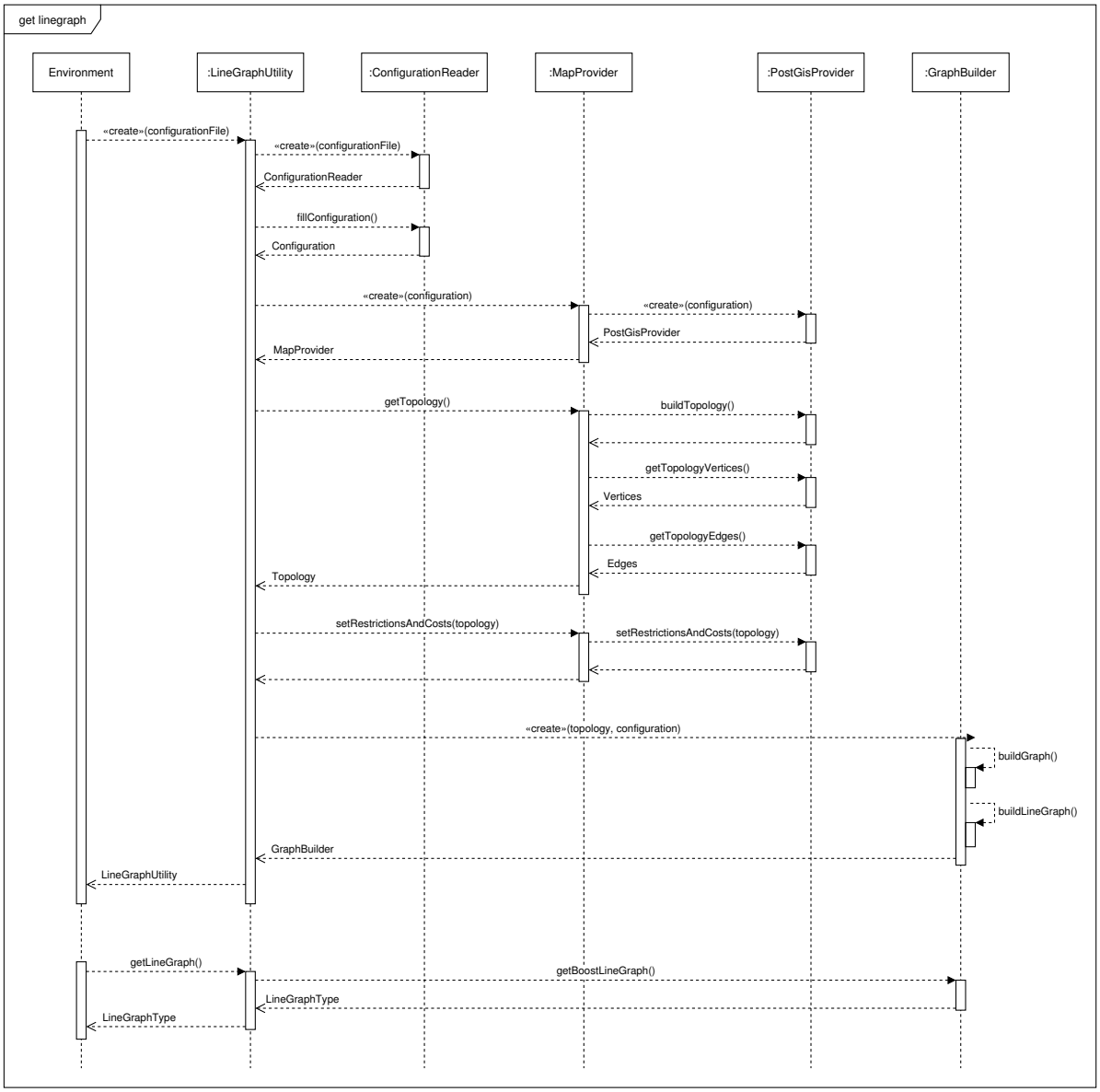


Figure B.1: Sequence diagram of main use case to get a line graph.

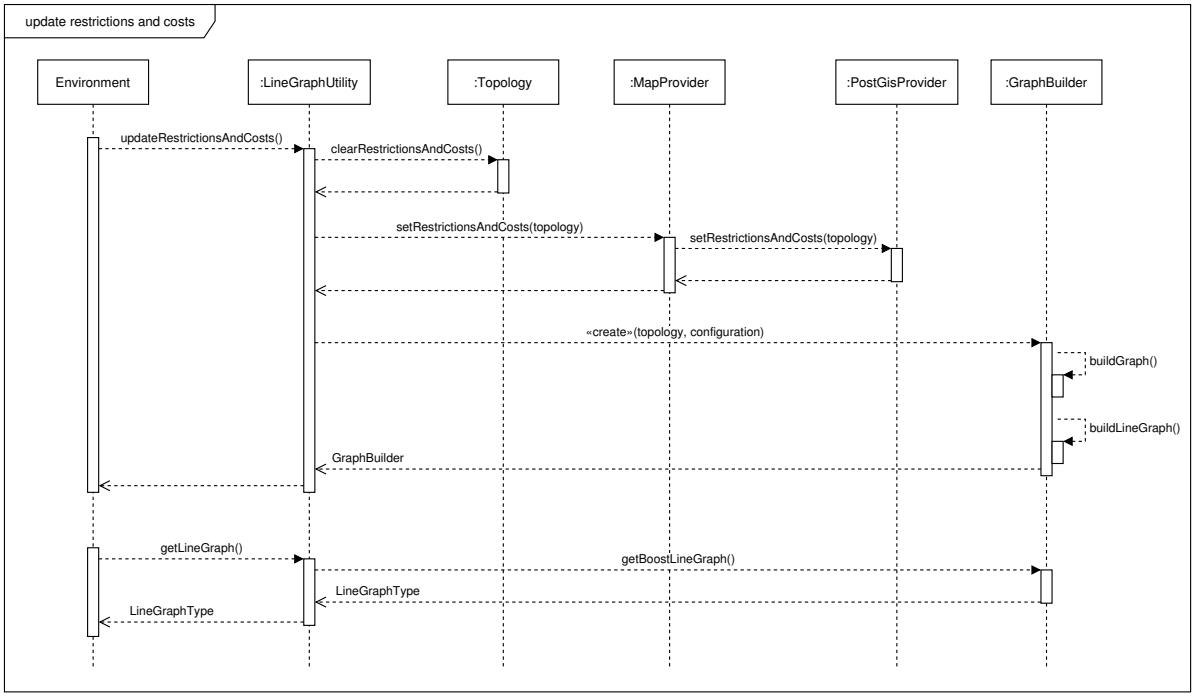


Figure B.2: Sequence diagram of updating costs and restrictions on a topology.

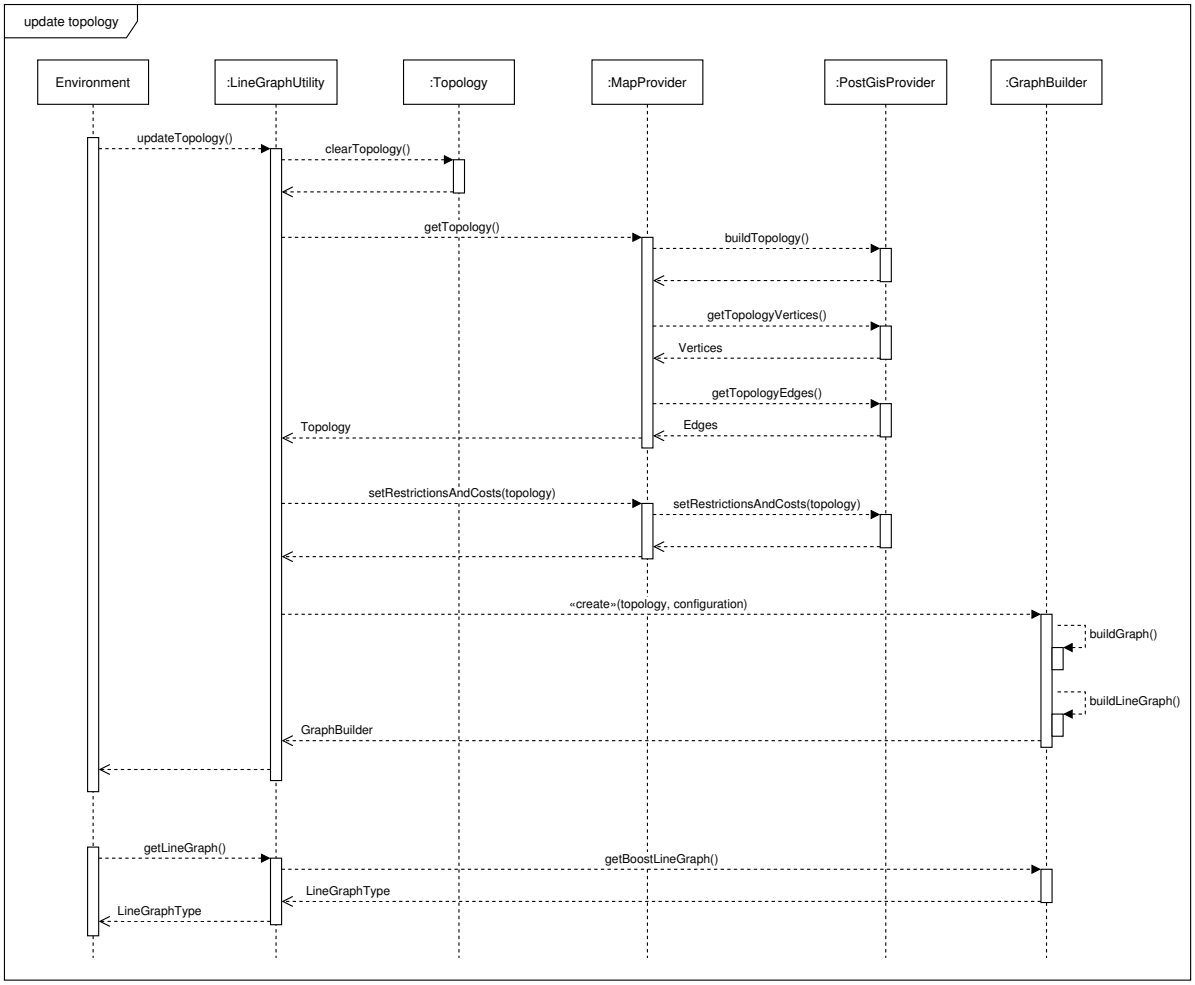


Figure B.3: Sequence diagram of updating the topology.

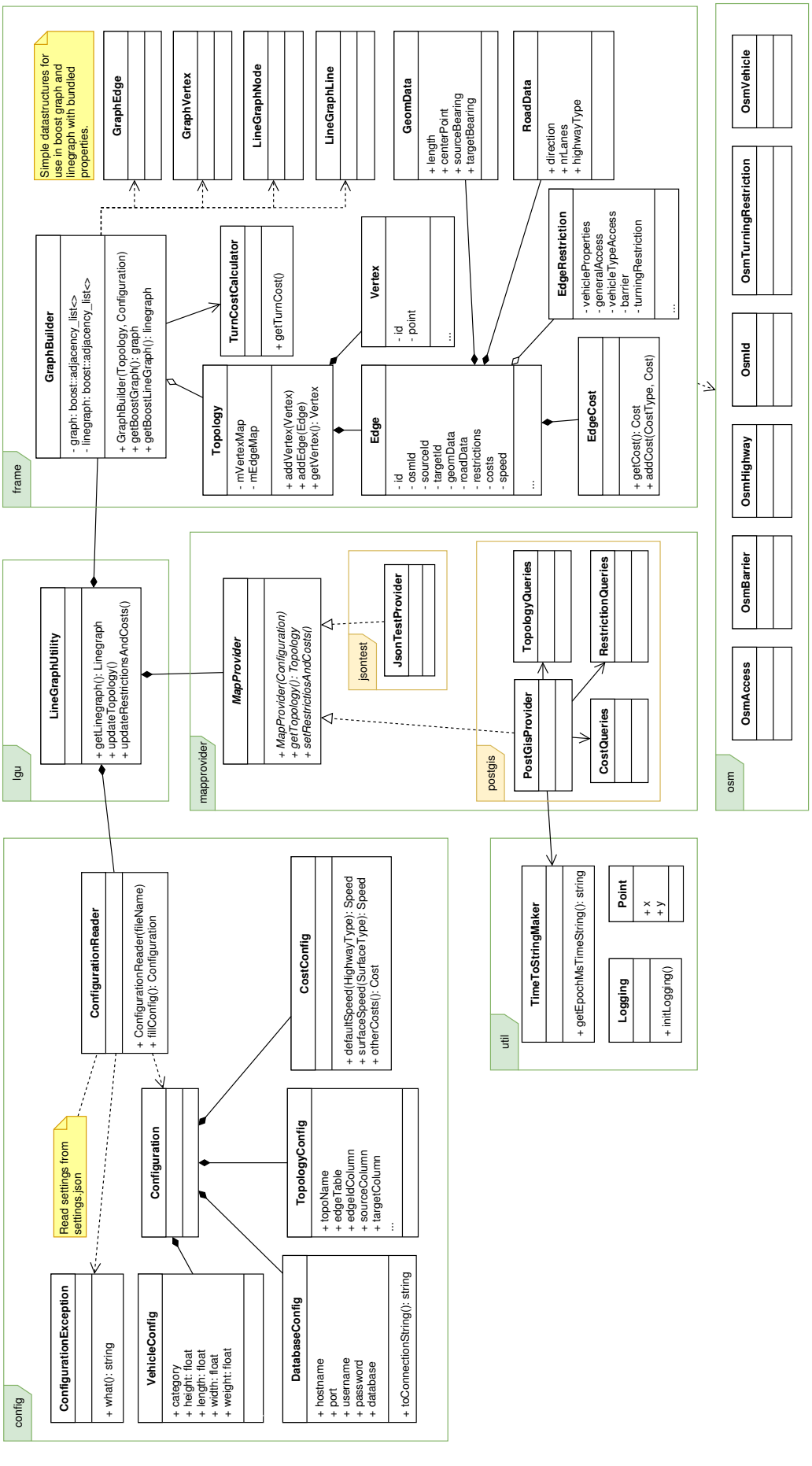


Figure B.4: Class diagram of the *line graph utility*.

C Directory listings

Contents of the directories in this project.

```
/
├── catchtest/
├── config/
├── doc/
├── graph/
├── lgu/
├── mapprovider/
├── osm/
├── preparation/
├── util/
└── README.md
```

Listing C.1: Directory structure in root of the project

```
/
└── catchtest/
    ├── catch.hpp
    ├── catchmain.cc
    ├── README.md
    └── testsettings/
        ├── mikhailovsk-original.json
        ├── mikhailovsk-original-temp.json
        ├── partille-original-temp.json
        ├── partille-original-temp.json
        ├── . . . ..... (10 more .json files)
        └── restrictions/
            ├── mikhailovsk-barrier_block.json
            ├── partille-highway_traffic_signals.json
            └── . . . ..... (17 more .json files)
```

Listing C.2: Files in `/catchtest`

```
/
└── config/
    ├── catchtest/
    │   └── ConfigurationReader_test.cc
    ├── Configuration.cc
    ├── Configuration.h
    ├── ConfigurationException.h
    ├── ConfigurationReader.cc
    ├── ConfigurationReader.h
    ├── CostConfig.h
    ├── DatabaseConfig.h
    ├── README.md
    ├── TopologyConfig.h
    └── VehicleConfig.h
```

Listing C.3: Files in `/config`

```
/
└── doc/
    ├── presentation/
    ├── report/
    │   ├── latex/ ..... LaTeX original
    │   ├── jobe0900_report.pdf
    │   └── README.md
    ├── uml/
    │   ├── ..... .xml, .svg, .pdf diagram files
    │   └── README.md
    └── README.md
```

Listing C.4: Files in `/doc`

```
/
└── graph/
    ├── catchtest/
    │   ├── EdgeCost_test.cc
    │   ├── GraphBuilder_test.cc
    │   ├── RestrictionsAndCosts_test.cc
    │   ├── Topology_test.cc
    │   └── TurnCostCalculator_test.cc
    ├── Cost.h
    ├── Edge.cc
    ├── Edge.h
    ├── EdgeCost.cc
    ├── EdgeCost.h
    ├── EdgeRestriction.cc
    ├── EdgeRestriction.h
    ├── GraphBuilder.cc
    ├── GraphBuilder.h
    ├── GraphException.h
    ├── README.md
    ├── RestrictionsException.h
    ├── Speed.h
    ├── Topology.cc
    ├── Topology.h
    ├── TopologyException.h
    ├── TurnCostCalculator.cc
    ├── TurnCostCalculator.h
    ├── Vertex.cc
    └── Vertex.h
```

Listing C.5: Files in `/graph`

```
/
└── lgu/
    ├── catchtest/
    │   └── LineGraphUtility_test.cc
    ├── LineGraphUtility.cc
    ├── LineGraphUtility.h
    ├── LineGraphUtilityException.h
    └── README.md
```

Listing C.6: Files in `/lgu`

```
/
└── mapprovider/
    ├── catchtest/
    │   └── MapProvider_test.cc
    ├── jsontest/
    │   ├── catchtest/
    │   │   ├── JsonTestProvider_test.cc
    │   │   ├── jsontest-settings.json
    │   │   └── test-topology.json
    │   ├── JsonTestProvider.cc
    │   ├── JsonTestProvider.h
    │   └── README.md
    ├── postgis/
    │   ├── catchtest/
    │   │   └── PostGisProvider_test.cc
    │   ├── CostQueries.cc
    │   ├── CostQueries.h
    │   ├── LineGraphSaveQueries.cc
    │   ├── LineGraphSaveQueries.h
    │   ├── PostGisProvider.cc
    │   ├── PostGisProvider.h
    │   ├── RestrictionQueries.cc
    │   ├── RestrictionQueries.h
    │   ├── TopologyQueries.cc
    │   └── README.md
    ├── MapProvider.h
    ├── MapProviderException.h
    └── README.md
```

Listing C.7: Files in `/mapprovider`

```
/
└── osm/
    ├── catchtest/
    │   ├── OsmAccess_test.cc
    │   ├── OsmBarrier_test.cc
    │   ├── OsmHighway_test.cc
    │   ├── OsmTurningRestriction_test.cc
    │   └── OsmVehicle_test.cc
    ├── OsmAccess.cc
    ├── OsmAccess.h
    ├── OsmBarrier.cc
    ├── OsmBarrier.h
    ├── OsmException.h
    ├── OsmHighway.cc
    ├── OsmHighway.h
    ├── OsmId.cc
    ├── OsmId.h
    ├── OsmTurningRestriction.cc
    ├── OsmTurningRestriction.h
    ├── OsmVehicle.cc
    ├── OsmVehicle.h
    └── README.md
```

Listing C.8: Files in `/osm`

```
/
└── preparation/
    ├── restrictions/
    │   └── . . . ..... osm-files modified with added restrictions
    ├── build_postgis_topology.sql
    ├── init_osm2pgsql_postgis_topology.sql
    ├── LGU.style
    ├── mikhailovsk.osm
    ├── partille.osm
    └── README.md
```

Listing C.9: Files in `/preparation`

```
/
└── util/
    ├── catchtest/ ..... empty directory
    ├── Logging.cc
    ├── Logging.h
    ├── Point.h
    ├── TimeToStringMaker.cc
    └── TimeToStringMaker.h
```

Listing C.10: Files in `/util`

D Source code

A complete repository can be found at:
https://bitbucket.org/jobe0900/exjobb/src or
https://github.com/jobe0900/exjobb.

D.1 README.md

```
LineGraphUtility (lgu)
======================
This software module uses OpenStreetMap data to fetch topology, restrictions and
 ↪  costs, and uses them to build a Graph, which is converted to a LineGraph.

## State of the software
The software module does not fulfill the specification yet.

#### Working features
- Building graph and linegraph respecting some **edge** restrictions:
    - Turning restrictions.
    - General access restrictions.
    - Vehicle type specific restrictions.
    - Vehicle property restrictions (weight, height...).
- Some restrictions on edges.
- Turning restrictions via a node, not via other roads.
- Costs.

#### NOT implemented features
- Inclination, different speed uphill or down hill.
- Conditional restrictions.
- Turning restrictions that are not one-to-one, but one-to-many.
- Turning restrictions via ways (not via nodes).
- Parsing units, i.e. assuming all dimensions are meters and weight in metric
 ↪  tons and speed in km/h.

### Organization
The code is organized in folders (kind of "packages") to keep it modularized. The
 ↪  packages are:

- **`catchtest`**: The main for the testing framework.
- **`config`**: For configuration related code.
- **`graph`**: For code that is related to Graphs.
- **`lgu`**: The main entry point into this software.
- **`mapprovider`**: The package for code providing access to map data.
- **`osm`**: Classes representing some concepts in OpenStreetMap data.
- **`preparation`**: osm-files and sql-files and instructions on how to set up
 ↪  database.
- **`uml`**: For uml documentation.
- **`util`**: A few utility classes.

Each folder should have its own `README.md` that describes what the contents and
 ↪  the purpose of that package is. Each package should also have their own tests
 ↪  in a `catchtest` folder, and preferably an *exception class*.

### Building
```

```
Right now  all development has been in Eclipse, so it is just a standard Eclipse
 ↪  project with the makefiles that Eclipse has set up in the `Debug` folder. The
 ↪  file `catchtest/catchmain.cc` provides the entry point for the software
 ↪  module during testing.

#### Libraries
There was only need for linking with `-lpqxx` and `-lpq` (for connecting to the
 ↪  database) until *Boost logging* was included, at which point it also became
 ↪  necessary to link with `-lboost_log -lboost_log_setup -lboost_thread
 ↪  -lboost_system -lpthread`.

#### Testing
As mentioned, testing is done with [Catch](https://github.com/philsquared/Catch).
 ↪  Tests can be written BDD-style, and it is header only. A few quirks: some of
 ↪  the macro-keywords, most notably `REQUIRE`, is reported as an error in the
 ↪  Eclipse editor, but one can ignore that.

### Style
I have tried to follow the style given in [C++ Coding
 ↪  Standard](http://www.possibility.com/Cpp/CppCodingStandard.html).

### Design
I have deliberately tried to avoid passing pointers around, and rather pass in
 ↪  references as IN-OUT parameters. The idea is that the central LGU class has
 ↪  stack variables of `Graph`, `Topology` that gets filled in, rather than
 ↪  obtained as pointers to objects on the heap. This is to try to reduce risks
 ↪  of complicated memory handling, while not have too much copying of large
 ↪  objects.

### Logging
Boost logging was the last feature added, and is so far only used in the `Graph`
 ↪  class. It needs to be compiled and linked with a lot of libraries:

    -lboost_log -lboost_log_setup -lboost_thread -lboost_system -lpthread

The log produced is `lgu.log` in the top level of the project.
```

D.2 catchtest

D.2.1 README.md

```
CATCH
=====
This project uses [Catch](https://github.com/philsquared/Catch) for testing. It
 ↪  allows for writing tests BDD-style.

It doesn't play really nicely with Eclipse, as Eclipse's editor marks `REQUIRE`
 ↪  as errors, so the project has a  lot of error markers throughout, without any
 ↪  real errors. But the Catch way of testing is nice, so it is worth is. And
 ↪  Eclipse flags a lot of errors for standard c++11 features as well...

When writing `SCENARIO`s or `TESTCASE`s one can tag those, which makes it easy to
 ↪  test small parts of the code. After building you can modify the Eclipse `Run
 ↪  Configuration` (or write on the command line) to only run those tests.

Example:
```

```cpp
SCENARIO ("Testing this module but not other", "[moduletag]")
{
    GIVEN ("a")
    {
        WHEN ("b")
        {
            THEN ("c")
            {
                REQUIRE (c)
            }
        }
    }
}
```

To specify which test to run, go to `Run` > `Run Configurations...`, select the `Arguments` tab and in `Program arguments` write the tag, e.g. `[moduletag]`, click `Apply` and `Run`.

A useful flag to add to the program arguments when running tests is `-s` to have print out of every step, else you only get the final report of how many scenarios have run.

D.2.2 catchmain.cc

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"
```

D.2.3 mikhailovsk-original.json

```
{
    "database":
    {
        "host":         "127.0.0.1",
        "port":          5432,
        "username":     "tester",
        "password":     "tester",
        "database":     "mikhailovsk-original"
    },

    "topology":
    {
        "provider":      "postgis",

        "postgis":
        {

            "topo_name":       "lgu",
            "roads_prefix":    "highways",
            "schema_prefix":   "topo",
            "build": {
                "temp_topo_name": "",
                "srid":           900913,
                "tolerance":      1.0
            },
            "edge":
```

```
            {
                "table":      "edge_data",
                "id_col":     "edge_id",
                "source_col": "start_node",
                "target_col": "end_node",
                "geom_col":   "geom"
            },
            "vertex":
            {
                "table":     "node",
                "id_col":    "node_id",
                "geom_col":  "geom"
            }
        },

        "pgrouting":
        {
        },

        "jsontest":
        {
            "test_file": ""
        }

    },

    "vehicle":
    {
        "category":   "motorcar",
        "motorcar":
        {
           "height":       1.6,
           "length":       4.5,
           "width":        1.9,
           "weight":       2.0,
           "maxspeed":     200,
           "acceleration": 10,
           "deceleration": 7
        }
    },

        "access":
   {
      "allow":
      [
         "yes",
         "permissive",
         "designated"
      ]
   },

   "restrict":
   {
      "barriers":
      [
         "block",
         "bollard",
```

```
            "bus_trap",
            "chain",
            "cycle_barrier",
            "debris",
            "full-height_turnstile",
            "horse_stile",
            "jersey_barrier",
            "kent_carriage_gap",
            "kissing_gate",
            "log",
            "motorcycle_barrier",
            "rope",
            "sally_port",
            "spikes",
            "stile",
            "sump_buster",
            "swing_gate",
            "turnstile",
            "yes"
        ]

    },

    "cost":
    {
        "default_speed":
        {
            "motorway":      {"high": 110, "low":  90},
            "motorway_link": {"high":  90, "low":  90},
            "trunk":         {"high":  90, "low":  60},
            "trunk_link":    {"high":  90, "low":  60},
            "primary":       {"high":  90, "low":  60},
            "primary_link":  {"high":  90, "low":  60},
            "secondary":     {"high":  90, "low":  60},
            "secondary_link":{"high":  90, "low":  60},
            "tertiary":      {"high":  90, "low":  60},
            "tertiary_link": {"high":  90, "low":  60},
            "unclassified":  {"high":  90, "low":  60},
            "residential":   {"high":  90, "low":  60},
            "service":       {"high":  40, "low":  20},
            "living_street": {"high":  20, "low":  20},
            "bus_guideway":  {"high":  80, "low":  60},
            "road":          {"high":  80, "low":  50}
        },

        "surface":
        {
            "paved":           1000,
            "asphalt":         1000,
            "cobblestone":     20,
            "cobblestone:flattened": 40,
            "sett":            40,
            "concrete":        1000,
            "concrete:lanes":  40,
            "concrete:plates": 100,
            "paving_stones":   40,
            "metal":           60,
```

```
141             "wood":             30,
142             "unpaved":          60,
143             "compacted":        70,
144             "dirt":             40,
145             "earth":            40,
146             "fine_gravel":      50,
147             "grass":            10,
148             "grass_paver":      20,
149             "gravel":           60,
150             "ground":           20,
151             "ice":              70,
152             "mud":              5,
153             "pebblestone":      50,
154             "salt":             70,
155             "sand":             70,
156             "snow":             50,
157             "woodchips":        5,
158             "metal_grid":       40
159         },
160
161         "barriers":
162         [
163             ["border_control",    120],
164             ["bump_gate",          30],
165             ["bus_trap",           30],
166             ["cattle_grid",        20],
167             ["entrance",           10],
168             ["gate",               30],
169             ["hampshire_gate",     60],
170             ["height_restrictor",   5],
171             ["jersey_barrier",     10],
172             ["lift_gate",          60],
173             ["sump_buster",        30],
174             ["swing_gate",         60],
175             ["toll_booth",         40]
176         ],
177
178         "highway":
179         [
180             ["bus_stop",            5],
181             ["crossing",            5],
182             ["give_way",           20],
183             ["mini_roundabout",    20],
184             ["stop",               30],
185             ["traffic_signals",    30]
186         ],
187
188         "railway":
189         [
190             ["level_crossing",     20]
191         ],
192
193         "public_transport":
194         [
195             ["stop_position",       5]
196         ],
197
```

```
      "traffic_calming":
      [
         ["yes",                10],
         ["bump",               10],
         ["hump",               10],
         ["table",              10],
         ["cushion",            10],
         ["rumble_strip",       10],
         ["chicane",            10],
         ["choker",             10],
         ["island",              5]
      ]

   }
}
```

D.2.4 mikhailovsk-original-temp.json

```
{
    "database":
    {
        "host":         "127.0.0.1",
        "port":          5432,
        "username":     "tester",
        "password":     "tester",
        "database":     "mikhailovsk-original-temp"
    },

    "topology":
    {
        "provider":      "postgis",

        "postgis":
        {

            "topo_name":       "lgu",
            "roads_prefix":    "highways",
            "schema_prefix":   "topo",
            "build": {
                "temp_topo_name": "epoch_ms",
                "srid":            900913,
                "tolerance":       1.0
            },
            "edge":
            {
                "table":       "edge_data",
                "id_col":      "edge_id",
                "source_col": "start_node",
                "target_col": "end_node",
                "geom_col":    "geom"
            },
            "vertex":
            {
                "table":      "node",
                "id_col":     "node_id",
                "geom_col":   "geom"
            }
```

```
        },

        "pgrouting":
        {
        },

        "jsontest":
        {
            "test_file": ""
        }

    },

    "vehicle":
    {
        "category":   "motorcar",
        "motorcar":
        {
           "height":       1.6,
           "length":       4.5,
           "width":        1.9,
           "weight":       2.0,
           "maxspeed":     200,
           "acceleration": 10,
           "deceleration": 7
        }
    },

  "access":
  {
     "allow":
     [
        "yes",
        "permissive",
        "designated"
     ]
  },

  "restrict":
  {
     "barriers":
     [
        "block",
        "bollard",
        "bus_trap",
        "chain",
        "cycle_barrier",
        "debris",
        "full-height_turnstile",
        "horse_stile",
        "jersey_barrier",
        "kent_carriage_gap",
        "kissing_gate",
        "log",
        "motorcycle_barrier",
        "rope",
        "sally_port",
```

```
97            "spikes",
98            "stile",
99            "sump_buster",
100           "swing_gate",
101           "turnstile",
102           "yes"
103       ]
104
105   },
106
107   "cost":
108   {
109       "default_speed":
110       {
111           "motorway":      {"high": 110, "low":  90},
112           "motorway_link": {"high":  90, "low":  90},
113           "trunk":         {"high":  90, "low":  60},
114           "trunk_link":    {"high":  90, "low":  60},
115           "primary":       {"high":  90, "low":  60},
116           "primary_link":  {"high":  90, "low":  60},
117           "secondary":     {"high":  90, "low":  60},
118           "secondary_link":{"high":  90, "low":  60},
119           "tertiary":      {"high":  90, "low":  60},
120           "tertiary_link": {"high":  90, "low":  60},
121           "unclassified":  {"high":  90, "low":  60},
122           "residential":   {"high":  90, "low":  60},
123           "service":       {"high":  40, "low":  20},
124           "living_street": {"high":  20, "low":  20},
125           "bus_guideway":  {"high":  80, "low":  60},
126           "road":          {"high":  80, "low":  50}
127       },
128
129       "surface":
130       {
131           "paved":            1000,
132           "asphalt":          1000,
133           "cobblestone":      20,
134           "cobblestone:flattened": 40,
135           "sett":             40,
136           "concrete":         1000,
137           "concrete:lanes":   40,
138           "concrete:plates":  100,
139           "paving_stones":    40,
140           "metal":            60,
141           "wood":             30,
142           "unpaved":          60,
143           "compacted":        70,
144           "dirt":             40,
145           "earth":            40,
146           "fine_gravel":      50,
147           "grass":            10,
148           "grass_paver":      20,
149           "gravel":           60,
150           "ground":           20,
151           "ice":              70,
152           "mud":              5,
153           "pebblestone":      50,
```

```
154             "salt":             70,
155             "sand":             70,
156             "snow":             50,
157             "woodchips":        5,
158             "metal_grid":       40
159         },
160
161         "barriers":
162         [
163             ["border_control",    120],
164             ["bump_gate",          30],
165             ["bus_trap",           30],
166             ["cattle_grid",        20],
167             ["entrance",           10],
168             ["gate",               30],
169             ["hampshire_gate",     60],
170             ["height_restrictor",   5],
171             ["jersey_barrier",     10],
172             ["lift_gate",          60],
173             ["sump_buster",        30],
174             ["swing_gate",         60],
175             ["toll_booth",         40]
176         ],
177
178         "highway":
179         [
180             ["bus_stop",            5],
181             ["crossing",            5],
182             ["give_way",           20],
183             ["mini_roundabout",    20],
184             ["stop",               30],
185             ["traffic_signals",    30]
186         ],
187
188         "railway":
189         [
190             ["level_crossing",     20]
191         ],
192
193         "public_transport":
194         [
195             ["stop_position",       5]
196         ],
197
198         "traffic_calming":
199         [
200             ["yes",                10],
201             ["bump",               10],
202             ["hump",               10],
203             ["table",              10],
204             ["cushion",            10],
205             ["rumble_strip",       10],
206             ["chicane",            10],
207             ["choker",             10],
208             ["island",              5]
209         ]
210     }
```

```
}
```

D.2.5 partille-original.json

```
{
    "database":
    {
        "host":         "127.0.0.1",
        "port":          5432,
        "username":     "tester",
        "password":     "tester",
        "database":     "partille-original"
    },

    "topology":
    {
        "provider":       "postgis",

        "postgis":
        {

            "topo_name":       "lgu",
            "roads_prefix":    "highways",
            "schema_prefix":   "topo",
            "build": {
                "temp_topo_name": "",
                "srid":            900913,
                "tolerance":       1.0
            },
            "edge":
            {
                "table":       "edge_data",
                "id_col":      "edge_id",
                "source_col": "start_node",
                "target_col": "end_node",
                "geom_col":    "geom"
            },
            "vertex":
            {
                "table":      "node",
                "id_col":     "node_id",
                "geom_col":   "geom"
            }
        },

        "pgrouting":
        {
        },

        "jsontest":
        {
            "test_file": ""
        }

    },

    "vehicle":
```

```
54      {
55          "category":   "motorcar",
56          "motorcar":
57          {
58             "height":       1.6,
59             "length":       4.5,
60             "width":        1.9,
61             "weight":       2.0,
62             "maxspeed":     200,
63             "acceleration": 10,
64             "deceleration": 7
65          }
66      },
67
68      "access":
69      {
70         "allow":
71         [
72            "yes",
73            "permissive",
74            "designated"
75         ]
76      },
77
78      "restrict":
79      {
80         "barriers":
81         [
82            "block",
83            "bollard",
84            "bus_trap",
85            "chain",
86            "cycle_barrier",
87            "debris",
88            "full-height_turnstile",
89            "horse_stile",
90            "jersey_barrier",
91            "kent_carriage_gap",
92            "kissing_gate",
93            "log",
94            "motorcycle_barrier",
95            "rope",
96            "sally_port",
97            "spikes",
98            "stile",
99            "sump_buster",
100           "swing_gate",
101           "turnstile",
102           "yes"
103        ]
104
105     },
106
107     "cost":
108     {
109        "default_speed":
110        {
```

```
            "motorway":      {"high": 110, "low":  90},
            "motorway_link": {"high":  90, "low":  90},
            "trunk":         {"high":  90, "low":  60},
            "trunk_link":    {"high":  90, "low":  60},
            "primary":       {"high":  90, "low":  60},
            "primary_link":  {"high":  90, "low":  60},
            "secondary":     {"high":  90, "low":  60},
            "secondary_link":{"high":  90, "low":  60},
            "tertiary":      {"high":  90, "low":  60},
            "tertiary_link": {"high":  90, "low":  60},
            "unclassified":  {"high":  90, "low":  60},
            "residential":   {"high":  90, "low":  60},
            "service":       {"high":  40, "low":  20},
            "living_street": {"high":  20, "low":  20},
            "bus_guideway":  {"high":  80, "low":  60},
            "road":          {"high":  80, "low":  50}
        },

        "surface":
        {
            "paved":            1000,
            "asphalt":          1000,
            "cobblestone":      20,
            "cobblestone:flattened": 40,
            "sett":             40,
            "concrete":         1000,
            "concrete:lanes":   40,
            "concrete:plates":  100,
            "paving_stones":    40,
            "metal":            60,
            "wood":             30,
            "unpaved":          60,
            "compacted":        70,
            "dirt":             40,
            "earth":            40,
            "fine_gravel":      50,
            "grass":            10,
            "grass_paver":      20,
            "gravel":           60,
            "ground":           20,
            "ice":              70,
            "mud":              5,
            "pebblestone":      50,
            "salt":             70,
            "sand":             70,
            "snow":             50,
            "woodchips":        5,
            "metal_grid":       40
        },

        "barriers":
        [
            ["border_control",    120],
            ["bump_gate",          30],
            ["bus_trap",           30],
            ["cattle_grid",        20],
            ["entrance",           10],
```

```
            ["gate",                30],
            ["hampshire_gate",      60],
            ["height_restrictor",    5],
            ["jersey_barrier",      10],
            ["lift_gate",           60],
            ["sump_buster",         30],
            ["swing_gate",          60],
            ["toll_booth",          40]
        ],

        "highway":
        [
            ["bus_stop",             5],
            ["crossing",             5],
            ["give_way",            20],
            ["mini_roundabout",     20],
            ["stop",                30],
            ["traffic_signals",     30]
        ],

        "railway":
        [
            ["level_crossing",      20]
        ],

        "public_transport":
        [
            ["stop_position",        5]
        ],

        "traffic_calming":
        [
            ["yes",                 10],
            ["bump",                10],
            ["hump",                10],
            ["table",               10],
            ["cushion",             10],
            ["rumble_strip",        10],
            ["chicane",             10],
            ["choker",              10],
            ["island",               5]
        ]
    }
}
```

D.2.6 partille-original-temp.json

```
{
    "database":
    {
        "host":         "127.0.0.1",
        "port":          5432,
        "username":     "tester",
        "password":     "tester",
        "database":     "partille-original-temp"
    },

```

```
    "topology":
    {
        "provider":      "postgis",

        "postgis":
        {

            "topo_name":      "lgu",
            "roads_prefix":   "highways",
            "schema_prefix":  "topo",
            "build": {
                "temp_topo_name": "epoch_ms",
                "srid":           900913,
                "tolerance":      1.0
            },
            "edge":
            {
                "table":      "edge_data",
                "id_col":     "edge_id",
                "source_col": "start_node",
                "target_col": "end_node",
                "geom_col":   "geom"
            },
            "vertex":
            {
                "table":     "node",
                "id_col":    "node_id",
                "geom_col":  "geom"
            }
        },

        "pgrouting":
        {
        },

        "jsontest":
        {
            "test_file": ""
        }

    },

    "vehicle":
    {
        "category":   "motorcar",
        "motorcar":
        {
           "height":       1.6,
           "length":       4.5,
           "width":        1.9,
           "weight":       2.0,
           "maxspeed":     200,
           "acceleration": 10,
           "deceleration": 7
        }
    },

```

```
68      "access":
69      {
70          "allow":
71          [
72              "yes",
73              "permissive",
74              "designated"
75          ]
76      },
77
78      "restrict":
79      {
80          "barriers":
81          [
82              "block",
83              "bollard",
84              "bus_trap",
85              "chain",
86              "cycle_barrier",
87              "debris",
88              "full-height_turnstile",
89              "horse_stile",
90              "jersey_barrier",
91              "kent_carriage_gap",
92              "kissing_gate",
93              "log",
94              "motorcycle_barrier",
95              "rope",
96              "sally_port",
97              "spikes",
98              "stile",
99              "sump_buster",
100             "swing_gate",
101             "turnstile",
102             "yes"
103         ]
104
105     },
106
107     "cost":
108     {
109         "default_speed":
110         {
111             "motorway":      {"high": 110, "low":  90},
112             "motorway_link": {"high":  90, "low":  90},
113             "trunk":         {"high":  90, "low":  60},
114             "trunk_link":    {"high":  90, "low":  60},
115             "primary":       {"high":  90, "low":  60},
116             "primary_link":  {"high":  90, "low":  60},
117             "secondary":     {"high":  90, "low":  60},
118             "secondary_link":{"high":  90, "low":  60},
119             "tertiary":      {"high":  90, "low":  60},
120             "tertiary_link": {"high":  90, "low":  60},
121             "unclassified":  {"high":  90, "low":  60},
122             "residential":   {"high":  90, "low":  60},
123             "service":       {"high":  40, "low":  20},
124             "living_street": {"high":  20, "low":  20},
```

```
125             "bus_guideway":  {"high":  80, "low":  60},
126             "road":          {"high":  80, "low":  50}
127         },
128
129         "surface":
130         {
131             "paved":            1000,
132             "asphalt":          1000,
133             "cobblestone":      20,
134             "cobblestone:flattened": 40,
135             "sett":             40,
136             "concrete":         1000,
137             "concrete:lanes":   40,
138             "concrete:plates":  100,
139             "paving_stones":    40,
140             "metal":            60,
141             "wood":             30,
142             "unpaved":          60,
143             "compacted":        70,
144             "dirt":             40,
145             "earth":            40,
146             "fine_gravel":      50,
147             "grass":            10,
148             "grass_paver":      20,
149             "gravel":           60,
150             "ground":           20,
151             "ice":              70,
152             "mud":              5,
153             "pebblestone":      50,
154             "salt":             70,
155             "sand":             70,
156             "snow":             50,
157             "woodchips":        5,
158             "metal_grid":       40
159         },
160
161         "barriers":
162         [
163             ["border_control",    120],
164             ["bump_gate",          30],
165             ["bus_trap",           30],
166             ["cattle_grid",        20],
167             ["entrance",           10],
168             ["gate",               30],
169             ["hampshire_gate",     60],
170             ["height_restrictor",   5],
171             ["jersey_barrier",     10],
172             ["lift_gate",          60],
173             ["sump_buster",        30],
174             ["swing_gate",         60],
175             ["toll_booth",         40]
176         ],
177
178         "highway":
179         [
180             ["bus_stop",            5],
181             ["crossing",            5],
```

```
            ["give_way",            20],
            ["mini_roundabout",     20],
            ["stop",                30],
            ["traffic_signals",     30]
        ],

        "railway":
        [
            ["level_crossing",      20]
        ],

        "public_transport":
        [
            ["stop_position",        5]
        ],

        "traffic_calming":
        [
            ["yes",                 10],
            ["bump",                10],
            ["hump",                10],
            ["table",               10],
            ["cushion",             10],
            ["rumble_strip",        10],
            ["chicane",             10],
            ["choker",              10],
            ["island",               5]
        ]
    }
}
```

D.2.7 mikhailovsk-barrier_block.json

```
{
    "database":
    {
        "host":         "127.0.0.1",
        "port":          5432,
        "username":     "tester",
        "password":     "tester",
        "database":     "mikhailovsk-barrier_block"
    },

    "topology":
    {
        "provider":       "postgis",

        "postgis":
        {

            "topo_name":       "lgu",
            "roads_prefix":    "highways",
            "schema_prefix":   "topo",
            "build": {
                "temp_topo_name": "",
                "srid":            900913,
                "tolerance":       1.0
```

```
            },
            "edge":
            {
                "table":      "edge_data",
                "id_col":     "edge_id",
                "source_col": "start_node",
                "target_col": "end_node",
                "geom_col":   "geom"
            },
            "vertex":
            {
                "table":     "node",
                "id_col":    "node_id",
                "geom_col":  "geom"
            }
        },

        "pgrouting":
        {
        },

        "jsontest":
        {
            "test_file": ""
        }

    },

    "vehicle":
    {
        "category":   "motorcar",
        "motorcar":
        {
           "height":       1.6,
           "length":       4.5,
           "width":        1.9,
           "weight":       2.0,
           "maxspeed":     200,
           "acceleration": 10,
           "deceleration": 7
        }
    },

    "access":
   {
      "allow":
      [
         "yes",
         "permissive",
         "designated"
      ]
   },

   "restrict":
   {
      "barriers":
      [
```

```
82            "block",
83            "bollard",
84            "bus_trap",
85            "chain",
86            "cycle_barrier",
87            "debris",
88            "full-height_turnstile",
89            "horse_stile",
90            "jersey_barrier",
91            "kent_carriage_gap",
92            "kissing_gate",
93            "log",
94            "motorcycle_barrier",
95            "rope",
96            "sally_port",
97            "spikes",
98            "stile",
99            "sump_buster",
100           "swing_gate",
101           "turnstile",
102           "yes"
103       ]
104
105   },
106
107   "cost":
108   {
109       "default_speed":
110       {
111           "motorway":      {"high": 110, "low":  90},
112           "motorway_link": {"high":  90, "low":  90},
113           "trunk":         {"high":  90, "low":  60},
114           "trunk_link":    {"high":  90, "low":  60},
115           "primary":       {"high":  90, "low":  60},
116           "primary_link":  {"high":  90, "low":  60},
117           "secondary":     {"high":  90, "low":  60},
118           "secondary_link":{"high":  90, "low":  60},
119           "tertiary":      {"high":  90, "low":  60},
120           "tertiary_link": {"high":  90, "low":  60},
121           "unclassified":  {"high":  90, "low":  60},
122           "residential":   {"high":  90, "low":  60},
123           "service":       {"high":  40, "low":  20},
124           "living_street": {"high":  20, "low":  20},
125           "bus_guideway":  {"high":  80, "low":  60},
126           "road":          {"high":  80, "low":  50}
127       },
128
129       "surface":
130       {
131           "paved":            1000,
132           "asphalt":          1000,
133           "cobblestone":      20,
134           "cobblestone:flattened": 40,
135           "sett":             40,
136           "concrete":         1000,
137           "concrete:lanes":   40,
138           "concrete:plates":  100,
```

```
            "paving_stones":    40,
            "metal":            60,
            "wood":             30,
            "unpaved":          60,
            "compacted":        70,
            "dirt":             40,
            "earth":            40,
            "fine_gravel":      50,
            "grass":            10,
            "grass_paver":      20,
            "gravel":           60,
            "ground":           20,
            "ice":              70,
            "mud":              5,
            "pebblestone":      50,
            "salt":             70,
            "sand":             70,
            "snow":             50,
            "woodchips":        5,
            "metal_grid":       40
        },

        "barriers":
        [
            ["border_control",    120],
            ["bump_gate",          30],
            ["bus_trap",           30],
            ["cattle_grid",        20],
            ["entrance",           10],
            ["gate",               30],
            ["hampshire_gate",     60],
            ["height_restrictor",   5],
            ["jersey_barrier",     10],
            ["lift_gate",          60],
            ["sump_buster",        30],
            ["swing_gate",         60],
            ["toll_booth",         40]
        ],

        "highway":
        [
            ["bus_stop",            5],
            ["crossing",            5],
            ["give_way",           20],
            ["mini_roundabout",    20],
            ["stop",               30],
            ["traffic_signals",    30]
        ],

        "railway":
        [
            ["level_crossing",     20]
        ],

        "public_transport":
        [
            ["stop_position",       5]
```

```
196         ],
197
198         "traffic_calming":
199         [
200             ["yes",                 10],
201             ["bump",                10],
202             ["hump",                10],
203             ["table",               10],
204             ["cushion",             10],
205             ["rumble_strip",        10],
206             ["chicane",             10],
207             ["choker",              10],
208             ["island",               5]
209         ]
210
211     }
212 }
```

D.2.8 partille-highway_traffic_signals.json

D.3 config

D.3.1 README.md

```
Configuration
=============

Configurations are set in the file `settings.json`. The different parts of the
 ↪  configuration is:

- Database
- Topology
- Vehicle
- Access
- Restrictions
- Costs

If one wishes to edit the setting, one can freely add and remove objects in
 ↪  _braces_ (`[` and `]`), for example if one wishes to edit which values for a
 ↪  tag inflicts a cost. But the strings in _curly braces_ (`{` and `}`) are keys
 ↪  that must exist, but the values for the keys can of course be edited.


Database
--------

Configuration for connecting to the database holding map data. The expected keys
 ↪  and values are:

- **"host"**:
    - *hostname* (e.g. `"localhost"`) or
    - *ip-address* (e.g. `"127.0.0.1"`).
- **"port"**:
    - *portnumber* (e.g. `5432`).
- **"username"**:
    - *username* (e.g. `"tester"`).
- **"password"**:
```

```
    - *password* (e.g. `"tester_pass"`).
- **"database"**:
    - *database name* (e.g. `"db_name"`).


Topology
--------

Configurations for building or reading topology from a database. Might have
↪  different meanings depending on which *MapProvider* are used. Topologies can
↪  be pre-built, or they can be generated each time, depending on the settings
↪  in `"build_topo"`. It is also possible to define a simple json test file, for
↪  testing simple topologies.

- **"provider"**:
    - *name* of *MapProvider*
        - `"postgis"` when using `postgis_topology` for building topologies.
        - `"pgrouting"` when using `pgrouting` for building topologies.
        - `"jsontest"` for simple json test topology.

- **"postgis"**:
    - **"topo_name"**:
        - *basename* for pre-built topologies (e.g. `"test"`), combined with
        ↪  `roads_prefix` and `topo_prefix` for actual names such as
        ↪  `"highways_test"` and `"topo_test"`.

    - **"roads_prefix"**:
        - *prefix* to add to `topo_name` (e.g. `"highways"`) for table of roads
        ↪  network, see above.

    - **"schema_prefix"**:
        - *prefix* to add to `topo_name` (e.g. `"topo"`) for schema with topology
        ↪  data when using `postgis_topology`, see above.

    - **"build"**:
        - **"temp_topo_name**:
            - `""` (*empty*) if not building temporary topologies.
            - `"epoch_ms"` for adding a string with the count of milliseconds
            ↪  since "Epoch" as the `topo_name`.

        - **"srid"**:
            - *number* identifying which projection to use.
                - `900913` for geometric metrical projection, unit meters.
                - `4326` for geographic spherical projection, unit degrees.

        - **"tolerance"**:
            - *snapping* of nodes in unit of projection when building topology,
            ↪  e.g. 1.0 for srid 900913, or 0.001 for srid 4326.

    - **"edge"**:

        - **"table"**:
            - *name* of the table with topology edges (e.g. `"edge_data"`), with
            ↪  column for id, source, target and geometry.

        - **"id_col"**:
```

```
            - *name* of the column in edge table with id of edges (e.g.
            ↪  `"edge_id"`).

        - **"source_col"**:
            - *name* of the column in edge table with vertex id of **source** of
            ↪  edge (e.g. `"start_node"`).

        - **"target_col"**:
            - *name* of the column in edge table with vertex id of **target** of
            ↪  edge (e.g. `"end_node"`).

        - **"geom_col"**:
            - *name* of the column in edge table with geometry of edge (e.g.
            ↪  `"geom"`).

    - **"vertex"**:
        - **"table"**:
            - *name* of the table with topology vertices (e.g. `"node"`), with
            ↪  column for id and geometry.

        - **"id_col"**:
            - *name* of the column in vertex table with id of vertices (e.g.
            ↪  `"node_id"`).

        - **"geom_col"**:
            - *name* of the column in vertex table with geometry of vertex (e.g.
            ↪  `"geom"`).

- **"pgrouting"**:
    - TODO.

- **"jsontest"**:
    - **"test_file"**:
        - `""` (*empty*) if not using *json-test provider*.
        - *filename* to a json test-file (e.g. `"test.json"`) looking like:

    ```json
    {
        "vertices": [
            [1,2,0],
            [2,2,1]
        ],
        "edges": [
            [1,1,2,0]
        ]
    }
    ```

        where each row in `vertices` are `[id,x,y]` and each row in `edges` are
        ↪ `[id, source vertex id, target vertex id, direction]`. Values for
        ↪ `direction` is `0 = BOTH`, `1 = FROM_TO`, `2 = TO_FROM`.

Vehicle

```

```
Configuration about the vehicle to route through the topology. Information might
↪  be needed to take restrictions in account.

- **"category"**:
    - *name* of OSM category of the vehicle. [OSM
    ↪  Access](http://wiki.openstreetmap.org/wiki/Key:access). (E.g.
    ↪  "motorcar"). Definition of the category must state dimensions as below.
- **_"category_name"_**:
    - *height* of vehicle in meters.
    - *length* of vehicle in meters.
    - *width* of vehicle in meters.
    - *weight* of vehicle in tons.
    - *maxspeed* of vehicle in km/h.
    - *acceleration* is the time it takes from 0 to 100 km/h.
    - *deceleration* is the time it takes from 100 to 0 km/h.


Access
------

- **"allow"**:
    List of which values for tag `access` that permits access. Those values for
    ↪  `access` that are not listed here are considered to restrict access.


Restrictions
------------

- **"barriers"**:
    - List of which values for `barriers` that restricts access. Those values not
    ↪  listed are assumed to allow access.


Cost
----

Configuration relating to costs when routing through the graph.

- **"default_speed"**:
    - each road category has default speeds when none is specified. [OSM default
    ↪  speeds](http://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Maxspeed).
    ↪  Most roads have two speeds, `high` and `low`, which differentiate the
    ↪  speeds inside and outside of a town. `living_street` is always inside so
    ↪  only the low is important. `motorway` is really the `high` number, and
    ↪  the `low` number is the speed on the links (ramps). It is not trivial to
    ↪  find out if a road is inside or outside of that area, so for this
    ↪  application which is meant to be used for routing in urban areas (?), the
    ↪  `low` number is assumed for all cost calculations.
- **"surface"**:
    - each surface type is associated with a max speed in km/h over which one
    ↪  should not drive.
- **"barriers"**:
    - this is a list of which barriers causes a slow-down, and the number of
    ↪  seconds it is  probable it takes to pass.
- **"highway"**:
    - a list of values for the `highway` tag that can mean a time cost in seconds,
    ↪  such as zebra crossings, bus stops, stop or give way sign, and more.
```

```
- **"railway"**:
    - a list of values for the `railway` tag that can mean a time cost in seconds,
     ↪  such as a crossing between railway and highway (`level_crossing`).
- **"public_transport"**:
    - a list of values for the `public_transport` tag that can mean a time cost in
     ↪  seconds, such as a `stop_position` if one thinks it is suitable to slow
     ↪  down when passing a bus stop.
- **"traffic_calming"**:
    - a list of traffic calming objects and their time costs in seconds.
```

D.3.2 Configuration.h

```
/**  A container for configurations.
 *
 * #include "Configuration.h"
 *
 * @author  Jonas Bergman
 */

#ifndef CONFIG_CONFIGURATION_H_
#define CONFIG_CONFIGURATION_H_

// SYSTEM INCLUDES
//

// PROJECT INCLUDES
//

// LOCAL INCLUDES
//
#include "DatabaseConfig.h"
#include "VehicleConfig.h"
#include "CostConfig.h"
#include "../osm/OsmAccess.h"
#include "../osm/OsmBarrier.h"

// FORWARD REFERENCES
//

/**
 * This class holds configurations for different parts of the utility.
 * The ConfigurationReader is friend so it can populate the different
 * configurations.
 */
class Configuration
{
    friend class ConfigurationReader;
public:
// LIFECYCLE

    /** Default constructor.
     */
    Configuration() = default;


    /** Copy constructor.
```

```
45          *
46          * @param from The value to copy to this object.
47          */
48         Configuration(const Configuration& from) = delete;
49
50
51         /** Destructor.
52          */
53         ~Configuration(void) = default;
54
55
56     // OPERATORS
57     // OPERATIONS
58     // ACCESS
59         /** Get the database related parts of the configuration.
60          * @return  Reference to a DatabaseConfig.
61          */
62         const DatabaseConfig& getDatabaseConfig() const;
63
64         /** Get the topology related parts of the configuration.
65          * @return  Reference to a TopologyConfig.
66          */
67         const TopologyConfig& getTopologyConfig() const;
68
69         /** Get the vehicle related parts of the configuration.
70          * @return  Reference to a VehicleConfig.
71          */
72         const VehicleConfig&  getVehicleConfig() const;
73
74         /** Get the rules for which values of the `access`-tag allows access
75          * and hence which values restricts access to an Edge.
76          * @return  Reference to an AccessRule
77          */
78         const OsmAccess::AccessRule&
79                               getAccessRule() const;
80
81         /** Get the rules for which values of the `barrier`-tag restricts access
82          * @return  Reference to an RestrictionsRule
83          */
84         const OsmBarrier::RestrictionsRule&
85                               getBarrierRestrictionsRule() const;
86
87         /** Get the rules for which values of the `barrier`-tag costs to pass
88          * @return  Reference to an CostsRule
89          */
90         const OsmBarrier::CostsRule&
91                               getBarrierCostsRule() const;
92
93         /** Get the cost related parts of the configuration.
94          * @return  Reference to a CostConfig.
95          */
96         const CostConfig&     getCostConfig() const;
97
98     // INQUIRY
99
100    protected:
101    private:
```

```
102 // ATTRIBUTES
103     DatabaseConfig          mDbConfig;
104     TopologyConfig          mTopoConfig;
105     VehicleConfig           mVehicleConfig;
106     CostConfig              mCostConfig;
107     OsmAccess::AccessRule   mAccessRule;
108     OsmBarrier::CostsRule   mBarrierCostsRule;
109     OsmBarrier::RestrictionsRule mBarrierRestrictionsRule;
110 };
111 
112 // INLINE METHODS
113 //
114 
115 // EXTERNAL REFERENCES
116 //
117 
118 #endif /* CONFIG_CONFIGURATION_H_ */
```

D.3.3 Configuration.cc

```
1  /*
2   * Configuration.cc
3   * @author  Jonas Bergman
4   */
5  
6  #include "Configuration.h"  // class implemented
7  
8  ////////////////////////////// PUBLIC //////////////////////////////////////
9  
10 //============================= LIFECYCLE ====================================
11 //============================= OPERATORS ====================================
12 //============================= OPERATIONS ===================================
13 //============================= ACESS      ===================================
14 const DatabaseConfig&
15 Configuration::getDatabaseConfig() const
16 {
17     return mDbConfig;
18 }
19 
20 const TopologyConfig&
21 Configuration::getTopologyConfig() const
22 {
23     return mTopoConfig;
24 }
25 
26 const VehicleConfig&
27 Configuration::getVehicleConfig() const
28 {
29     return mVehicleConfig;
30 }
31 
32 const OsmAccess::AccessRule&
33 Configuration::getAccessRule() const
34 {
35     return mAccessRule;
36 }
37 
```

```
38 const OsmBarrier::RestrictionsRule&
39 Configuration::getBarrierRestrictionsRule() const
40 {
41     return mBarrierRestrictionsRule;
42 }
43
44 const OsmBarrier::CostsRule&
45 Configuration::getBarrierCostsRule() const
46 {
47     return mBarrierCostsRule;
48 }
49
50 const CostConfig&
51 Configuration::getCostConfig() const
52 {
53     return mCostConfig;
54 }
55
56 //============================== INQUIRY    ====================================
57 ////////////////////////////////// PROTECTED  ////////////////////////////////////
58
59 ////////////////////////////////// PRIVATE    ////////////////////////////////////
```

D.3.4 ConfigurationException.h

```
1  /**  Exception thrown by the Configuration package.
2   *
3   * #include "ConfigurationException.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef CONFIG_CONFIGURATIONEXCEPTION_H_
9  #define CONFIG_CONFIGURATIONEXCEPTION_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <exception>
14 #include <string>
15
16 // PROJECT INCLUDES
17 //
18
19 // LOCAL INCLUDES
20 //
21
22 // FORWARD REFERENCES
23 //
24
25 /**
26  * Exception to throw from the 'config' package.
27  * More information of the type of exception is given in the 'what()' message.
28  */
29 class ConfigurationException : public std::exception
30 {
31 public:
32 // LIFECYCLE
```

```
33      /** Default constructor.
34       */
35      ConfigurationException() = delete;
36
37      /** Constructor taking a message to display.
38       *
39       * @param        message                The message to prepend when 'what()' is called.
40       */
41      ConfigurationException(const std::string& rMessage) noexcept
42          : std::exception(), mMessage(rMessage)
43      {}
44
45  // OPERATORS
46  // OPERATIONS
47  // ACCESS
48  // INQUIRY
49      const char* what() const noexcept
50      { return (mMessage + " " + std::exception::what()).c_str(); }
51
52  protected:
53  private:
54  // ATTRIBUTES
55      std::string                mMessage;
56  };
57
58  // INLINE METHODS
59  //
60
61  // EXTERNAL REFERENCES
62  //
63
64  #endif /* CONFIG_CONFIGURATIONEXCEPTION_H_ */
```

D.3.5 ConfigurationReader.h

```
1   /**  Read configurations from a json file.
2    *
3    * #include "ConfigurationReader.h"
4    *
5    * @author  Jonas Bergman
6    */
7   #ifndef CONFIG_CONFIGURATIONREADER_H_
8   #define CONFIG_CONFIGURATIONREADER_H_
9
10  // SYSTEM INCLUDES
11  //
12  #include <string>
13
14  // PROJECT INCLUDES
15  //
16  #include <boost/property_tree/ptree.hpp>
17  #include <boost/property_tree/json_parser.hpp>
18
19  // LOCAL INCLUDES
20  //
21  #include "Configuration.h"
22  #include "ConfigurationException.h"
```

```
23 #include "DatabaseConfig.h"
24 #include "TopologyConfig.h"
25 #include "VehicleConfig.h"
26 #include "../osm/OsmVehicle.h"
27
28 // FORWARD REFERENCES
29 //
30
31 /**
32  * A class to handle the reading of data from a json configuration file.
33  */
34 class ConfigurationReader
35 {
36 public:
37 // LIFECYCLE
38     /** Default constructor.
39      */
40     ConfigurationReader() = delete;
41
42     /** Constructor.
43      * Always initialize a Configuration reader with the configuration file.
44      *
45      * @param    rFilename    The filename for the configuration json file
46      * @throw    ConfigurationException        If invalid file
47      */
48     ConfigurationReader(const std::string& rFilename);
49
50
51 // OPERATORS
52 // OPERATIONS
53
54     /** Get the configurations from the file.
55      * @param   Reference to a Configuration to populate.
56      * @throws  ConfigurationException
57      */
58     void    fillConfiguration(Configuration& rConfig) const;
59
60 // ACCESS
61 // INQUIRY
62
63 protected:
64
65 private:
66 // ATTRIBUTES
67     std::string                    mFilename;
68     boost::property_tree::ptree    mPropertyTree;
69
70 // HELPERS
71     /** Read the database part of the configuration and populate config struct.
72      * @param    The Database configuration
73      * @throw    ConfigurationException    If missing configuration.
74      */
75     void    fillDatabaseConfiguration(DatabaseConfig& rDatabaseConfig) const;
76
77     /** Read the topology part of the configuration and populate config struct.
78      * @param    The Topology configuration
79      * @throw    ConfigurationException    If missing configuration.
```

```
80          */
81         void    fillTopologyConfiguration(TopologyConfig& rTopologyConfig) const;
82
83         /** Read the vehicle part of the configuration and populate config struct.
84          * @param   The Vehicle configuration
85          * @throw   ConfigurationException  If missing configuration.
86          */
87         void    fillVehicleConfiguration(VehicleConfig& rVehicleConfig) const;
88
89         /** Read the Access part of the configuration and build the rule for
90          * which tags allows access (and hence which tags restricts access).
91          * @param   rAccessRule     The rule to fill out.
92          * @throw   ConfigurationException  If missing configuration.
93          */
94         void    fillAccessRule(OsmAccess::AccessRule& rAccessRule) const;
95
96         /** Read the Barrier part of the configuration and build the rule for
97          * which barriers restricts access.
98          * @param   rRestrictRule     The rule to fill out.
99          * @throw   ConfigurationException  If missing configuration.
100         */
101        void    fillBarrierRestrictRule(OsmBarrier::RestrictionsRule& rRestrictRule) const;
102
103        /** Read the Barrier part of the configuration and build the rule for
104         * which barriers imposes a cost.
105         * @param   rCostRule     The rule to fill out.
106         * @throw   ConfigurationException  If missing configuration.
107         */
108        void    fillBarrierCostsRule(OsmBarrier::CostsRule& rCostsRule) const;
109
110        /** Read the Cost part of the configuration and populate config struct.
111         * @param   The Cost configuration
112         * @throw   ConfigurationException  If missing configuration.
113         */
114        void    fillCostConfiguration(CostConfig& rCostConfig) const;
115
116        /** Helper to `fillCostConfig()`. Fill in the Default Speed part.
117         * @param   The Cost configuration.
118         */
119        void    fillDefaultSpeedCost(CostConfig& rCostConfig) const;
120
121        /** Helper to `fillCostConfig()`. Fill in the Surface Max Speed part.
122         * @param   The Cost configuration.
123         */
124        void    fillSurfaceMaxSpeedCost(CostConfig& rCostConfig) const;
125
126        /** Helper to `fillCostConfig()`. Fill in the cost for other edge costs.
127         * @param   The Cost configuration.
128         */
129        void    fillOtherEdgeCosts(CostConfig& rCostConfig) const;
130    };
131
132    // INLINE METHODS
133    //
134
135    // EXTERNAL REFERENCES
136    //
```

```
137
138 #endif /* CONFIG_CONFIGURATIONREADER_H_ */
```

D.3.6 ConfigurationReader.cc

```
1  /*
2   * ConfigurationReader.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "ConfigurationReader.h"  // class implemented
8
9  ////////////////////////////// PUBLIC /////////////////////////////////////
10
11 //============================= LIFECYCLE ====================================
12
13 ConfigurationReader::ConfigurationReader(const std::string& rFilename)
14     : mFilename(rFilename)
15 {
16     try
17     {
18         boost::property_tree::read_json(mFilename, mPropertyTree);
19     }
20     catch (boost::property_tree::json_parser_error& e)
21     {
22         throw ConfigurationException("Could not read file " + mFilename);
23     }
24 }
25
26 //============================= OPERATORS ====================================
27
28 //============================= OPERATIONS ===================================
29 void ConfigurationReader::fillConfiguration(Configuration& rConfig) const
30 {
31     fillDatabaseConfiguration(rConfig.mDbConfig);
32     fillTopologyConfiguration(rConfig.mTopoConfig);
33     fillVehicleConfiguration(rConfig.mVehicleConfig);
34     fillAccessRule(rConfig.mAccessRule);
35     fillBarrierRestrictRule(rConfig.mBarrierRestrictionsRule);
36     fillBarrierCostsRule(rConfig.mBarrierCostsRule);
37     fillCostConfiguration(rConfig.mCostConfig);
38 }
39
40 //============================= ACESS      ===================================
41 //============================= INQUIRY    ===================================
42 ////////////////////////////// PROTECTED  /////////////////////////////////
43
44 ////////////////////////////// PRIVATE    /////////////////////////////////
45 void ConfigurationReader::fillDatabaseConfiguration(
46     DatabaseConfig& rDbConfig) const
47 {
48     std::string prefix("database.");
49
50     try
51     {
52         rDbConfig.hostname = mPropertyTree.get<std::string>(prefix + "host");
```

```
53          rDbConfig.port = mPropertyTree.get<int>(prefix + "port");
54          rDbConfig.username = mPropertyTree.get<std::string>(
55              prefix + "username");
56          rDbConfig.password = mPropertyTree.get<std::string>(
57              prefix + "password");
58          rDbConfig.database = mPropertyTree.get<std::string>(
59              prefix + "database");
60      }
61      catch (boost::property_tree::ptree_error& e)
62      {
63          throw ConfigurationException(
64              std::string("Could not read config ") + e.what());
65      }
66  }
67
68  void ConfigurationReader::fillTopologyConfiguration(
69      TopologyConfig& rTopoConfig) const
70  {
71      std::string prefix("topology.");
72
73      try
74      {
75          rTopoConfig.providerName = mPropertyTree.get<std::string>(
76              prefix + "provider");
77
78          if(rTopoConfig.providerName == TopologyConfig::PROVIDER_JSONTEST)
79          {
80              rTopoConfig.testFile = mPropertyTree.get<std::string>(
81                  prefix + "jsontest.test_file");
82          }
83          else if(rTopoConfig.providerName == TopologyConfig::PROVIDER_POSTGIS
84              || rTopoConfig.providerName == TopologyConfig::PROVIDER_PGROUTING)
85          {
86              prefix += rTopoConfig.providerName + ".";
87
88              rTopoConfig.topoName = mPropertyTree.get<std::string>(
89                  prefix + "topo_name");
90
91              rTopoConfig.roadsPrefix = mPropertyTree.get<std::string>(
92                  prefix + "roads_prefix");
93              rTopoConfig.topologySchemaPrefix = mPropertyTree.get<std::string>(
94                  prefix + "schema_prefix");
95
96              rTopoConfig.tempTopoName = mPropertyTree.get<std::string>(
97                  prefix + "build.temp_topo_name");
98              rTopoConfig.srid = mPropertyTree.get<int>(prefix + "build.srid");
99              rTopoConfig.tolerance = mPropertyTree.get<double>(
100                 prefix + "build.tolerance");
101
102             rTopoConfig.edgeTableName = mPropertyTree.get<std::string>(
103                 prefix + "edge.table");
104             rTopoConfig.edgeIdColumnName = mPropertyTree.get<std::string>(
105                 prefix + "edge.id_col");
106             rTopoConfig.sourceColumnName = mPropertyTree.get<std::string>(
107                 prefix + "edge.source_col");
108             rTopoConfig.targetColumnName = mPropertyTree.get<std::string>(
109                 prefix + "edge.target_col");
```

```
110             rTopoConfig.edgeGeomColumnName = mPropertyTree.get<std::string>(
111                 prefix + "edge.geom_col");
112
113             rTopoConfig.vertexTableName = mPropertyTree.get<std::string>(
114                 prefix + "vertex.table");
115             rTopoConfig.vertexIdColumnName = mPropertyTree.get<std::string>(
116                 prefix + "vertex.id_col");
117             rTopoConfig.vertexGeomColumnName = mPropertyTree.get<std::string>(
118                 prefix + "vertex.geom_col");
119         }
120     }
121     catch (boost::property_tree::ptree_error& e)
122     {
123         throw ConfigurationException(
124             std::string("Could not read config ") + e.what());
125     }
126 }
127
128 void ConfigurationReader::fillVehicleConfiguration(
129     VehicleConfig& rVehicleConfig) const
130 {
131     std::string prefix("vehicle.");
132
133     try
134     {
135         std::string categoryString = mPropertyTree.get<std::string>(
136             prefix + "category");
137         rVehicleConfig.category = OsmVehicle::parseString(categoryString);
138         prefix += categoryString + ".";
139         rVehicleConfig.height = mPropertyTree.get<double>(prefix + "height");
140         rVehicleConfig.length = mPropertyTree.get<double>(prefix + "length");
141         rVehicleConfig.weight = mPropertyTree.get<double>(prefix + "weight");
142         rVehicleConfig.width = mPropertyTree.get<double>(prefix + "width");
143         rVehicleConfig.maxspeed = mPropertyTree.get<unsigned>(
144             prefix + "maxspeed");
145         rVehicleConfig.acceleration = mPropertyTree.get<unsigned>(
146             prefix + "acceleration");
147         rVehicleConfig.deceleration = mPropertyTree.get<unsigned>(
148             prefix + "deceleration");
149     }
150     catch (ConfigurationException& e)
151     {
152         throw e;
153     }
154     catch (boost::property_tree::ptree_error& e)
155     {
156         throw ConfigurationException(
157             std::string("Could not read config ") + e.what());
158     }
159 }
160
161 void ConfigurationReader::fillAccessRule(
162     OsmAccess::AccessRule& rAccessRule) const
163 {
164     std::string prefix("access.allow");
165
166     try
```

```
167     {
168         std::vector<OsmAccess::AccessType> allow_tags;
169         for (auto& item : mPropertyTree.get_child(prefix))
170         {
171             std::string tag_string = item.second.get_value<std::string>();
172             allow_tags.push_back(OsmAccess::parseString(tag_string));
173         }
174         rAccessRule.allowAccessToTypes = allow_tags;
175     }
176     catch (ConfigurationException& e)
177     {
178         throw e;
179     }
180     catch (OsmException& ose)
181     {
182         throw ConfigurationException(
183             std::string("Could not read config")
184                 + ", error parsing access tag: " + ose.what());
185     }
186     catch (boost::property_tree::ptree_error& e)
187     {
188         throw ConfigurationException(
189             std::string("Could not read config ") + e.what());
190     }
191 }
192
193 void ConfigurationReader::fillBarrierRestrictRule(
194     OsmBarrier::RestrictionsRule& rRestrictRule) const
195 {
196     std::string prefix("restrict.barriers");
197
198     try
199     {
200         std::vector<OsmBarrier::BarrierType> restrict_barriers;
201         for (auto& item : mPropertyTree.get_child(prefix))
202         {
203             std::string restrict_string =
204                 item.second.get_value<std::string>();
205             restrict_barriers.push_back(
206                 OsmBarrier::parseString(restrict_string));
207         }
208         rRestrictRule.restrictionTypes = restrict_barriers;
209     }
210     catch (ConfigurationException& e)
211     {
212         throw e;
213     }
214     catch (OsmException& ose)
215     {
216         throw ConfigurationException(
217             std::string("Could not read config")
218                 + ", error parsing barrier restrictions: " + ose.what());
219     }
220     catch (boost::property_tree::ptree_error& e)
221     {
222         throw ConfigurationException(
223             std::string("Could not read config ") + e.what());
```

```
224     }
225 }
226
227 void ConfigurationReader::fillBarrierCostsRule(
228     OsmBarrier::CostsRule& rCostsRule) const
229 {
230     std::string prefix("cost.barriers");
231
232     try
233     {
234         for (auto& row : mPropertyTree.get_child(prefix))
235         {
236             int i = 0;
237             std::string type_string;
238             unsigned cost;
239             for (auto& item : row.second)
240             {
241                 if(i == 0)
242                 {
243                     type_string = item.second.get_value<std::string>();
244                 }
245                 else
246                 {
247                     cost = item.second.get_value<unsigned>();
248                 }
249                 ++i;
250             }
251             OsmBarrier::BarrierType barrier_type = OsmBarrier::parseString(
252                 type_string);
253             rCostsRule.addCost(barrier_type, cost);
254         }
255     }
256     catch (ConfigurationException& e)
257     {
258         throw e;
259     }
260     catch (OsmException& ose)
261     {
262         throw ConfigurationException(
263             std::string("Could not read config")
264                 + ", error parsing barrier costs: " + ose.what());
265     }
266     catch (boost::property_tree::ptree_error& e)
267     {
268         throw ConfigurationException(
269             std::string("Could not read config ") + e.what());
270     }
271 }
272
273 void ConfigurationReader::fillCostConfiguration(CostConfig& rCostConfig) const
274 {
275     try
276     {
277         fillDefaultSpeedCost(rCostConfig);
278         fillSurfaceMaxSpeedCost(rCostConfig);
279         fillOtherEdgeCosts(rCostConfig);
280     }
```

```
281     catch (ConfigurationException& e)
282     {
283         throw e;
284     }
285     catch (boost::property_tree::ptree_error& e)
286     {
287         throw ConfigurationException(
288             std::string("Could not read config ") + e.what());
289     }
290 }
291 
292 void ConfigurationReader::fillDefaultSpeedCost(CostConfig& rCostConfig) const
293 {
294     std::string prefix("cost.default_speed.");
295 
296     CostConfig::DefaultSpeed::HighLowSpeed hilo;
297     std::string type_string;
298     OsmHighway::HighwayType type;
299 
300     for (size_t i = 0; i < OsmHighway::NR_HIGHWAY_TYPES; ++i)
301     {
302         type_string = OsmHighway::typeStrings().at(i);
303         hilo.high = mPropertyTree.get<int>(prefix + type_string + ".high");
304         hilo.low = mPropertyTree.get<int>(prefix + type_string + ".low");
305         type = static_cast<OsmHighway::HighwayType>(i);
306         rCostConfig.defaultSpeed.addDefaultSpeed(type, hilo);
307     }
308 }
309 
310 void ConfigurationReader::fillSurfaceMaxSpeedCost(
311     CostConfig& rCostConfig) const
312 {
313     std::string prefix("cost.surface.");
314 
315     Speed speed;
316     std::string type_string;
317     OsmHighway::SurfaceType type;
318 
319     for (size_t i = 0; i < OsmHighway::NR_SURFACE_TYPES; ++i)
320     {
321         type_string = OsmHighway::surfaceTypeStrings().at(i);
322         speed = mPropertyTree.get<int>(prefix + type_string);
323         type = static_cast<OsmHighway::SurfaceType>(i);
324         rCostConfig.surfaceMaxSpeed.addSurfaceMaxSpeed(type, speed);
325     }
326 }
327 
328 void ConfigurationReader::fillOtherEdgeCosts(CostConfig& rCostConfig) const
329 {
330     std::string section("cost.");
331     std::vector<std::string> subsections { "highway", "railway",
332         "public_transport", "traffic_calming" };
333 
334     try
335     {
336         for (const auto& sub : subsections)
337         {
```

```
338             std::string prefix(section + sub + ".");
339
340             for (auto& row : mPropertyTree.get_child(prefix))
341             {
342                 int i = 0;
343                 std::string key;
344                 Cost cost;
345                 for (auto& item : row.second)
346                 {
347                     if(i == 0)
348                     {
349                         key = item.second.get_value<std::string>();
350                     }
351                     else
352                     {
353                         cost = item.second.get_value<Cost>();
354                     }
355                     ++i;
356                 }
357                 rCostConfig.otherEdgeCosts.addOtherCost(sub + "=" + key,
358                     cost);
359             }
360         }
361     }
362     catch (ConfigurationException& e)
363     {
364         throw e;
365     }
366     catch (OsmException& ose)
367     {
368         throw ConfigurationException(
369             std::string("Could not read config")
370                 + ", error parsing other costs: " + ose.what());
371     }
372     catch (boost::property_tree::ptree_error& e)
373     {
374         throw ConfigurationException(
375             std::string("Could not read config ") + e.what());
376     }
377 }
```

D.3.7 CostConfig.h

```
1  /**  Data structure for configuration of costs.
2   *
3   * #include "CostConfig.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef CONFIG_COSTCONFIG_H_
9  #define CONFIG_COSTCONFIG_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <map>
14 #include <string>
```

```
15
16  // PROJECT INCLUDES
17  //
18
19  // LOCAL INCLUDES
20  //
21  #include "../osm/OsmHighway.h"
22  #include "../graph/Cost.h"
23  #include "../graph/Speed.h"
24
25  // FORWARD REFERENCES
26  //
27
28  /**
29   * Information about names in the database for cost data.
30   */
31  struct CostConfig
32  {
33  // TYPES
34
35      /** Keep track of default speeds for different categories of roads.
36       * The values are a high and a low value, depending of if we are inside or
37       * outside of an urban area.
38       */
39      struct DefaultSpeed
40      {
41          enum HIGH_LOW
42          {
43              HIGH,
44              LOW
45          };
46
47          struct HighLowSpeed
48          {
49              Speed high {0};
50              Speed low {0};
51          };
52
53          std::map<OsmHighway::HighwayType, HighLowSpeed> defaultSpeed;
54
55          /** Add a speed far a specific road category (highway type).
56           * @param   type    The highway type
57           * @param   speed   The high and low speed limits.
58           */
59          void            addDefaultSpeed(
60                              OsmHighway::HighwayType type,
61                              HighLowSpeed speed)
62          {
63              defaultSpeed.erase(type);
64              defaultSpeed.insert({type, speed});
65          }
66
67          /**
68           * @param   type    The highway type
69           * @return  The high/low speed for this type of highway.
70           */
71          HighLowSpeed    getDefaultSpeed(OsmHighway::HighwayType type) const
```

```
72          {
73              const auto& it = defaultSpeed.find(type);
74              if(it != defaultSpeed.end())
75              {
76                  return it->second;
77              }
78              return HighLowSpeed();
79          }
80
81          /** Get a high or low speed limit for a highway type.
82           * @param   type        The Type of highway.
83           * @param   highOrLow   Either HIGH or LOW speed
84           * @return  Either the high or low speed for a highway type.
85           */
86          Speed           getDefaultSpeed(
87                              OsmHighway::HighwayType type,
88                              HIGH_LOW                highOrLow) const
89          {
90              HighLowSpeed hl = getDefaultSpeed(type);
91              if(highOrLow == HIGH)
92              {
93                  return hl.high;
94              }
95              else
96              {
97                  return hl.low;
98              }
99          }
100     };
101
102     /** Keep track of max speed that are suitable for different kind of
103      * surfaces.
104      */
105     struct SurfaceMaxSpeed
106     {
107         std::map<OsmHighway::SurfaceType, Speed> surfaceSpeed;
108
109         /** Add a surface type and the max speed.
110          * @param   type    The type of surface.
111          * @param   speed   The max suitable speed for the surface type.
112          */
113         void    addSurfaceMaxSpeed(OsmHighway::SurfaceType type, Speed speed)
114         {
115             surfaceSpeed.erase(type);
116             surfaceSpeed.insert({type, speed});
117         }
118
119         /**
120          * @return  The suitable max speed for a surface type.
121          */
122         Speed   getSurfaceMaxSpeed(OsmHighway::SurfaceType type) const
123         {
124             const auto& it = surfaceSpeed.find(type);
125             if(it != surfaceSpeed.end())
126             {
127                 return it->second;
128             }
```

```
129             return 0;
130         }
131     };
132 
133     /** Other edge costs are kept track of simply by strings as keys and
134      * Costs as values. The costs are "penalties" added to the travel time.
135      * The string that make up the keys are simply constructed as "tag=value",
136      * e.g. "highway=give_way".
137      */
138     struct OtherEdgeCosts
139     {
140         std::map<std::string, Cost> otherCostValues;
141 
142         /** Add a 'penalty' for another kind of EdgeCost.
143          * @param   key     String of "tag=value" that makes up the cost.
144          * @param   cost    The cost for this kind of hindrance.
145          */
146         void    addOtherCost(std::string key, Cost cost)
147         {
148             otherCostValues.erase(key);
149             otherCostValues.insert({key, cost});
150         }
151 
152         /** Get other costs associated with the key.
153          * @param   key
154          * @return  The cost for this key.
155          */
156         Cost    getOtherCost(std::string key) const
157         {
158             const auto& it = otherCostValues.find(key);
159             if(it !=otherCostValues.end())
160             {
161                 return it->second;
162             }
163             return 0;
164         }
165     };
166 
167 // ATTRIBUTES
168     DefaultSpeed    defaultSpeed;
169     SurfaceMaxSpeed surfaceMaxSpeed;
170     OtherEdgeCosts  otherEdgeCosts;
171 
172 // ACCESS
173 // CONSTANTS
174 
175 private:
176 };
177 
178 // INLINE METHODS
179 //
180 
181 // EXTERNAL REFERENCES
182 //
183 
184 #endif /* CONFIG_COSTCONFIG_H_ */
```

D.3.8 DatabaseConfig.h

```
/**  Data structure for configuration of database connection.
 *
 * #include "DatabaseConfig.h"
 *
 * @author  Jonas Bergman
 */
#ifndef CONFIG_DATABASECONFIG_H_
#define CONFIG_DATABASECONFIG_H_

// SYSTEM INCLUDES
//
#include <string>
#include <sstream>

// PROJECT INCLUDES
//

// LOCAL INCLUDES
//
#include "TopologyConfig.h"

// FORWARD REFERENCES
//

/** A simple data structure for holding the configuration
 * for database connections.
 */
struct DatabaseConfig
{
// ATTRIBUTES
    std::string        hostname;
    int                port;
    std::string        username;
    std::string        password;
    std::string        database;


// OPERATIONS
    /** Construct a connection string from the attributes.
     * @return    A valid connection string for 'pqxx::conn()'
     */
    std::string     getConnectionString() const
    {
        std::ostringstream oss;
        oss << "host=" << hostname
            << " port=" << port
            << " user=" << username
            << " password=" << password
            << " dbname=" << database;
        return  oss.str();
    }

private:
};

// INLINE METHODS
```

```
57 //
58
59 // EXTERNAL REFERENCES
60 //
61
62 #endif /* CONFIG_DATABASECONFIG_H_ */
```

D.3.9 TopologyConfig.h

```
1  /**  Data structure for configuration of topology data in database.
2   *
3   * #include "TopologyConfig.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef CONFIG_TOPOLOGYCONFIG_H_
9  #define CONFIG_TOPOLOGYCONFIG_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <string>
14
15 // PROJECT INCLUDES
16 //
17
18 // LOCAL INCLUDES
19 //
20
21 // FORWARD REFERENCES
22 //
23
24 /**
25  * Information about names in the database for topology data.
26  */
27 struct TopologyConfig
28 {
29 // ATTRIBUTES
30     std::string     providerName;
31
32     std::string     tempTopoName;
33     std::string     topoName;
34
35     std::string     roadsPrefix;
36     std::string     topologySchemaPrefix;
37
38     int             srid;
39     double          tolerance;
40
41     std::string     edgeTableName;
42     std::string     edgeIdColumnName;
43     std::string     sourceColumnName;
44     std::string     targetColumnName;
45     std::string     edgeGeomColumnName;
46
47     std::string     vertexTableName;
48     std::string     vertexIdColumnName;
```

```
49      std::string     vertexGeomColumnName;
50
51      std::string     testFile;
52
53  // CONSTANTS
54      static constexpr const char* PROVIDER_POSTGIS = "postgis";
55      static constexpr const char* PROVIDER_PGROUTING = "pgrouting";
56      static constexpr const char* PROVIDER_JSONTEST = "jsontest";
57      static constexpr const char* TEMP_TOPO_NAMEBASE = "epoch_ms";
58
59
60  };
61
62  // INLINE METHODS
63  //
64
65  // EXTERNAL REFERENCES
66  //
67
68  #endif /* CONFIG_TOPOLOGYCONFIG_H_ */
```

D.3.10 VehicleConfig.h

```
1   /**  Data structure for configuration of vehicle we are routing.
2    *
3    * #include "VehicleConfig.h"
4    *
5    * @author  Jonas Bergman
6    */
7   #ifndef CONFIG_VEHICLECONFIG_H_
8   #define CONFIG_VEHICLECONFIG_H_
9
10  // SYSTEM INCLUDES
11  //
12  #include <string>
13
14  // PROJECT INCLUDES
15  //
16
17  // LOCAL INCLUDES
18  //
19  #include "../osm/OsmVehicle.h"
20
21  // FORWARD REFERENCES
22  //
23
24  /** A simple data structure for holding the configuration
25   *  of the vehicle we are routing.
26   */
27  struct VehicleConfig
28  {
29  // ATTRIBUTES
30      OsmVehicle::VehicleType     category;
31      double                      height;
32      double                      length;
33      double                      weight;
34      double                      width;
```

```
35      unsigned                    maxspeed;
36      unsigned                    acceleration; // seconds 0 - 100 km/h
37      unsigned                    deceleration; // seconds 100 - 0 km/h
38  };
39
40  // INLINE METHODS
41  //
42
43  // EXTERNAL REFERENCES
44  //
45
46  #endif /* CONFIG_VEHICLECONFIG_H_ */
```

D.3.11 ConfigurationReader_test.cc

```
1   /*
2    * ConfigurationReader_test.cc
3    * @author  Jonas Bergman
4    */
5
6   #include "../../catchtest/catch.hpp"
7   #include "../Configuration.h"
8   #include "../ConfigurationReader.h"
9   #include "../ConfigurationException.h"
10
11  SCENARIO ("Use ConfigurationReader to read configuration from json file",
12      "[json],[config]")
13  {
14      //-------------------------------------------------------------------------
15      GIVEN ("a filename to a valid configuration file")
16      {
17          std::string filename("catchtest/testsettings/testsettings.json");
18
19          WHEN ("asking for database configuration")
20          {
21              ConfigurationReader config_reader(filename);
22              Configuration config;
23              config_reader.fillConfiguration(config);
24              const DatabaseConfig& r_db_config = config.getDatabaseConfig();
25
26              THEN ("we get a  database configuration filled out")
27              {
28                  REQUIRE (r_db_config.hostname == "127.0.0.1");
29                  REQUIRE (r_db_config.port == 5432);
30                  REQUIRE (r_db_config.username == "tester");
31                  REQUIRE (r_db_config.password == "tester");
32                  REQUIRE (r_db_config.database == "mikh_0530");
33              }
34          }
35      }
36
37      //-------------------------------------------------------------------------
38      GIVEN ("a filename to a configuration file with missing information")
39      {
40          std::string filename("catchtest/testsettings/testsettings-missing-name.json");
41
42          WHEN ("asking for database configuration")
```

```
43          {
44              ConfigurationReader config_reader(filename);
45
46              THEN ("we get an exception")
47              {
48                  Configuration config;
49                  REQUIRE_THROWS_AS (config_reader.fillConfiguration(config),
50                      ConfigurationException&);
51              }
52          }
53      }
54
55      //-------------------------------------------------------------------------
56      GIVEN ("a filename to a non-existing file")
57      {
58          std::string filename("config/catchtest/foo.json");
59
60          WHEN ("asking for database configuration")
61          {
62              THEN ("we get an exception")
63              {
64                  REQUIRE_THROWS_AS (ConfigurationReader config_reader(filename),
65                      ConfigurationException&);
66              }
67          }
68      }
69
70      //-------------------------------------------------------------------------
71      GIVEN ("a filename to a valid configuration file")
72      {
73          std::string filename("catchtest/testsettings/testsettings.json");
74          ConfigurationReader config_reader(filename);
75          Configuration config;
76          config_reader.fillConfiguration(config);
77
78          //.....................................................................
79          WHEN ("asking for topology configuration")
80          {
81              const TopologyConfig& r_topo_config = config.getTopologyConfig();
82
83              THEN ("we get a topology configuration filled out")
84              {
85                  REQUIRE (r_topo_config.providerName == "postgis");
86                  REQUIRE (r_topo_config.topoName == "lgu");
87                  REQUIRE (r_topo_config.roadsPrefix == "highways");
88                  REQUIRE (r_topo_config.topologySchemaPrefix == "topo");
89                  REQUIRE (r_topo_config.tempTopoName == "");
90                  REQUIRE (r_topo_config.srid == 900913);
91                  REQUIRE (r_topo_config.tolerance == Approx(1.0));
92
93                  REQUIRE (r_topo_config.edgeTableName == "edge_data");
94                  REQUIRE (r_topo_config.edgeIdColumnName == "edge_id");
95                  REQUIRE (r_topo_config.sourceColumnName == "start_node");
96                  REQUIRE (r_topo_config.targetColumnName == "end_node");
97                  REQUIRE (r_topo_config.edgeGeomColumnName == "geom");
98                  REQUIRE (r_topo_config.vertexTableName == "node");
99                  REQUIRE (r_topo_config.vertexIdColumnName == "node_id");
```

```
100                 REQUIRE (r_topo_config.vertexGeomColumnName == "geom");
101             }
102         }
103
104         //.....................................................................
105         WHEN ("asking for vehicle configuration")
106         {
107             const VehicleConfig& r_vehicle_config = config.getVehicleConfig();
108
109             THEN ("we get a vehicle configuration filled out")
110             {
111                 REQUIRE (r_vehicle_config.category == OsmVehicle::MOTORCAR);
112                 REQUIRE (r_vehicle_config.height == Approx(1.6));
113                 REQUIRE (r_vehicle_config.length == Approx(4.5));
114                 REQUIRE (r_vehicle_config.weight == Approx(2.0));
115                 REQUIRE (r_vehicle_config.width == Approx(1.9));
116                 REQUIRE (r_vehicle_config.maxspeed == 200);
117                 REQUIRE (r_vehicle_config.acceleration == 10);
118                 REQUIRE (r_vehicle_config.deceleration == 7);
119             }
120         }
121
122         //.....................................................................
123         WHEN ("asking for cost configuration")
124         {
125             const CostConfig& r_cost_config = config.getCostConfig();
126
127             THEN ("we get a cost configuration filled out")
128             {
129                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
130                     OsmHighway::MOTORWAY, CostConfig::DefaultSpeed::HIGH) == 110);
131                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
132                     OsmHighway::MOTORWAY, CostConfig::DefaultSpeed::LOW) == 90);
133                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
134                     OsmHighway::MOTORWAY_LINK, CostConfig::DefaultSpeed::HIGH) == 90);
135                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
136                     OsmHighway::MOTORWAY_LINK, CostConfig::DefaultSpeed::LOW) == 90);
137
138                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
139                     OsmHighway::TRUNK, CostConfig::DefaultSpeed::HIGH) == 90);
140                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
141                     OsmHighway::TRUNK, CostConfig::DefaultSpeed::LOW) == 60);
142                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
143                     OsmHighway::TRUNK_LINK, CostConfig::DefaultSpeed::HIGH) == 90);
144                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
145                     OsmHighway::TRUNK_LINK, CostConfig::DefaultSpeed::LOW) == 60);
146
147                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
148                     OsmHighway::PRIMARY, CostConfig::DefaultSpeed::HIGH) == 90);
149                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
150                     OsmHighway::PRIMARY, CostConfig::DefaultSpeed::LOW) == 60);
151                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
152                     OsmHighway::PRIMARY_LINK, CostConfig::DefaultSpeed::HIGH) == 90);
153                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
154                     OsmHighway::PRIMARY_LINK, CostConfig::DefaultSpeed::LOW) == 60);
155
156                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
```

```
157                     OsmHighway::SECONDARY, CostConfig::DefaultSpeed::HIGH) == 90);
158                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
159                     OsmHighway::SECONDARY, CostConfig::DefaultSpeed::LOW) == 60);
160                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
161                     OsmHighway::SECONDARY_LINK, CostConfig::DefaultSpeed::HIGH) == 90);
162                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
163                     OsmHighway::SECONDARY_LINK, CostConfig::DefaultSpeed::LOW) == 60);
164
165                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
166                     OsmHighway::TERTIARY, CostConfig::DefaultSpeed::HIGH) == 90);
167                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
168                     OsmHighway::TERTIARY, CostConfig::DefaultSpeed::LOW) == 60);
169                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
170                     OsmHighway::TERTIARY_LINK, CostConfig::DefaultSpeed::HIGH) == 90);
171                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
172                     OsmHighway::TERTIARY_LINK, CostConfig::DefaultSpeed::LOW) == 60);
173
174                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
175                     OsmHighway::UNCLASSIFIED, CostConfig::DefaultSpeed::HIGH) == 90);
176                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
177                     OsmHighway::UNCLASSIFIED, CostConfig::DefaultSpeed::LOW) == 60);
178
179                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
180                     OsmHighway::RESIDENTIAL, CostConfig::DefaultSpeed::HIGH) == 90);
181                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
182                     OsmHighway::RESIDENTIAL, CostConfig::DefaultSpeed::LOW) == 60);
183
184                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
185                     OsmHighway::SERVICE, CostConfig::DefaultSpeed::HIGH) == 40);
186                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
187                     OsmHighway::SERVICE, CostConfig::DefaultSpeed::LOW) == 20);
188
189                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
190                     OsmHighway::LIVING_STREET, CostConfig::DefaultSpeed::HIGH) == 20);
191                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
192                     OsmHighway::LIVING_STREET, CostConfig::DefaultSpeed::LOW) == 20);
193
194                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
195                     OsmHighway::BUS_GUIDEWAY, CostConfig::DefaultSpeed::HIGH) == 80);
196                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
197                     OsmHighway::BUS_GUIDEWAY, CostConfig::DefaultSpeed::LOW) == 60);
198
199                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
200                     OsmHighway::ROAD, CostConfig::DefaultSpeed::HIGH) == 80);
201                 REQUIRE (r_cost_config.defaultSpeed.getDefaultSpeed(
202                     OsmHighway::ROAD, CostConfig::DefaultSpeed::LOW) == 50);
203
204                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
205                     OsmHighway::PAVED) == 1000);
206                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
207                     OsmHighway::ASPHALT) == 1000);
208                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
209                     OsmHighway::COBBLESTONE) == 20);
210                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
211                     OsmHighway::COBBLESTONE_FLATTENED) == 40);
212                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
213                     OsmHighway::SETT) == 40);
```

```
214                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
215                     OsmHighway::CONCRETE) == 1000);
216                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
217                     OsmHighway::CONCRETE_LANES) == 40);
218                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
219                     OsmHighway::CONCRETE_PLATES) == 100);
220                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
221                     OsmHighway::PAVING_STONES) == 40);
222                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
223                     OsmHighway::METAL) == 60);
224                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
225                     OsmHighway::WOOD) == 30);
226                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
227                     OsmHighway::UNPAVED) == 60);
228                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
229                     OsmHighway::COMPACTED) == 70);
230                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
231                     OsmHighway::DIRT) == 40);
232                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
233                     OsmHighway::EARTH) == 40);
234                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
235                     OsmHighway::FINE_GRAVEL) == 50);
236                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
237                     OsmHighway::GRASS) == 10);
238                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
239                     OsmHighway::GRASS_PAVER) == 20);
240                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
241                     OsmHighway::GRAVEL) == 60);
242                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
243                     OsmHighway::GROUND) == 20);
244                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
245                     OsmHighway::ICE) == 70);
246                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
247                     OsmHighway::MUD) == 5);
248                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
249                     OsmHighway::PEBBLESTONE) == 50);
250                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
251                     OsmHighway::SALT) == 70);
252                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
253                     OsmHighway::SAND) == 70);
254                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
255                     OsmHighway::SNOW) == 50);
256                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
257                     OsmHighway::WOODCHIPS) == 5);
258                 REQUIRE (r_cost_config.surfaceMaxSpeed.getSurfaceMaxSpeed(
259                     OsmHighway::METAL_GRID) == 40);
260
261                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
262                     "highway=bus_stop") == 5);
263                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
264                     "highway=crossing") == 5);
265                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
266                     "highway=give_way") == 20);
267                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
268                     "highway=mini_roundabout") == 20);
269                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
270                     "highway=stop") == 30);
```

```
271                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
272                     "highway=traffic_signals") == 30);
273
274                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
275                     "railway=level_crossing") == 20);
276
277                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
278                     "public_transport=stop_position") == 5);
279
280                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
281                     "traffic_calming=yes") == 10);
282                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
283                     "traffic_calming=bump") == 10);
284                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
285                     "traffic_calming=table") == 10);
286                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
287                     "traffic_calming=cushion") == 10);
288                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
289                     "traffic_calming=rumble_strip") == 10);
290                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
291                     "traffic_calming=chicane") == 10);
292                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
293                     "traffic_calming=choker") == 10);
294                 REQUIRE(r_cost_config.otherEdgeCosts.getOtherCost(
295                     "traffic_calming=island") == 5);
296             }
297         }
298
299         WHEN ("asking for access rules")
300         {
301             const OsmAccess::AccessRule r_access_rule = config.getAccessRule();
302
303             THEN ("we get an AccessRule filled out")
304             {
305                 std::vector<OsmAccess::AccessType> types =
306                     r_access_rule.allowAccessToTypes;
307                 REQUIRE (types.size() == 3);
308
309                 auto it = std::find(types.begin(), types.end(),
310                     OsmAccess::AccessType::YES);
311                 INFO ("Allows access:" + OsmAccess::toString(*it));
312                 REQUIRE (it != types.end());
313
314                 it = std::find(types.begin(), types.end(),
315                     OsmAccess::AccessType::PERMISSIVE);
316                 INFO ("Allows access:" + OsmAccess::toString(*it));
317                 REQUIRE (it != types.end());
318
319                 it = std::find(types.begin(), types.end(),
320                     OsmAccess::AccessType::DESIGNATED);
321                 INFO ("Allows access:" + OsmAccess::toString(*it));
322                 REQUIRE (it != types.end());
323
324                 it = std::find(types.begin(), types.end(),
325                     OsmAccess::AccessType::NO);
326                 INFO ("Denies access: no");
327                 REQUIRE (it == types.end());
```

```
328             }
329         }
330
331         WHEN ("asking for restrictions rules")
332         {
333             const OsmBarrier::RestrictionsRule
334             r_restrict_rule = config.getBarrierRestrictionsRule();
335
336             THEN ("we get RestrictionsRule filled out")
337             {
338                 std::vector<OsmBarrier::BarrierType> types =
339                     r_restrict_rule.restrictionTypes;
340                 REQUIRE (types.size() == 21);
341
342                 auto it = std::find(types.begin(), types.end(),
343                     OsmBarrier::BarrierType::BLOCK);
344                 INFO ("Restrict: " + OsmBarrier::toString(*it));
345                 REQUIRE (it != types.end());
346
347                 it = std::find(types.begin(), types.end(),
348                     OsmBarrier::BarrierType::BOLLARD);
349                 INFO ("Restrict: " + OsmBarrier::toString(*it));
350                 REQUIRE (it != types.end());
351
352                 it = std::find(types.begin(), types.end(),
353                     OsmBarrier::BarrierType::BUS_TRAP);
354                 INFO ("Restrict: " + OsmBarrier::toString(*it));
355                 REQUIRE (it != types.end());
356
357                 it = std::find(types.begin(), types.end(),
358                     OsmBarrier::BarrierType::CHAIN);
359                 INFO ("Restrict: " + OsmBarrier::toString(*it));
360                 REQUIRE (it != types.end());
361
362                 it = std::find(types.begin(), types.end(),
363                     OsmBarrier::BarrierType::CYCLE_BARRIER);
364                 INFO ("Restrict: " + OsmBarrier::toString(*it));
365                 REQUIRE (it != types.end());
366
367                 it = std::find(types.begin(), types.end(),
368                     OsmBarrier::BarrierType::DEBRIS);
369                 INFO ("Restrict: " + OsmBarrier::toString(*it));
370                 REQUIRE (it != types.end());
371
372                 it = std::find(types.begin(), types.end(),
373                     OsmBarrier::BarrierType::FULLHEIGHT_TURNSTILE);
374                 INFO ("Restrict: " + OsmBarrier::toString(*it));
375                 REQUIRE (it != types.end());
376
377                 it = std::find(types.begin(), types.end(),
378                     OsmBarrier::BarrierType::HORSE_STILE);
379                 INFO ("Restrict: " + OsmBarrier::toString(*it));
380                 REQUIRE (it != types.end());
381
382                 it = std::find(types.begin(), types.end(),
383                     OsmBarrier::BarrierType::JERSEY_BARRIER);
384                 INFO ("Restrict: " + OsmBarrier::toString(*it));
```

```
385                 REQUIRE (it != types.end());
386
387                 it = std::find(types.begin(), types.end(),
388                     OsmBarrier::BarrierType::KENT_CARRIAGE_GAP);
389                 INFO ("Restrict: " + OsmBarrier::toString(*it));
390                 REQUIRE (it != types.end());
391
392                 it = std::find(types.begin(), types.end(),
393                     OsmBarrier::BarrierType::KISSING_GATE);
394                 INFO ("Restrict: " + OsmBarrier::toString(*it));
395                 REQUIRE (it != types.end());
396
397                 it = std::find(types.begin(), types.end(),
398                     OsmBarrier::BarrierType::LOG);
399                 INFO ("Restrict: " + OsmBarrier::toString(*it));
400                 REQUIRE (it != types.end());
401
402                 it = std::find(types.begin(), types.end(),
403                     OsmBarrier::BarrierType::MOTORCYCLE_BARRIER);
404                 INFO ("Restrict: " + OsmBarrier::toString(*it));
405                 REQUIRE (it != types.end());
406
407                 it = std::find(types.begin(), types.end(),
408                     OsmBarrier::BarrierType::ROPE);
409                 INFO ("Restrict: " + OsmBarrier::toString(*it));
410                 REQUIRE (it != types.end());
411
412                 it = std::find(types.begin(), types.end(),
413                     OsmBarrier::BarrierType::SALLY_PORT);
414                 INFO ("Restrict: " + OsmBarrier::toString(*it));
415                 REQUIRE (it != types.end());
416
417                 it = std::find(types.begin(), types.end(),
418                     OsmBarrier::BarrierType::SPIKES);
419                 INFO ("Restrict: " + OsmBarrier::toString(*it));
420                 REQUIRE (it != types.end());
421
422                 it = std::find(types.begin(), types.end(),
423                     OsmBarrier::BarrierType::STILE);
424                 INFO ("Restrict: " + OsmBarrier::toString(*it));
425                 REQUIRE (it != types.end());
426
427                 it = std::find(types.begin(), types.end(),
428                     OsmBarrier::BarrierType::SUMP_BUSTER);
429                 INFO ("Restrict: " + OsmBarrier::toString(*it));
430                 REQUIRE (it != types.end());
431
432                 it = std::find(types.begin(), types.end(),
433                     OsmBarrier::BarrierType::SWING_GATE);
434                 INFO ("Restrict: " + OsmBarrier::toString(*it));
435                 REQUIRE (it != types.end());
436
437                 it = std::find(types.begin(), types.end(),
438                     OsmBarrier::BarrierType::TURNSTILE);
439                 INFO ("Restrict: " + OsmBarrier::toString(*it));
440                 REQUIRE (it != types.end());
441
```

```
442                 it = std::find(types.begin(), types.end(),
443                     OsmBarrier::BarrierType::YES);
444                 INFO ("Restrict: " + OsmBarrier::toString(*it));
445                 REQUIRE (it != types.end());
446 
447                 it = std::find(types.begin(), types.end(),
448                     OsmBarrier::BarrierType::GATE);
449                 INFO ("Allow: gate");
450                 REQUIRE (it == types.end());
451             }
452         }
453 
454         WHEN ("asking for costs rules")
455         {
456             const OsmBarrier::CostsRule r_costs_rule = config.getBarrierCostsRule();
457 
458             THEN ("we get CostssRule filled out")
459             {
460                 REQUIRE (r_costs_rule.costs.size() == 13);
461 
462                 INFO("Costs: border control");
463                 REQUIRE (r_costs_rule.getCost(
464                     OsmBarrier::BarrierType::BORDER_CONTROL) == 120);
465 
466                 INFO("Costs: bump gate");
467                 REQUIRE (r_costs_rule.getCost(
468                     OsmBarrier::BarrierType::BUMP_GATE) == 30);
469 
470                 INFO("Costs: bus trap");
471                 REQUIRE (r_costs_rule.getCost(
472                     OsmBarrier::BarrierType::BUS_TRAP) == 30);
473 
474                 INFO("Costs: cattle grid");
475                 REQUIRE (r_costs_rule.getCost(
476                     OsmBarrier::BarrierType::CATTLE_GRID) == 20);
477 
478                 INFO("Costs: entrance");
479                 REQUIRE (r_costs_rule.getCost(
480                     OsmBarrier::BarrierType::ENTRANCE) == 10);
481 
482                 INFO("Costs: gate");
483                 REQUIRE (r_costs_rule.getCost(
484                     OsmBarrier::BarrierType::GATE) == 30);
485 
486                 INFO("Costs: hampshire gate");
487                 REQUIRE (r_costs_rule.getCost(
488                     OsmBarrier::BarrierType::HAMPSHIRE_GATE) == 60);
489 
490                 INFO("Costs: height restrictor");
491                 REQUIRE (r_costs_rule.getCost(
492                     OsmBarrier::BarrierType::HEIGHT_RESTRICTOR) == 5);
493 
494                 INFO("Costs: jersey barrier");
495                 REQUIRE (r_costs_rule.getCost(
496                     OsmBarrier::BarrierType::JERSEY_BARRIER) == 10);
497 
498                 INFO("Costs: lift gate");
```

```
499                 REQUIRE (r_costs_rule.getCost(
500                     OsmBarrier::BarrierType::LIFT_GATE) == 60);
501
502                 INFO("Costs: sump buster");
503                 REQUIRE (r_costs_rule.getCost(
504                     OsmBarrier::BarrierType::SUMP_BUSTER) == 30);
505
506                 INFO("Costs: swing gate");
507                 REQUIRE (r_costs_rule.getCost(
508                     OsmBarrier::BarrierType::SWING_GATE) == 60);
509
510                 INFO("Costs: toll both");
511                 REQUIRE (r_costs_rule.getCost(
512                     OsmBarrier::BarrierType::TOLL_BOOTH) == 40);
513
514                 INFO("No cost: yes");
515                 REQUIRE (r_costs_rule.costsToPass(
516                     OsmBarrier::BarrierType::YES) == false);
517             }
518         }
519     }
520 }
```

D.4 doc

D.4.1 README.md

```
Documentation
=============

This directory contains a directory for the project's report, and a directory for
↪  the UML diagrams.
```

D.5 report

D.5.1 README.md

```
Report
======

The originals for the project's report. Written in
↪  [ShareLaTeX](https://www.sharelatex.com).
```

D.6 uml

D.6.1 README.md

```
UML
===

Class and sequence diagrams to get an idea of the concepts, although not 100%
↪  accurate.

Diagrams are created in `draw.io`, exported as  `.svg` files, opened in
↪  `Inkscape` and saved as `.pdf` files (for usage in report).
```

D.7 graph

D.7.1 README.md

```
graph
=====

The `graph` package consists of classes for representing graphs.

## GraphBuilder
The GraphBuilder is responsible for buildign graphs and linegraphs. It takes a
 ↪  `Topology` and a `Configuration`and uses them for building a `graph` and
 ↪  `linegraph` based on [Boost adjacency
 ↪  lists](http://www.boost.org/doc/libs/1_54_0/libs/graph/doc/adjacency_list.html).
 ↪  The `GraphBuilder` class holds several `maps` that connects the original
 ↪  Edges and Vertices to those used internally in the Boost graphs, so that it
 ↪  is possible to backtrack information about those elements. The internal Boost
 ↪  types keeps some properties
 ↪  ["bundled"](http://www.boost.org/doc/libs/1_54_0/libs/graph/doc/bundles.html),
 ↪  instead of as "interior" properties.

The ordinary `graph` is a directed graph that connects the _**vertices**_ and
 ↪  _**edges**_ from the topology. The `linegraph` transforms that graph to an
 ↪  edge-based graph that turns the graph's edges into _**nodes**_ in the
 ↪  linegraph, and those edges are connected with _**lines**_.

### Topology
`Topology` is a class holding `Edges` and `Vertices` for the topology fetched
 ↪  from the `MapProvider`. It simply states which `Vertices` are connected by
 ↪  which `Edges`, without any costs or restrictions or directions. When created
 ↪  it validates that the `source` and `target` Vertices of the Edges actually
 ↪  exists in the topology.

### Edge
The Edge holds some relevant data from the topology. It has an `id`, and a field
 ↪  for which the original `osm_id` was before building the database topology. It
 ↪  also holds id to `source` Vertex and id to `target` Vertex, some data about
 ↪  the geometry and the "road" such as number of lanes, a structure for costs
 ↪  and optionally for restrictions.

### EdgeCost
The cost for travel among an edge is the number of seconds it takes. The base for
 ↪  this calculation is of course dependent on the length of the edge, and the
 ↪  speed. The speed an be set as an explicit `maxspeed` restriction, or by
 ↪  looking up the configuration for a `surface` if such is stated, else the
 ↪  speed is found by a look up for the default speed for the "highway type"
 ↪  (road category).

The travel time cost can than be modified by barriers, speed bumps, traffic
 ↪  lights ... on the edge (or points that can be applied on the edge).

### EdgeRestriction
The `EdgeRestriction` keeps track of restrictions that can be imposed on
an edge. Those restrictions are:

- **Vehicle properties**: weight, height, length, width, maxspeed.
- **General access**: [OSM wiki for
 ↪  access](http://wiki.openstreetmap.org/wiki/Key:access).
```

```
- **Vehicle type access**: as for General access, but specified for a category of
 ↪  vehicles, such as `motorcar` or `goods`.
- **Barrier**: if the edge is blocked with some kind of barrier.
- **Turning restrictions**: [OSM wiki for turn
 ↪  restrictions](http://wiki.openstreetmap.org/wiki/Relation:restriction).
- **Disused**: if the edge (road) is marked as no longer in use.
- **NoExit**: if the edge has no exit, it should not be used for building a
 ↪  linegraph.

(Turn restriction via other edges and not just via a vertex are difficult. At the
 ↪  time when converting the topology to a line graph it is impossible to have the
 ↪  relevant information. The solution is to set a flag on the Edge that there
 ↪  exist a VIA_WAY restriction that must be taken into account when routing, and
 ↪  the routing module must look up and make its own decisions somehow.)

### Vertex
The Vertex class is simple with just an `id` and a `point` location.

### TurnCostCalculator
This is a static helper class that assist in calculating the cost of making
 ↪  turns, when transforming the _graph_ into a _linegraph_. The cost for making
 ↪  turns are dependent on the angle of the turn, the category of the roads and
 ↪  the size of the vehicle. Other factors can be added.

### Exceptions
`GraphException` is the main public exception to be thrown from this package.
 ↪  `RestrictionsException` and `TopologyException` are thrown when building
 ↪  those classes, but not as exposed externally.
```

D.7.2 Edge.h

```
1  /**  Data structure for edges in Topology.
2   *
3   * #include "Edge.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef GRAPH_EDGE_H_
9  #define GRAPH_EDGE_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <limits>
14
15 // PROJECT INCLUDES
16 //
17
18 // LOCAL INCLUDES
19 //
20 #include "Vertex.h"
21 #include "../config/Configuration.h"
22 #include "../osm/OsmHighway.h"
23 #include "../osm/OsmId.h"
24 #include "EdgeCost.h"
25 #include "Speed.h"
26
```

```
27  // FORWARD REFERENCES
28  //
29  typedef long    EdgeIdType;
30  class           EdgeRestriction;
31
32  /**
33   * Data structure for edges in the topology.
34   */
35  class Edge
36  {
37  public:
38  // TYPES
39      //-------------------------------------------------------------------------
40      enum DirectionType
41      {
42          BOTH,       // bidirectional
43          TO_FROM,    // one-way: from Source to Target
44          FROM_TO     // one-way: from Target to Source
45      };
46
47      //-------------------------------------------------------------------------
48      /** A data structure for geometric information for the Edge.
49       * Bearing is the compass direction in degrees at source and at target.
50       */
51      struct GeomData
52      {
53          double      length {1.0};
54          Point       centerPoint;
55          int         sourceBearing {0};
56          int         targetBearing {0};
57
58          /** Constructor. */
59          GeomData() = default;
60
61          /** Constructor. */
62          GeomData(double length,
63                   Point  centerPoint,
64                   int    sourceBearing,
65                   int    targetBearing);
66      };
67
68      //-------------------------------------------------------------------------
69      /** A data structure for road related information for the Edge.
70       */
71      struct RoadData
72      {
73          DirectionType               direction {BOTH};
74          size_t                      nrLanes   {1};
75          OsmHighway::HighwayType     roadType  {OsmHighway::HighwayType::ROAD};
76
77          /** Constructor. */
78          RoadData() = default;
79
80          /** Constructor. */
81          RoadData(DirectionType direction, size_t nrLanes);
82
83          /** Print this information. */
```

```
84          void print(std::ostream& os) const;
85      };
86
87      static const EdgeIdType MAX_ID;
88
89  // LIFECYCLE
90      /** Constructor.
91       */
92      Edge() = delete;
93
94      /** Constructor.
95       * @param        id                    Id for this Edge
96       * @param   osmId       The original OsmId this edge belongs to.
97       * @param        source            Source vertex
98       * @param        target            Target vertex
99       * @param   geomData    Geometric data for the edge.
100      * @param   roadData    Road data for the edge.
101      */
102     Edge(EdgeIdType      id,
103          OsmIdType       osmId,
104          VertexIdType    source,
105          VertexIdType    target,
106          GeomData        geomData,
107          RoadData        roadData);
108
109     /** Constructor.
110      * Using default values for geometry and road.
111      * @param        id                Id for this Edge
112      * @param   osmId   The original OsmId this edge belongs to.
113      * @param        source        Source vertex
114      * @param        target        Target vertex
115      */
116     Edge(EdgeIdType      id,
117          OsmIdType       osmId,
118          VertexIdType    source,
119          VertexIdType    target);
120
121     /** Move constructor.
122      * @param   from    The Edge to make a move of.
123      */
124     Edge(Edge&& from);
125
126     /** Destructor.
127      */
128     ~Edge();
129
130 // OPERATORS
131     /** Textual output of Edge.
132      */
133     friend
134     std::ostream&     operator<<(std::ostream& os, const Edge& rEdge);
135
136 // OPERATIONS
137     /** Set the Geometric data for this edge.
138      * @param   geomData    The GeomData to use.
139      */
140     void              setGeomData(GeomData geomData);
```

```
141
142     /** Set the Road data for this edge.
143      * @param   roadData    The RoadData to use.
144      */
145     void              setRoadData(RoadData roadData);
146
147     /** Set the OsmId corresponding to this edge.
148      * @param   osmId   The OsmId to set.
149      */
150     void              setOsmId(OsmIdType osmId);
151
152     /** Set the restrictions for this edge.
153      * @param   pRestrictions   The restrictions for this edge.
154      */
155     void              setRestrictions(EdgeRestriction* pRestrictions);
156
157     /** Set the speed for the edge in this actual configuration.
158      * @param   speed   The speed to set in km/h.
159      */
160     void              setSpeed(Speed speed);
161
162     /** Remove the restrictions for this edge.
163      */
164     void              clearCostsAndRestrictions();
165
166     /** Parse a string into an EdgeIdType.
167      *  @param  idString    The string representing the id.
168      *  @return The corresponding edge id.
169      *  @throw  std::invalid_argument
170      *  @throw  std::out_of_range
171      */
172     static EdgeIdType parse(const std::string& idStr);
173
174 // ACCESSORS
175     /**
176      * @return  The id of this edge.
177      */
178     EdgeIdType        id()        const;
179
180     /**
181      * @return  The source vertex for this edge.
182      */
183     VertexIdType      sourceId()  const;
184
185     /**
186      * @return  The target vertex of this edge.
187      */
188     VertexIdType      targetId()  const;
189
190     /**
191      * @return  The original OSM id for this edge.
192      */
193     OsmIdType         osmId()     const;
194
195     /**
196      * @return The geometric data for this edge.
197      */
```

```
198     const GeomData&   geomData()  const;
199
200     /**
201      * @return The road data for this edge.
202      */
203     const RoadData&   roadData()  const;
204
205     /** Get hold of the restrictions associated with the edge.
206      * @return  Reference to EdgeRestriction
207      * @throw   RestrictionException if no restriction is applied on Edge.
208      */
209     EdgeRestriction&  restrictions();
210
211     /** Get hold of the restrictions associated with the edge.
212      * @return  Reference to EdgeRestriction
213      * @throw   RestrictionException if no restriction is applied on Edge.
214      */
215     const EdgeRestriction&
216                       restrictions() const;
217
218     /** Get the structure of different costs for traveling the edge.
219      * @return  Reference to EdgeCost
220      */
221     EdgeCost&         edgeCost();
222
223     /** Get the structure of different costs for traveling the edge.
224      * @return  Reference to EdgeCost
225      */
226     const EdgeCost&   edgeCost() const;
227
228     /**
229      * @return  The cost or weight for this edge.
230      */
231     Cost              cost() const;
232
233     /** The speed must be kept track of because of turn cost calculations,
234      * but they are not  part of `RoadData` which are meant to be constant,
235      * while the speed varies with configuration.
236      * @return  The speed for this edge in km/h
237      */
238     Speed             speed() const;
239
240 // INQUIRY
241     /**
242      * @return  true if there exists restrictions for this edge.
243      */
244     bool              hasRestrictions() const;
245
246     /** An edge needs special attention during routing if there exists
247      * a turning restriction via other ways (edges).
248      * @return  true if there exists a turn restriction via ways.
249      */
250     bool              hasViaWayRestriction() const;
251
252     /** Check if travel on the Edge is restricted given the configuration.
253      * @param   rConfig     Configuration with restriction rules.
254      * @return  true        If travel is restricted.
```

```
255      * @throws  RestrictionsException
256      */
257     bool              isRestricted(const Configuration& rConfig) const;
258 
259 private:
260 // ATTRIBUTES
261     EdgeIdType        mId;      // id in topology
262     OsmIdType         mOsmId;
263     VertexIdType      mSourceId;
264     VertexIdType      mTargetId;
265     GeomData          mGeomData;
266     RoadData          mRoadData;
267     EdgeRestriction*  mpRestrictions;
268     EdgeCost          mCost;
269     Speed             mSpeed;
270 };
271 
272 // INLINE METHODS
273 //
274 
275 // EXTERNAL REFERENCES
276 //
277 
278 #endif /* GRAPH_EDGE_H_ */
```

D.7.3 Edge.cc

```
1  /*
2   * Edge.cc
3   *
4   *@author   Jonas Bergman
5   */
6 
7  #include "Edge.h"  // class implemented
8 
9  #include "EdgeRestriction.h"
10 
11 //============================== TYPES     ===================================
12 const EdgeIdType Edge::MAX_ID = std::numeric_limits<EdgeIdType>::max();
13 
14 // Edge::GeomData ------------------------------------------------------------
15 Edge::GeomData::GeomData(double length,
16                          Point  centerPoint,
17                          int    sourceBearing,
18                          int    targetBearing)
19     : length(length),
20       centerPoint(centerPoint),
21       sourceBearing(sourceBearing),
22       targetBearing(targetBearing)
23 {}
24 
25 // Edge::RoadData ------------------------------------------------------------
26 Edge::RoadData::RoadData(DirectionType direction, size_t nrLanes)
27     : direction(direction), nrLanes(nrLanes)
28 {}
29 
30 void
```

```
31  Edge::RoadData::print(std::ostream& os) const
32  {
33      os  << "direction: ";
34
35      switch(direction)
36      {
37          case Edge::DirectionType::BOTH:
38              os << "BOTH"; break;
39          case Edge::DirectionType::FROM_TO:
40              os << "FROM_TO"; break;
41          case Edge::DirectionType::TO_FROM:
42              os << "TO_FROM"; break;
43      }
44
45      os << ", #lanes: " << nrLanes;
46      os << ", type: " << OsmHighway::toString(roadType);
47  }
48
49
50  // Edge ------------------------------------------------------------------------
51  ////////////////////////////// PUBLIC ///////////////////////////////////////
52
53  //============================ LIFECYCLE ====================================
54  Edge::Edge(EdgeIdType       id,
55             OsmIdType        osmId,
56             VertexIdType     source,
57             VertexIdType     target,
58             Edge::GeomData   geomData,
59             Edge::RoadData   roadData)
60      : mId(id),
61        mOsmId(osmId),
62        mSourceId(source),
63        mTargetId(target),
64        mGeomData(geomData),
65        mRoadData(roadData),
66        mpRestrictions(nullptr),
67        mCost(),
68        mSpeed()
69  { }
70
71  Edge::Edge(EdgeIdType       id,
72             OsmIdType        osmId,
73             VertexIdType     source,
74             VertexIdType     target)
75      : mId(id),
76        mOsmId(osmId),
77        mSourceId(source),
78        mTargetId(target),
79        mGeomData(),
80        mRoadData(),
81        mpRestrictions(nullptr),
82        mCost(),
83        mSpeed()
84  { }
85
86  Edge::Edge(Edge&& from)
87      : mId(from.mId),
```

```
88        mOsmId(from.mOsmId),
89        mSourceId(from.mSourceId),
90        mTargetId(from.mTargetId),
91        mGeomData(from.mGeomData),
92        mRoadData(from.mRoadData),
93        mpRestrictions(from.mpRestrictions),
94        mCost(),
95        mSpeed()
96  {
97      from.mpRestrictions = nullptr;
98  }
99
100 Edge::~Edge()
101 {
102     delete mpRestrictions;
103 }
104
105 //============================== OPERATORS ===================================
106 std::ostream&
107 operator<<(std::ostream& os, const Edge& rEdge)
108 {
109     os  << "Edge [id: " << rEdge.id()
110         << ", osmId: " << rEdge.osmId()
111         << ", source: " << rEdge.sourceId()
112         << ", target: " << rEdge.targetId()
113         << ", cost: " << rEdge.cost()
114         << ", length: " << rEdge.geomData().length
115         << ", speed: " << rEdge.speed()
116         << "\n   road data: ";
117     rEdge.roadData().print(os);
118
119     os  << "]";
120
121     return os;
122 }
123
124
125 //============================== OPERATIONS ==================================
126
127 void
128 Edge::setGeomData(Edge::GeomData geomData)
129 { mGeomData = geomData; }
130
131 void
132 Edge::setRoadData(Edge::RoadData roadData)
133 { mRoadData = roadData; }
134
135 void
136 Edge::setOsmId(OsmIdType osmId)
137 { mOsmId = osmId; }
138
139 void
140 Edge::setRestrictions(EdgeRestriction* pRestrictions)
141 {
142     delete mpRestrictions;
143     mpRestrictions = pRestrictions;
144 }
```

```
145
146 void
147 Edge::setSpeed(Speed speed)
148 {
149     mSpeed = speed;
150 }
151
152 void
153 Edge::clearCostsAndRestrictions()
154 {
155     mCost.clearCosts();
156
157     delete mpRestrictions;
158     mpRestrictions = nullptr;
159
160     mSpeed = 0;
161 }
162
163 //static
164 EdgeIdType
165 Edge::parse(const std::string& idStr)
166 {
167     return static_cast<EdgeIdType>(std::stoul(idStr));
168 }
169
170 //============================== ACESS      ===================================
171 EdgeIdType
172 Edge::id() const
173 { return mId; }
174
175 VertexIdType
176 Edge::sourceId() const
177 { return mSourceId; }
178
179 VertexIdType
180 Edge::targetId() const
181 { return mTargetId; }
182
183 OsmIdType
184 Edge::osmId() const
185 { return mOsmId; }
186
187 const Edge::GeomData&
188 Edge::geomData() const
189 { return mGeomData; }
190
191 const Edge::RoadData&
192 Edge::roadData() const
193 { return mRoadData; }
194
195
196 EdgeRestriction&
197 Edge::restrictions()
198 {
199     if(mpRestrictions == nullptr) {
200         mpRestrictions = new EdgeRestriction();
201     }
```

```
202     return *mpRestrictions;
203 }
204
205 const EdgeRestriction&
206 Edge::restrictions() const
207 {
208     if(mpRestrictions == nullptr) {
209         throw RestrictionsException(std::string("No restriction on edge ")
210             + std::to_string(mId));
211     }
212     return *mpRestrictions;
213 }
214
215 EdgeCost&
216 Edge::edgeCost()
217 {
218     return mCost;
219 }
220
221 const EdgeCost&
222 Edge::edgeCost() const
223 {
224     return mCost;
225 }
226
227 Cost
228 Edge::cost() const
229 {
230     return mCost.getCost();
231 }
232
233 Speed
234 Edge::speed() const
235 {
236     return mSpeed;
237 }
238
239 //============================== INQUIRY    ==================================
240 bool
241 Edge::hasRestrictions() const
242 { return mpRestrictions != nullptr; }
243
244 bool
245 Edge::hasViaWayRestriction() const
246 {
247     if(hasRestrictions())
248     {
249         return mpRestrictions->hasViaWayRestriction();
250     }
251     return false;
252 }
253
254 bool
255 Edge::isRestricted(const Configuration& rConfig) const
256 {
257     if(mpRestrictions == nullptr)
258     {
```

```
259         return false;
260     }
261 
262     try
263     {
264         return mpRestrictions->restricts(rConfig);
265     }
266     catch (RestrictionsException& re)
267     {
268         re.addEdgeId(std::to_string(mId));
269         throw re;
270     }
271 }
272 
273 ////////////////////////////// PROTECTED  ///////////////////////////////////
274 
275 ////////////////////////////// PRIVATE    ///////////////////////////////////
```

D.7.4 Vertex.h

```
1  /**  Data structure for vertices in Topology.
2   *
3   * #include "Vertex.h"
4   *
5   * @author  Jonas Bergman
6   */
7 
8  #ifndef GRAPH_VERTEX_H_
9  #define GRAPH_VERTEX_H_
10 
11 // SYSTEM INCLUDES
12 //
13 #include <limits>
14 #include <ostream>
15 
16 // PROJECT INCLUDES
17 //
18 
19 // LOCAL INCLUDES
20 //
21 #include "../util/Point.h"
22 
23 // FORWARD REFERENCES
24 //
25 typedef long        VertexIdType;
26 
27 /**
28  * Data structure for vertices in the topology.
29  */
30 class Vertex
31 {
32 public:
33 // TYPES and CONSTANTS
34     static const VertexIdType MAX_ID;
35 // LIFECYCLE
36     /** Constructor.
37      * @param    id               Id for this vertex.
```

```
38          * @param    point             The Point (geometry).
39          */
40         Vertex(VertexIdType id, Point point);
41
42         /** Default constructor. Deleted */
43         Vertex() = delete;
44
45         /** Copy constructor. Default. */
46         Vertex(const Vertex&) = default;
47
48     // OPERATORS
49         friend
50         std::ostream&   operator<<(std::ostream& os, const Vertex& rVertex);
51
52         bool            operator==(const Vertex& rhs) const;
53
54     // OPERATIONS
55     // ACCESS
56         /**
57          * @return  The id of this Vertex.
58          */
59         VertexIdType    id() const;
60
61         /**
62          * @return  The coordinates for this Vertex.
63          */
64         Point           point() const;
65
66     //INQUIRY
67         /**
68          * @return  True if the Vertex has restrictions.
69          */
70         bool            hasRestrictions() const;
71
72     private:
73     // ATTRIBUTES
74         VertexIdType    mId;
75         Point           mPoint;
76     };
77
78     // INLINE METHODS
79     //
80
81     // EXTERNAL REFERENCES
82     //
83
84     #endif /* GRAPH_VERTEX_H_ */
```

D.7.5 Vertex.cc

```
1  /*
2   * Vertex.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "Vertex.h"  // class implemented
```

```
8
9
10  ////////////////////////////// PUBLIC //////////////////////////////////////
11  const VertexIdType Vertex::MAX_ID = std::numeric_limits<VertexIdType>::max();
12
13  //============================== LIFECYCLE ====================================
14  Vertex::Vertex(VertexIdType id, Point point)
15      : mId(id), mPoint(point)
16  {}
17
18  //============================== OPERATORS ====================================
19  std::ostream&
20  operator<<(std::ostream& os, const Vertex& rVertex)
21  {
22      os  << "Vertex [id: " << rVertex.mId
23          << ", point: " << rVertex.mPoint << "]";
24      return os;
25  }
26
27  bool
28  Vertex::operator==(const Vertex& rhs) const
29  {
30      return (rhs.mId == mId) && (rhs.mPoint == mPoint);
31  }
32
33  //============================== OPERATIONS ===================================
34  //============================== ACESS      ===================================
35  VertexIdType
36  Vertex::id() const
37  { return mId; }
38
39  Point
40  Vertex::point() const
41  { return mPoint; }
42
43  //============================== INQUIRY    ===================================
44  bool
45  Vertex::hasRestrictions() const
46  { return false; }
47
48  ////////////////////////////// PROTECTED  //////////////////////////////////
49
50  ////////////////////////////// PRIVATE    //////////////////////////////////
```

D.7.6 Topology.h

```
1   /**  A class holding the elements of the topology.
2    *
3    * #include "Topology.h"
4    *
5    * @author  Jonas Bergman
6    */
7
8   #ifndef GRAPH_TOPOLOGY_H_
9   #define GRAPH_TOPOLOGY_H_
10
11  // SYSTEM INCLUDES
```

```
12  //
13  #include <map>
14
15  // PROJECT INCLUDES
16  //
17  #include <boost/graph/adjacency_list.hpp>
18  #include <boost/graph/graph_traits.hpp>
19
20  // LOCAL INCLUDES
21  //
22  #include "Edge.h"
23  #include "TopologyException.h"
24  #include "Vertex.h"
25  #include "../util/Point.h"
26
27
28  // FORWARD REFERENCES
29  //
30
31  // TYPES
32  //
33  /** Mapping of topology vertex id and topology Vertex object. */
34  typedef std::map<VertexIdType, Vertex>          TopoVertexMapType;
35
36  /** Mapping of topology edge id and topology Edge object. */
37  typedef std::map<EdgeIdType, Edge>              TopoEdgeMapType;
38
39  /** Keep track of which topology edges that make up an edge in the original
40   * OSM map data, as the OSM edge might have been split into several edges when
41   * building the topology.
42   */
43  typedef std::multimap<OsmIdType, EdgeIdType>    OsmIdToTopoIdEdgeMap;
44
45  /** This class holds Edges and Vertices such as they are in the database.
46   */
47  class Topology
48  {
49      friend class GraphBuilder;
50
51  public:
52  // LIFECYCLE
53
54      /** Default constructor.
55       */
56      Topology();
57
58      /** Copy constructor.
59       *
60       * @param from The value to copy to this object.
61       */
62      Topology(const Topology& from) = delete;
63
64
65  // OPERATORS
66  // OPERATIONS
67
68      /** Try to add a vertex to the topology.
```

```
69          * If a vertex with the id already exists: return old value.
70          * @param   id       Id for the vertex
71          * @param   point    The position of the vertex
72          * @return  A reference to a vertex with given id
73          */
74         Vertex&         addVertex(VertexIdType id, Point point);
75
76
77         /** Try to add an edge to the topology.
78          * If an edge with the id already exists: return old value.
79          * @param   id          Id for the edge
80          * @param   osmId       The original OsmId this edge belongs to.
81          * @param   source      Id for source vertex
82          * @param   target      Id for target vertex
83          * @param   geomData    Geometric data for the edge
84          * @param   roadData    Road data for the edge
85          * @return  A reference to an edge with given id
86          * @throw   Topology Exception if vertices are not in topology.
87          */
88         Edge&           addEdge(EdgeIdType      id,
89                                 OsmIdType       osmId,
90                                 VertexIdType    source,
91                                 VertexIdType    target,
92                                 Edge::GeomData  geomData,
93                                 Edge::RoadData  roadData);
94
95         /** Try to add an edge to the topology.
96          * Using default values for geometric and road data.
97          * If an edge with the id already exists: return old value.
98          * @param   id      Id for the edge
99          * @param   osmId   The original OsmId this edge belongs to.
100         * @param   source  Id for source vertex
101         * @param   target  Id for target vertex
102         * @return  A reference to an edge with given id
103         * @throw   Topology Exception if vertices are not in topology.
104         */
105        Edge&           addEdge(EdgeIdType      id,
106                                OsmIdType       osmId,
107                                VertexIdType    source,
108                                VertexIdType    target);
109
110        /** Fetch the vertex with given id.
111         * @param   id        Id of the vertex to get
112         * @return  Reference to the found vertex
113         * @throws  TopologyException if vertex does not exist.
114         */
115        Vertex&         getVertex(VertexIdType id);
116        const Vertex&   getVertex(VertexIdType id) const;
117
118        /** Fetch the edge with given id.
119         * @param   id        Id of the edge to get
120         * @return  Reference to the found vertex
121         * @throws  TopologyException if vertex does not exist.
122         */
123        Edge&           getEdge(EdgeIdType id);
124        const Edge&     getEdge(EdgeIdType id) const;
125
```

```
126     /** Clear everything in the topology: edges and vertices.
127      */
128     void            clearTopology();
129 
130     /** Remove restrictions and costs on all edges in the topology.
131      */
132     void            clearEdgeCostAndRestrictions();
133 
134 // ACCESS
135     /**
136      * @return    the Number of vertices in topology.
137      */
138     size_t          nrVertices() const;
139 
140     /**
141      * @return    the Number of vertices in topology.
142      */
143     size_t          nrEdges() const;
144 
145 // INQUIRY
146 
147 protected:
148 private:
149 // ATTRIBUTES
150     TopoVertexMapType     mVertexMap;
151     TopoEdgeMapType       mEdgeMap;
152     OsmIdToTopoIdEdgeMap  mOsmEdgeMap;
153 };
154 
155 // INLINE METHODS
156 //
157 
158 // EXTERNAL REFERENCES
159 //
160 
161 #endif /* GRAPH_TOPOLOGY_H_ */
```

D.7.7 Topology.cc

```
1  /*
2   * Topology.cc
3   *
4   * @author  Jonas Bergman
5   */
6  
7  #include "Topology.h"  // class implemented
8  
9  #include <utility>
10 
11 ////////////////////////////// PUBLIC //////////////////////////////////////
12 
13 //============================= LIFECYCLE ====================================
14 Topology::Topology()
15     : mVertexMap(), mEdgeMap(), mOsmEdgeMap()
16 {
17 }
18 //============================= OPERATORS ====================================
```

```
19  //============================ OPERATIONS ================================
20
21  Vertex&
22  Topology::addVertex(VertexIdType id, Point point)
23  {
24      auto res = mVertexMap.emplace(id, Vertex(id, point));
25      return res.first->second;
26  }
27
28
29  Edge&
30  Topology::addEdge(EdgeIdType        id,
31                    OsmIdType         osmId,
32                    VertexIdType      source,
33                    VertexIdType      target,
34                    Edge::GeomData    geomData,
35                    Edge::RoadData    roadData)
36  {
37       try
38      {
39          getVertex(source);
40          getVertex(target);
41          Edge edge(id, osmId, source, target, geomData, roadData);
42          auto res = mEdgeMap.emplace(id, std::move(edge));
43          mOsmEdgeMap.insert({osmId, id});
44          return res.first->second;
45      }
46      catch (TopologyException& e)
47      {
48          throw TopologyException("Cannot add edge: " + std::to_string(id) +
49                  ". " + e.what());
50      }
51  }
52
53  Edge&
54  Topology::addEdge(EdgeIdType        id,
55                    OsmIdType         osmId,
56                    VertexIdType      source,
57                    VertexIdType      target)
58  {
59      Edge::GeomData gd;
60      Edge::RoadData rd;
61      return addEdge(id, osmId, source, target, gd, rd);
62  }
63
64  Vertex&
65  Topology::getVertex(VertexIdType id)
66  {
67      auto it = mVertexMap.find(id);
68      if(it == mVertexMap.end()) {
69          throw TopologyException("Vertex not found: " + std::to_string(id));
70      }
71      return it->second;
72  }
73
74  const Vertex&
75  Topology::getVertex(VertexIdType id) const
```

```
76  {
77      return const_cast<Topology&>(*this).getVertex(id);
78  }
79
80  Edge&
81  Topology::getEdge(EdgeIdType id)
82  {
83      auto it = mEdgeMap.find(id);
84      if(it == mEdgeMap.end()) {
85          throw TopologyException("Edge not found: " + std::to_string(id));
86      }
87      return it->second;
88  }
89
90  const Edge&
91  Topology::getEdge(EdgeIdType id) const
92  {
93      auto it = mEdgeMap.find(id);
94      if(it == mEdgeMap.end()) {
95          throw TopologyException("Edge not found: " + std::to_string(id));
96      }
97      return it->second;
98  }
99
100 void
101 Topology::clearTopology()
102 {
103     mVertexMap.clear();
104     mEdgeMap.clear();
105     mOsmEdgeMap.clear();
106 }
107
108 void
109 Topology::clearEdgeCostAndRestrictions()
110 {
111     for(auto& it : mEdgeMap)
112     {
113         it.second.clearCostsAndRestrictions();
114     }
115 }
116
117
118 //============================== ACESS      ==================================
119 size_t
120 Topology::nrVertices() const
121 {
122     return mVertexMap.size();
123 }
124
125 size_t
126 Topology::nrEdges() const
127 {
128     return mEdgeMap.size();
129 }
130 //============================== INQUIRY    ==================================
131 ////////////////////////////// PROTECTED  //////////////////////////////////
132
```

```
133 ////////////////////////////// PRIVATE    ///////////////////////////////////
```

D.7.8 GraphBuilder.h

```
1  /**  GraphBuilder.
2   *
3   * #include "GraphBuilder.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef GRAPH_GRAPHBUILDER_H_
9  #define GRAPH_GRAPHBUILDER_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <algorithm>
14 #include <map>
15 #include <ostream>
16
17 // PROJECT INCLUDES
18 //
19 #include <boost/graph/adjacency_list.hpp>
20
21 // LOCAL INCLUDES
22 //
23 #include "Edge.h"
24 #include "EdgeRestriction.h"
25 #include "GraphException.h"
26 #include "Topology.h"
27 #include "TurnCostCalculator.h"
28 #include "Vertex.h"
29 #include "../config/Configuration.h"
30 #include "../util/Logging.h"
31
32 // FORWARD REFERENCES
33 //
34
35 // TYPES
36 typedef EdgeIdType  NodeIdType;
37 typedef EdgeIdType  LineIdType;
38
39 /**
40  * Map the GraphEdges to the original Edge id in the Topology.
41  * Also indicate if the GraphEdge is the same or opposite direction to
42  * the graph in the topology.
43  */
44 struct GraphEdge
45 {
46     EdgeIdType      graphEdgeId;
47     EdgeIdType      topoEdgeId;
48     bool            oppositeDirection {false};
49 };
50
51 /**
52  * A Node in the LineGraph corresponds directly to an Edge in the original
53  * GraphBuilder and topology. It is connected to another Node (Edge) if both the
```

```
54   * edges are adjacent and there is no restriction in the Vertex for travel
55   * along them.
56   * lgNodeId === graphEdgeId
57   */
58  typedef GraphEdge   LineGraphNode;
59
60  /**
61   * Map the GraphVertices to the original Vertex id in the Topology.
62   */
63  struct GraphVertex
64  {
65      VertexIdType    graphVertexId;
66      VertexIdType    topoVertexId;
67  };
68
69  /**
70   * A LineGraphLine corresponds to a travel along an incoming edge,
71   * via a vertex and out an outgoing edge.
72   * The cost is the cost of travel on the incoming edge and the turn cost
73   * at the vertex.
74   * The Line connects two edges in the graph with an allowed turn in between.
75   */
76  struct LineGraphLine
77  {
78      NodeIdType      lgSourceNodeId;
79      NodeIdType      lgTargetNodeId;
80      VertexIdType    topoViaVertexId;
81      double          cost;
82  };
83
84  /** The 'normal' vertex based graph type. */
85  typedef boost::adjacency_list
86      < boost::listS, boost::vecS, boost::directedS,
87      GraphVertex, GraphEdge >                                    GraphType;
88  /** The edge based graph type. */
89  typedef boost::adjacency_list
90      < boost::listS, boost::vecS, boost::directedS,
91      LineGraphNode, LineGraphLine >                              LineGraphType;
92
93  /** A vertex in the normal graph. */
94  typedef boost::graph_traits<GraphType>::vertex_descriptor       VertexType;
95  /** An edge in the normal graph. */
96  typedef boost::graph_traits<GraphType>::edge_descriptor         EdgeType;
97
98  /** A node in the line graph. */
99  typedef boost::graph_traits<LineGraphType>::vertex_descriptor   NodeType;
100 /** An edge in the line graph. */
101 typedef boost::graph_traits<LineGraphType>::edge_descriptor     LineType;
102
103 /** Mapping of a topology vertex id and graph vertex object. */
104 typedef std::map<VertexIdType, VertexType>    TopoVertexIdToGraphVertexMapType;
105 /** Mapping of a topology edge id and graph edge object. */
106 typedef std::multimap<EdgeIdType, EdgeType>   TopoEdgeIdToGraphEdgeMapType;
107 /** Mapping of a graph edge id and linegraph node object. */
108 typedef std::map<EdgeIdType, NodeType>        GraphEdgeIdToNodeMapType;
109
110
```

```
111 /**
112  * A class for building (Boost) Graph and LineGraph from a Topology and
113  * a Configuration with optional Restrictions and Costs applied.
114  */
115 class GraphBuilder
116 {
117 public:
118 // LIFECYCLE
119     /** Constructor.
120      * Disabled.
121      */
122     GraphBuilder() = delete;
123 
124     /** Constructor.
125      * GraphBuilder should be based on the supplied topology.
126      * @param   rTopology       The topology to use as basis for the graph.
127      * @param   rConfig         The configuration used for topology and all.
128      * @param   useRestrictions If the graph should be built with restrictions or not.
129      */
130     GraphBuilder(
131         Topology& rTopology,
132         const Configuration& rConfig,
133         bool useRestrictions = true);
134 
135     /** Copy constructor.
136      * Disabled.
137      */
138     GraphBuilder(const GraphBuilder& from) = delete;
139 
140     /** Destructor.
141      */
142     ~GraphBuilder();
143 
144 // OPERATORS
145     /** Output operator to print to a stream.
146      */
147     friend
148     std::ostream&       operator<<(std::ostream& os, const GraphBuilder& rGraph);
149 
150 // OPERATIONS
151 // ACCESS
152     /**
153      * @return  The number of Vertices in the Graph.
154      */
155     size_t              nrVertices() const;
156 
157     /**
158      * @return  The number of Edges in the Graph.
159      */
160     size_t              nrEdges() const;
161 
162     /**
163      * @return  The number of Nodes in the LineGraph.
164      */
165     size_t              nrNodes() const;
166 
167     /**
```

```
168         * @return  The number of Nodes in the LineGraph.
169         */
170        size_t              nrLines() const;
171
172        /** Builds graph if necessary before returning.
173         * @return  The Boost Graph representation of the Graph.
174         * @throws  GraphException if something goes wrong building the graph.
175         */
176        const GraphType&    getBoostGraph();
177
178        /** Get a reference to the line graph.
179         * @return  The Boost Graph representation of the LineGraph.
180         * @throws  GraphException if something goes wrong building the graph.
181         */
182        LineGraphType&       getBoostLineGraph();
183        const LineGraphType& getBoostLineGraph() const;
184
185        /** Get access to the topology that is the base for the graph.
186         * @return  The Topology
187         */
188        const Topology&     getTopology() const;
189
190    // INQUIRY
191        /**
192         * @return  true    If graph has a vertex with given id.
193         */
194        bool                hasVertex(VertexIdType vertexId) const;
195
196        /**
197         * @return  true    If LineGraph has a node with given id.
198         */
199        bool                hasNode(EdgeIdType nodeId) const;
200
201        /** Get an already existing Node from the LineGraph.
202         * @param   id  The Edge id (== the Node id).
203         * @param   The LineGraph Node.
204         * @throw   GraphException if there is no Node with that id.
205         */
206        const NodeType&     getLineGraphNode(NodeIdType id) const;
207
208        /**
209         * @return  true    If graph was built with restrictions.
210         */
211        bool                isRestricted() const;
212
213        /** Output information about # vertices, edges, nodes, lines.
214         */
215        void                printGraphInformation(std::ostream& os) const;
216
217    protected:
218
219    private:
220    // HELPERS
221
222        // buildGraph() ------------------------------------------------------------
223        // Used when constructing the internal Boost graph representation
224        // from the Topology.
```

```
225
226     /** Build the graph by adding vertices and edges from the topology. */
227     void                buildGraph();
228
229     /** Add the topology vertices to the graph, respecting restrictions.
230      * Helper for 'buildGraph()'.
231      */
232     void                addTopoVerticesToGraph();
233
234     /** Add the topology edges to the graph, respecting restrictions.
235      * Helper for 'buildGraph()'.
236      */
237     void                addTopoEdgesToGraph();
238
239     /** Check if an edge is restricted
240      * @param   rEdge   Reference to edge
241      * @return  bool
242      */
243     bool                isEdgeRestricted(const Edge& rEdge) const;
244
245     /** Add the correct number of directed edges from the topo Edge.
246      * @param   rEdge        The topological graph data
247      * @param   rNewEdgeId   The running id for the graph's directed edges.
248      */
249     void                addDirectedGraphEdges(
250                             const Edge& rEdge,
251                             EdgeIdType& rNewEdgeId);
252
253     /** Add a directed edge from source to target.
254      * Helper for 'addTopoEdgesToGraph()'.
255      * @param   id      The edge's topology id.
256      * @param   source  The source vertex.
257      * @param   target  The target vertex.
258      * @param   e_ix    The running index amongst edges added to graph.
259      * @param   oppositeDirection
260      *                  If the directed edge runs opposite of the original
261      *                  edge direction as specified in the topology.
262      */
263     void                addDirectedEdge(
264                             EdgeIdType id,
265                             const VertexType& source,
266                             const VertexType& target,
267                             EdgeIdType ix,
268                             bool  oppositeDirection);
269
270     /** Get the graph vertex corresponding to a given id.
271      * @param   id      The vertex' topology id.
272      * @return  Reference to the Graph vertex corresponding to id.
273      * @throw   GraphException if there is no corresponding vertex to id.
274      */
275     const VertexType&   getGraphVertex(VertexIdType id) const;
276
277     // buidlLineGraph() ----------------------------------------------------
278     // Used when transforming the Graph to a LineGraph
279
280     /** Start converting the GraphBuilder to a LineGraph.
281      */
```

```
282     void                buildLineGraph();
283
284     /** Add Edges from the graph as Nodes in the Linegraph.
285      * Helper for 'buildLineGraph()'
286      */
287     void                addGraphEdgesToLineGraph();
288
289     /** Actually add a graph edge as a linegraph node, checking if it already
290      * exists or not.
291      * @param   rGraphEdge  The Edge to add to the LineGraph as Node.
292      * @param   rNode       The Node corresponding to the edge returned here.
293      */
294     void                addGraphEdgeAsLineGraphNode(
295                             const EdgeType& rGraphEdge,
296                             NodeType&       rNode);
297
298
299     /** Connect the newly added Node to all Nodes it should be connected to,
300      * that is look up which outgoing edges there are from the Edge's (node's)
301      * target vertex, and if there are no restrictions: add the Edge as a Node
302      * to the LineGraph and add a Line between the Nodes.
303      * @param   rSourceNode     The Node to add Lines from.
304      * @parma   rViaVertex      Are there any restrictions in the vertex?
305      * @throw   GraphException
306      */
307     void                connectSourceNodeToTargetNodesViaVertex(
308                             const NodeType& rSourceNode,
309                             const VertexType& rViaVertex);
310
311     /** Extract LineGraphNode data from the LineGraph.
312      * @param   rNode     The descriptor in the LineGraph
313      * @return  a LineGraphNode
314      */
315     LineGraphNode       getLineGraphNodeData(const NodeType& rNode) const;
316
317     /** Add a line in the LineGraph, connecting the source and target nodes.
318      * @param   rSourceNode
319      * @param   rTargetNode
320      * @return  the added line
321      * @throw   GraphException
322      */
323     LineType            addLineGraphLine(
324                             const NodeType& rSourceNode,
325                             const NodeType& rTargetNode);
326
327     /** Add meta data ids for source, target and vertex to the newly added Line.
328      * @param   rLine
329      * @param   sourceId
330      * @param   targetId
331      * @param   viaVertexId
332      */
333     void                addLineMetaIds(
334                             const LineType& rLine,
335                             EdgeIdType sourceId,
336                             EdgeIdType targetId,
337                             VertexIdType viaVertexId);
338
```

```
339     /** Add the meta information about the cost to the new line.
340      * @param   rLine
341      * @param   rSourceEdge     The Source Edge
342      * @param   targetId        The id of the target edge in topology
343      */
344     void                addLineMetaCost(
345                             const LineType& rLine,
346                             const Edge&     rSourceEdge,
347                             EdgeIdType      targetId);
348
349     /** Calculate the cost for making a turn from source edge to target.
350      * Helper to `connectSourceNodeToTargetNodesViaVertex()`.
351      * @param   sourceEdgeId    The edge (and node) id of the source.
352      * @param   targetEdgeId    The edge (and node) id of the target.
353      */
354     double              calculateTurnCost(
355                             EdgeIdType sourceEdgeId,
356                             EdgeIdType targetEdgeId) const;
357
358     /**
359      * @param   edgeId  Id to edge to look up.
360      * @return  true if this edge has no exits, meaning it is no use adding it.
361      */
362     bool                edgeHasNoExit(EdgeIdType edgeId);
363
364     /**
365      * @return A vector of all Edges going out from a vertex.
366      */
367     std::vector<EdgeIdType>
368                         getOutEdges(VertexIdType vertexId) const;
369
370     /**
371      * @param   rSourceNode     The LineGraph Node
372      * @return  A vector of all restricted edges from this Edge.
373      */
374     std::vector<EdgeIdType>
375                         getRestrictedTargets(
376                             const LineGraphNode& rSourceNode) const;
377
378     /** Look through the targets from a source to find which are restricted
379      * and add them to a collection of restricted.
380      * @param   rSsourceEdge        The source edge.
381      * @param   rTargets            Targets from that source.
382      * @param   rRestrictedTargets  A collection to build up.
383      */
384     void                findRestrictedTargets(
385                             const Edge&                     rSourceEdge,
386                             const std::vector<EdgeIdType>&  rTargets,
387                             std::vector<EdgeIdType>&        rRestrictedTargets) const;
388
389     /** Add the turning restricted targets to the other restricted targets.
390      * @param   rSourceEdge         The source edge.
391      * @param   rRestrictedTargets  The collection of restricted targets.
392      */
393     void                addTurningRestrictedTargets(
394                             const Edge&                 rSource,
395                             std::vector<EdgeIdType>&    rRestrictedTargets) const;
```

```
396
397     /**
398      * @return  true if this target edge has restricted access from the source.
399      */
400     bool                isTargetRestricted(
401                             const std::vector<EdgeIdType>&  rRestrictedTargets,
402                             EdgeIdType                      targetId) const;
403
404     void                printVertices(std::ostream& os) const;
405     void                printEdges(std::ostream& os)    const;
406     void                printNodes(std::ostream& os)    const;
407     void                printLines(std::ostream& os)    const;
408
409 // ATTRIBUTES
410     GraphType                           mGraph;
411     LineGraphType                       mLineGraph;
412     TopoVertexIdToGraphVertexMapType    mIdToVertexMap;     // map original id to GraphVertex
413     TopoEdgeIdToGraphEdgeMapType        mIdToEdgeMap;       // map original id to GraphEdge
414     GraphEdgeIdToNodeMapType            mEdgeIdToNodeMap;   // map GraphEdge.id to LineGraphNode
415     Topology&                           mrTopology;
416     const Configuration&                mrConfiguration;
417     mutable boost::log::sources::severity_logger
418         <boost::log::trivial::severity_level>
419                                         mLog;
420     bool                                mUseRestrictions;
421
422 // CONSTANTS
423 };
424
425 // INLINE METHODS
426 //
427
428 // EXTERNAL REFERENCES
429 //
430
431 #endif /* GRAPH_GRAPHBUILDER_H_ */
```

D.7.9 GraphBuilder.cc

```
1  /*
2   * GraphBuilder.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "GraphBuilder.h"  // class implemented
8
9  #include <typeinfo>
10
11 ////////////////////////////// PUBLIC //////////////////////////////////////
12
13 //============================= LIFECYCLE ====================================
14 GraphBuilder::GraphBuilder(
15     Topology& rTopology,
16     const Configuration& rConfig,
17     bool useRestrictions)
18     : mGraph(),
```

```
19          mLineGraph(),
20          mIdToVertexMap(),
21          mIdToEdgeMap(),
22          mrTopology(rTopology),
23          mrConfiguration(rConfig),
24          mLog(),
25          mUseRestrictions(useRestrictions)
26  {
27      Logging::initLogging();
28      boost::log::add_common_attributes();
29
30      buildGraph();
31      buildLineGraph();
32  }
33
34  GraphBuilder::~GraphBuilder()
35  {
36  }
37
38  //============================== OPERATORS ===================================
39  std::ostream&
40  operator<<(std::ostream& os, const GraphBuilder& rGraph)
41  {
42      rGraph.printGraphInformation(os);
43
44      os << std::endl << "Vertices: " << std::endl;
45      rGraph.printVertices(os);
46
47      os << std::endl << "Edges: " << std::endl;
48      rGraph.printEdges(os);
49
50      os << std::endl << "Nodes: " << std::endl;
51      rGraph.printNodes(os);
52
53      os << std::endl << "Lines: " << std::endl;
54      rGraph.printLines(os);
55
56      return os;
57  }
58  //============================== OPERATIONS ==================================
59  //============================== ACESS       ==================================
60  size_t
61  GraphBuilder::nrVertices() const
62  {
63      return mIdToVertexMap.size();
64  }
65
66  size_t
67  GraphBuilder::nrEdges() const
68  {
69      return mIdToEdgeMap.size();
70  }
71
72  size_t
73  GraphBuilder::nrNodes() const
74  {
75      return boost::num_vertices(mLineGraph);
```

```
76  }
77
78  size_t
79  GraphBuilder::nrLines() const
80  {
81      return boost::num_edges(mLineGraph);
82  }
83
84  const GraphType&
85  GraphBuilder::getBoostGraph()
86  {
87      return mGraph;
88  }
89
90  LineGraphType&
91  GraphBuilder::getBoostLineGraph()
92  {
93      return mLineGraph;
94  }
95
96  const LineGraphType&
97  GraphBuilder::getBoostLineGraph() const
98  {
99      return mLineGraph;
100 }
101
102 const Topology&
103 GraphBuilder::getTopology() const
104 {
105     return mrTopology;
106 }
107
108 //============================== INQUIRY    ===================================
109 bool
110 GraphBuilder::hasVertex(VertexIdType vertexId) const
111 {
112     const auto& it = mIdToVertexMap.find(vertexId);
113     return (it != mIdToVertexMap.end());
114 }
115
116 bool
117 GraphBuilder::hasNode(EdgeIdType nodeId) const
118 {
119     const auto& it = mEdgeIdToNodeMap.find(nodeId);
120     return (it != mEdgeIdToNodeMap.end());
121 }
122
123 const NodeType&
124 GraphBuilder::getLineGraphNode(NodeIdType id) const
125 {
126     const auto& res = mEdgeIdToNodeMap.find(id);
127     if(res == mEdgeIdToNodeMap.end())
128     {
129         throw GraphException("Graph:getLineGraphNode: Missing node: "
130             + std::to_string(id));
131     }
132     return res->second;
```

```
133 }
134
135 bool
136 GraphBuilder::isRestricted() const
137 {
138     return mUseRestrictions;
139 }
140
141 ////////////////////////////// PROTECTED  ///////////////////////////////////
142
143 ////////////////////////////// PRIVATE    ///////////////////////////////////
144 void
145 GraphBuilder::buildGraph()
146 {
147     addTopoVerticesToGraph();
148     addTopoEdgesToGraph();
149 }
150
151 void
152 GraphBuilder::addTopoVerticesToGraph()
153 {
154     VertexIdType v_ix = 0;
155     for(const auto& vertexpair : mrTopology.mVertexMap)
156     {
157         VertexType v = boost::add_vertex(mGraph);
158         mIdToVertexMap.insert({vertexpair.second.id(), v});
159         mGraph[v].graphVertexId = v_ix;
160         mGraph[v].topoVertexId = vertexpair.second.id();
161         ++v_ix;
162     }
163 }
164
165 void
166 GraphBuilder::addTopoEdgesToGraph()
167 {
168     EdgeIdType e_ix = 0;
169
170     for(const auto& edgepair : mrTopology.mEdgeMap)
171     {
172         const Edge& e = edgepair.second;
173
174         if(isEdgeRestricted(e))
175         {
176             continue;
177         }
178
179         addDirectedGraphEdges(e, e_ix);
180     }
181 }
182
183
184 bool
185 GraphBuilder::isEdgeRestricted(const Edge& rEdge) const
186 {
187     if(mUseRestrictions && rEdge.isRestricted(mrConfiguration))
188     {
189         BOOST_LOG_SEV(mLog, boost::log::trivial::info)
```

```
190                             << "Graph:addTopoEdgeToGraph(): "
191                             << "Restricted Edge id " << rEdge.id();
192         return true;
193     }
194     return false;
195 }
196
197 void
198 GraphBuilder::addDirectedGraphEdges(const Edge& rEdge, EdgeIdType& rNewEdgeId)
199 {
200     const VertexType& s = getGraphVertex(rEdge.sourceId());
201     const VertexType& t = getGraphVertex(rEdge.targetId());
202
203     // add all lanes in forward direction
204     if(rEdge.roadData().direction == Edge::DirectionType::FROM_TO
205         || rEdge.roadData().direction == Edge::DirectionType::BOTH)
206     {
207         for(size_t lane = 1; lane <= rEdge.roadData().nrLanes; ++lane) {
208             addDirectedEdge(rEdge.id(), s, t, rNewEdgeId, false);
209             ++rNewEdgeId;
210         }
211     }
212
213     // add all lanes in backward direction
214     if(rEdge.roadData().direction == Edge::DirectionType::TO_FROM
215         || rEdge.roadData().direction == Edge::DirectionType::BOTH)
216     {
217         for(size_t lane = 1; lane <= rEdge.roadData().nrLanes; ++lane) {
218             addDirectedEdge(rEdge.id(), t, s, rNewEdgeId, true);
219             ++rNewEdgeId;
220         }
221     }
222 }
223
224 void
225 GraphBuilder::addDirectedEdge(
226     EdgeIdType          id,
227     const VertexType&   source,
228     const VertexType&   target,
229     EdgeIdType          e_ix,
230     bool                oppositeDirection)
231 {
232     const auto& res = boost::add_edge(source, target, mGraph);
233     if(res.second == true)
234     {
235         mIdToEdgeMap.insert({id, res.first});
236         mGraph[res.first].graphEdgeId = e_ix;
237         mGraph[res.first].topoEdgeId = id;
238         mGraph[res.first].oppositeDirection = oppositeDirection;
239     }
240     else
241     {
242         throw GraphException("Graph:addDirectedEdge: cannot add edge: "
243             + std::to_string(id));
244     }
245 }
246
```

```
247 const VertexType&
248 GraphBuilder::getGraphVertex(VertexIdType id) const
249 {
250     const auto& res = mIdToVertexMap.find(id);
251     if(res == mIdToVertexMap.end())
252     {
253         throw GraphException("Graph:getGraphVertex: Missing vertex: "
254             + std::to_string(id));
255     }
256     return res->second;
257 }
258
259 void
260 GraphBuilder::buildLineGraph()
261 {
262     mLineGraph.clear();
263     addGraphEdgesToLineGraph();
264 }
265
266 void
267 GraphBuilder::addGraphEdgesToLineGraph()
268 {
269     // iterate through edges: add as Node.
270     for(auto e_it = boost::edges(mGraph);
271         e_it.first != e_it.second;
272         ++e_it.first)
273     {
274         const EdgeType& edge = *(e_it.first);
275
276         NodeType node;
277         addGraphEdgeAsLineGraphNode(edge, node);
278
279         // look up targetId vertex.
280         VertexType via_vertex = boost::target(edge, mGraph);
281
282         // connect all possible travels from 'edge' via the vertex
283         connectSourceNodeToTargetNodesViaVertex(node, via_vertex);
284     }
285 }
286
287 void
288 GraphBuilder::addGraphEdgeAsLineGraphNode(const EdgeType& rGraphEdge,
289                                           NodeType& rNode)
290 {
291     EdgeIdType e_graph_id =
292         boost::get(&GraphEdge::graphEdgeId, mGraph, rGraphEdge);
293     EdgeIdType e_topo_id  =
294         boost::get(&GraphEdge::topoEdgeId, mGraph, rGraphEdge);
295
296     if(!hasNode(e_graph_id))
297     {
298         rNode = boost::add_vertex(mLineGraph);
299         mLineGraph[rNode].graphEdgeId = e_graph_id;
300         mLineGraph[rNode].topoEdgeId = e_topo_id;
301         mEdgeIdToNodeMap.insert({e_graph_id, rNode});
302     }
303     else
```

```
304     {
305         rNode = getLineGraphNode(e_graph_id);
306     }
307 }
308 
309 
310 void
311 GraphBuilder::connectSourceNodeToTargetNodesViaVertex(
312     const NodeType& rSourceNode,
313     const VertexType& rViaVertex)
314 {
315     // SOURCE
316     LineGraphNode source_node = getLineGraphNodeData(rSourceNode);
317 
318     if(edgeHasNoExit(source_node.topoEdgeId))
319     {
320         return;
321     }
322 
323     // get the edge corresponding to the node
324     Edge& source_edge = mrTopology.getEdge(source_node.topoEdgeId);
325 
326     // VIA
327     VertexIdType via_topo_vertex_id =
328         boost::get(&GraphVertex::topoVertexId, mGraph, rViaVertex);
329 
330     // TARGET
331     // get targets that are restricted
332     std::vector<EdgeIdType> restricted_targets =
333         getRestrictedTargets(source_node);
334 
335     // look at all out edges from the via-vertex
336     for(auto target_it = boost::out_edges(rViaVertex, mGraph);
337         target_it.first != target_it.second;
338         ++target_it.first)
339     {
340         const EdgeType& target = *(target_it.first);
341         EdgeIdType target_topo_id =
342             boost::get(&GraphEdge::topoEdgeId, mGraph, target);
343 
344         if(!isTargetRestricted(restricted_targets, target_topo_id))
345         {
346             // add nodes to LineGraph
347             NodeType target_node;
348             addGraphEdgeAsLineGraphNode(target, target_node);
349 
350             NodeIdType target_edge_id =
351                 boost::get(&LineGraphNode::graphEdgeId, mLineGraph, target_node);
352 
353             // add Line between Nodes to the LineGraph
354             try
355             {
356                 LineType line = addLineGraphLine(rSourceNode, target_node);
357                 addLineMetaIds(
358                     line,
359                     source_node.graphEdgeId,
360                     target_edge_id,
```

```
361                     via_topo_vertex_id);
362                 addLineMetaCost(line, source_edge, target_topo_id);
363             }
364             catch (GraphException& ge)
365             {
366                 throw GraphException(
367                     "Graph:connectSourceNodeToTargetNodesViaVertex: source: "
368                     + std::to_string(source_node.graphEdgeId)
369                     + ", target: " + std::to_string(target_edge_id));
370             }
371         }
372         else // log restricted targetId
373         {
374             Edge& s = mrTopology.getEdge(source_node.topoEdgeId);
375             Edge& t = mrTopology.getEdge(target_topo_id);
376             BOOST_LOG_SEV(mLog, boost::log::trivial::info)
377             << "Graph:connectSourceNodeToTargetNodesViaVertex(): Restricted: "
378             << "Source: " << source_node.topoEdgeId << " (osm: " << s.osmId()
379             << ") , Target: " << target_topo_id << " (osm: " << t.osmId() << ")";
380         }
381     }
382 }
383
384 LineGraphNode
385 GraphBuilder::getLineGraphNodeData(const NodeType& rNode) const
386 {
387     LineGraphNode node;
388     node.topoEdgeId = boost::get(&LineGraphNode::topoEdgeId, mLineGraph, rNode);
389     node.graphEdgeId = boost::get(&LineGraphNode::graphEdgeId, mLineGraph, rNode);
390     node.oppositeDirection =
391         boost::get(&LineGraphNode::oppositeDirection, mLineGraph, rNode);
392
393     return node;
394 }
395
396 LineType
397 GraphBuilder::addLineGraphLine(const NodeType& rSourceNode,
398                                const NodeType& rTargetNode)
399 {
400     const auto& line_add =
401         boost::add_edge(rSourceNode, rTargetNode, mLineGraph);
402     if(line_add.second == true)
403     {
404         return line_add.first;
405     }
406     else // could not add the line to the linegraph
407     {
408         throw GraphException("GraphBuilder:addLineGraphLine");
409     }
410 }
411
412 void
413 GraphBuilder::addLineMetaIds(
414     const LineType& rLine,
415     EdgeIdType      sourceId,
416     EdgeIdType      targetId,
417     VertexIdType    viaVertexId)
```

```
418 {
419     mLineGraph[rLine].lgSourceNodeId = sourceId;
420     mLineGraph[rLine].lgTargetNodeId = targetId;
421     mLineGraph[rLine].topoViaVertexId = viaVertexId;
422 }
423
424 void
425 GraphBuilder::addLineMetaCost(
426     const LineType& rLine,
427     const Edge&     rSourceEdge,
428     EdgeIdType      targetId)
429 {
430     mLineGraph[rLine].cost =
431         rSourceEdge.cost() +
432         calculateTurnCost(rSourceEdge.id(), targetId);
433 }
434
435 double
436 GraphBuilder::calculateTurnCost(EdgeIdType sourceEdgeId,
437                                 EdgeIdType targetEdgeId) const
438 {
439     const Edge& source = mrTopology.getEdge(sourceEdgeId);
440     const Edge& target = mrTopology.getEdge(targetEdgeId);
441     return TurnCostCalculator::getTurnCost(source, target, mrConfiguration);
442 }
443
444 bool
445 GraphBuilder::edgeHasNoExit(EdgeIdType edgeId)
446 {
447     Edge& e = mrTopology.getEdge(edgeId);
448     if(e.hasRestrictions() && e.restrictions().hasNoExitRestriction())
449     {
450         return true;
451     }
452     return false;
453 }
454
455 std::vector<EdgeIdType>
456 GraphBuilder::getOutEdges(VertexIdType vertexId) const
457 {
458     std::vector<EdgeIdType> out_edges;
459     VertexType graphVertex = getGraphVertex(vertexId);
460     auto edge_iterators = boost::out_edges(graphVertex, mGraph);
461     while(edge_iterators.first != edge_iterators.second) {
462         const EdgeType& e = *(edge_iterators.first);
463         EdgeIdType edgeId = boost::get(&GraphEdge::topoEdgeId, mGraph, e);
464         out_edges.push_back(edgeId);
465         ++edge_iterators.first;
466     }
467     return out_edges;
468 }
469
470 std::vector<EdgeIdType>
471 GraphBuilder::getRestrictedTargets(const LineGraphNode& rSourceNode) const
472 {
473     std::vector<EdgeIdType> restricted_targets;
474
```

```
475     // Find all out edges from the targetId vertex of the edge,
476     // which depends on if the edge is the opposite direction of the topo edge.
477     Edge& sourceEdge = mrTopology.getEdge(rSourceNode.topoEdgeId);
478
479     VertexIdType target_vertex =
480         rSourceNode.oppositeDirection ?
481             sourceEdge.sourceId() : sourceEdge.targetId();
482
483     std::vector<EdgeIdType> out_edges = getOutEdges(target_vertex);
484     std::vector<EdgeIdType> targets;
485     targets.insert(targets.end(), out_edges.begin(), out_edges.end());
486
487     // build map of restricted targets
488     findRestrictedTargets(sourceEdge, targets, restricted_targets);
489
490     return restricted_targets;
491 }
492
493 void
494 GraphBuilder::findRestrictedTargets(
495     const Edge&                     rSourceEdge,
496     const std::vector<EdgeIdType>&  rTargets,
497     std::vector<EdgeIdType>&        rRestrictedTargets) const
498 {
499     for(EdgeIdType e_id : rTargets)
500     {
501         // don't add self to targetId
502         if(e_id == rSourceEdge.id())
503         {
504             continue;
505         }
506
507         Edge& e = mrTopology.getEdge(e_id);
508
509         if(e.isRestricted(mrConfiguration))
510         {
511             BOOST_LOG_SEV(mLog, boost::log::trivial::info)
512                         << "Graph:getRestrictedTargets(): "
513                         << "Source id " << rSourceEdge.id()
514                         << " has restricted target: " << e_id;
515             rRestrictedTargets.push_back(e_id);
516         }
517     }
518
519     addTurningRestrictedTargets(rSourceEdge, rRestrictedTargets);
520 }
521
522 void
523 GraphBuilder::addTurningRestrictedTargets(
524     const Edge& rSourceEdge,
525     std::vector<EdgeIdType>& rRestrictedTargets) const
526 {
527     if(rSourceEdge.hasRestrictions() &&
528         rSourceEdge.restrictions().hasTurningRestriction())
529     {
530         BOOST_LOG_SEV(mLog, boost::log::trivial::info)
531                     << "Graph:getRestrictedTargets(): "
```

```
532                     << "Source id " << rSourceEdge.id()
533                     << " has TURN restricted targets. ";
534         std::vector<EdgeIdType> turn_restricted_targets =
535             rSourceEdge.restrictions().restrictedTargetEdges();
536         rRestrictedTargets.insert(rRestrictedTargets.end(),
537             turn_restricted_targets.begin(), turn_restricted_targets.end());
538     }
539 }
540
541 bool
542 GraphBuilder::isTargetRestricted(
543     const std::vector<EdgeIdType>& rRestrictedTargets,
544     EdgeIdType targetId) const
545 {
546     if(mUseRestrictions && rRestrictedTargets.size() > 0)
547     {
548         const auto& restr_it = std::find(
549             rRestrictedTargets.begin(),
550             rRestrictedTargets.end(),
551             targetId);
552         if(restr_it != rRestrictedTargets.end())
553         {
554             BOOST_LOG_SEV(mLog, boost::log::trivial::info)
555                 << "Graph:isTargetRestricted(): "
556                 << "Restricted target id " << targetId;
557             return true;
558         }
559     }
560     return false;
561 }
562
563 void
564 GraphBuilder::printGraphInformation(std::ostream& os) const
565 {
566     os << "Graph: #vertices: " << nrVertices()
567        << ", #edges: " << nrEdges()
568        << ". LineGraph: #nodes: " << nrNodes()
569        << ", #lines: " << nrLines()
570        << std::endl;
571 }
572
573 void
574 GraphBuilder::printVertices(std::ostream& os) const
575 {
576     for(auto v_it = boost::vertices(mGraph);
577         v_it.first != v_it.second;
578         ++v_it.first)
579     {
580         const VertexType& v = *v_it.first;
581         VertexIdType graph_vertex_id =
582             boost::get(&GraphVertex::graphVertexId, mGraph, v);
583         VertexIdType topo_vertex_id  =
584             boost::get(&GraphVertex::topoVertexId, mGraph, v);
585         const Vertex& vertex = mrTopology.getVertex(topo_vertex_id);
586
587         os << "   graph_vertex_id: " << graph_vertex_id
588            << ", topo_vertex_id: " << topo_vertex_id
```

```
589             << "\n      v: " << v
590             << "  " << vertex << std::endl;
591     }
592 }
593
594 void
595 GraphBuilder::printEdges(std::ostream& os) const
596 {
597     for(auto e_it = boost::edges(mGraph);
598         e_it.first != e_it.second;
599         ++e_it.first)
600     {
601         const EdgeType& e = *(e_it.first);
602         EdgeIdType graph_edge_id =
603             boost::get(&GraphEdge::graphEdgeId, mGraph, e);
604         EdgeIdType topo_edge_id  =
605             boost::get(&GraphEdge::topoEdgeId, mGraph, e);
606         const Edge& edge = mrTopology.getEdge(topo_edge_id);
607
608         os << "   graph_edge_id: " << graph_edge_id
609            << ", e_topo_id: " << topo_edge_id
610            << "\n      e: " << e
611            << "  " << edge << std::endl;
612     }
613 }
614
615 void
616 GraphBuilder::printNodes(std::ostream& os) const
617 {
618     for(auto n_it = boost::vertices(mLineGraph);
619         n_it.first != n_it.second;
620         ++n_it.first)
621     {
622         const NodeType& node = *(n_it.first);
623         NodeIdType lg_node_id =
624             boost::get(&LineGraphNode::graphEdgeId, mLineGraph, node);
625         EdgeIdType topo_edge_id  =
626             boost::get(&LineGraphNode::graphEdgeId, mLineGraph, node);
627
628         os << "   lg_node_id (graph_edge_id): " << lg_node_id
629            << ", topo_edge_id: " << topo_edge_id << std::endl;
630     }
631 }
632
633 void
634 GraphBuilder::printLines(std::ostream& os) const
635 {
636     for(auto line_it = boost::edges(mLineGraph);
637         line_it.first != line_it.second;
638         ++line_it.first)
639     {
640         const LineType& line = *(line_it.first);
641         NodeIdType lg_source_id =
642             boost::get(&LineGraphLine::lgSourceNodeId, mLineGraph, line);
643         NodeIdType lg_target_id =
644             boost::get(&LineGraphLine::lgTargetNodeId, mLineGraph, line);
645         VertexIdType topo_via_vertex_id =
```

```
646             boost::get(&LineGraphLine::topoViaVertexId, mLineGraph, line);
647
648         os << "   lg_source_id: " << lg_source_id
649            << ", lg_target_id: " << lg_target_id
650            << ", topo_via_vertex_id: " << topo_via_vertex_id << std::endl;
651     }
652 }
```

D.7.10 Cost.h

```
1  /* The Cost type.
2   *
3   * Cost.h
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef GRAPH_COST_H_
9  #define GRAPH_COST_H_
10
11 typedef double Cost;
12
13 #endif /* GRAPH_COST_H_ */
```

D.7.11 Speed.h

```
1  /** The Speed type.
2   * Speed.h
3   *
4   * @author  Jonas Bergman
5   */
6
7  #ifndef GRAPH_SPEED_H_
8  #define GRAPH_SPEED_H_
9
10
11 typedef unsigned Speed;
12
13
14 #endif /* GRAPH_SPEED_H_ */
```

D.7.12 EdgeCost.h

```
1  /**  The Costs for an Edge.
2   * The cost or weight can be thought of as seconds, with the time to travel
3   * the edge as a base, and different obstacles as additional costs.
4   *
5   * #include "EdgeCost.h"
6   *
7   * @author  Jonas Bergman
8   */
9  #ifndef GRAPH_EDGECOST_H_
10 #define GRAPH_EDGECOST_H_
11
12 // SYSTEM INCLUDES
13 //
14 #include <map>
```

```
15
16  // PROJECT INCLUDES
17  //
18
19  // LOCAL INCLUDES
20  //
21  #include "Cost.h"
22
23  // FORWARD REFERENCES
24  //
25  //typedef double Cost;
26
27  /** Costs for edges:
28   * - speed (either set explicitly or default  from road category)
29   * - barriers (should be imported from the EdgeRestriction)
30   * - incline (not implemented yet)
31   * - surface
32   * - traffic_calming
33   * - highway => bus_stop
34   * - highway => crossing
35   * - highway => give_way
36   * - highway => mini_roundabout
37   * - highway => stop
38   * - highway => traffic_signals
39   * - public_transport => stop_position
40   * - railway => level_crossing
41   */
42  class EdgeCost
43  {
44  public:
45  // TYPES
46      enum CostType
47      {
48          TRAVEL_TIME,
49          BARRIER,
50          OTHER
51      };
52
53  // LIFECYCLE
54
55      /** Default constructor.
56       */
57      EdgeCost() = default;
58
59
60      /** Copy constructor.
61       *
62       * @param from The value to copy to this object.
63       */
64      EdgeCost(const EdgeCost& from) = delete;
65
66
67      /** Destructor.
68       */
69      ~EdgeCost() = default;
70
71
```

```
72  // OPERATORS
73
74      /** Accumulate a cost of a certain type, except `travel` which can not
75       * be accumulated.
76       * @param   type    The type of cost
77       * @param   cost    The value of the cost in seconds.
78       */
79      void    addCost(CostType type, Cost cost);
80
81      /** Clear out all costs.
82       */
83      void    clearCosts();
84
85  // OPERATIONS
86  // ACCESS
87
88      /**
89       * @return  The sum of all costs
90       */
91      Cost    getCost() const;
92
93      /**
94       * @return  The accumulated costs of a CostType
95       */
96      Cost    getCost(CostType type) const;
97
98  // INQUIRY
99      /** Find out if there are costs of a certain type
100      * @return  True if there is such a cost
101      */
102     bool    hasCost(CostType type) const;
103
104 protected:
105 private:
106     std::map<CostType, Cost> costs;
107 };
108
109 // INLINE METHODS
110 //
111
112 // EXTERNAL REFERENCES
113 //
114
115 #endif /* GRAPH_EDGECOST_H_ */
```

D.7.13 EdgeCost.cc

```
1   /*
2    * EdgeCost.cc
3    *
4    * @author  Jonas Bergman
5    */
6
7   #include "EdgeCost.h"  // class implemented
8
9   ////////////////////////////// PUBLIC //////////////////////////////////////
10  //============================= LIFECYCLE ====================================
```

```
11  //============================== OPERATORS ===================================
12  void
13  EdgeCost::addCost(CostType type, Cost cost)
14  {
15      if(hasCost(type))
16      {
17          if(type != EdgeCost::TRAVEL_TIME)
18          {
19              cost += getCost(type);
20          }
21          costs.erase(type);
22      }
23      costs.insert({type, cost});
24  }
25
26  void
27  EdgeCost::clearCosts()
28  {
29      costs.clear();
30  }
31  //============================== OPERATIONS ==================================
32  //============================== ACESS      ==================================
33  Cost
34  EdgeCost::getCost() const
35  {
36      Cost sum {0};
37      for(const auto& pair : costs)
38      {
39          sum += pair.second;
40      }
41      return sum;
42  }
43
44  Cost
45  EdgeCost::getCost(EdgeCost::CostType type) const
46  {
47      const auto& it = costs.find(type);
48      if(it != costs.end())
49      {
50          return it->second;
51      }
52      return 0;
53  }
54
55  //============================== INQUIRY    ==================================
56  bool
57  EdgeCost::hasCost(EdgeCost::CostType type) const
58  {
59      return costs.find(type) != costs.end();
60  }
61  ////////////////////////////// PROTECTED  //////////////////////////////////
62  ////////////////////////////// PRIVATE    //////////////////////////////////
```

D.7.14 EdgeRestriction.h

```
1  /**  The EdgeRestriction class contains different restrictions for edges
2   * in the graph such as dimensions, access, turn restrictions.
```

```
 *
 * #include "EdgeRestriction.h"
 *
 * @author  Jonas Bergman
 */

#ifndef GRAPH_EDGERESTRICTION_H_
#define GRAPH_EDGERESTRICTION_H_

// SYSTEM INCLUDES
//
#include <limits>
#include <map>
#include <set>

// PROJECT INCLUDES
//

// LOCAL INCLUDES
//
#include "Edge.h"
#include "RestrictionsException.h"
#include "Speed.h"
#include "Vertex.h"
#include "../config/Configuration.h"
#include "../osm/OsmAccess.h"
#include "../osm/OsmBarrier.h"
#include "../osm/OsmTurningRestriction.h"
#include "../osm/OsmVehicle.h"

// FORWARD REFERENCES
//
//class OsmTurningRestriction;

/**
 * EdgeRestriction are:
 * - vehicle properties
 * - General access to an edge
 * - vehicle type specific access
 * - barriers
 * - turn restrictions
 * - disused roads
 * - no-exit roads
 */
class EdgeRestriction
{
public:
// TYPES
    /** EdgeRestrictions on Vehicles to travel an Edge.
     *  Dimensions in meters.
     *  Speed in km/h
     */
    struct VehicleProperties
    {
        static double DEFAULT_DIMENSION_MAX;
        static Speed  DEFAULT_SPEED_MAX;
        static Speed  DEFAULT_SPEED_MIN;
```

```
60
61          double      maxHeight   {DEFAULT_DIMENSION_MAX};
62          double      maxLength   {DEFAULT_DIMENSION_MAX};
63          double      maxWeight   {DEFAULT_DIMENSION_MAX};
64          double      maxWidth    {DEFAULT_DIMENSION_MAX};
65
66          Speed       maxSpeed    {DEFAULT_SPEED_MAX};
67          Speed       minSpeed    {DEFAULT_SPEED_MIN};
68
69          /** Look if the vehicle properties restricts
70           * vehicle with given configuration.
71           * @return  True if these vehicle properties restricts access.
72           */
73          bool    restrictsAccess(const VehicleConfig& rVehicleConfig) const
74          {
75              return (maxHeight <= rVehicleConfig.height)
76                  || (maxLength <= rVehicleConfig.length)
77                  || (maxWeight <= rVehicleConfig.weight)
78                  || (maxWidth  <= rVehicleConfig.width)
79                  || (minSpeed  >= rVehicleConfig.maxspeed);
80          }
81      };
82
83      /** Types of restrictions.
84       */
85      enum RestrictionType
86      {
87          VEHICLE_PROPERTIES,
88          GENERAL_ACCESS,
89          VEHICLE_TYPE_ACCESS,
90          BARRIER,
91          TURNING,
92          DISUSED,
93          NO_EXIT,
94
95          NR_RESTRICTION_TYPES
96      };
97
98  // LIFECYCLE
99
100     /** Default constructor.
101      */
102     EdgeRestriction() = default;
103
104
105     /** Copy constructor.
106      *
107      * @param from The value to copy to this object.
108      */
109     EdgeRestriction(const EdgeRestriction& from) = delete;
110
111
112     /** Destructor.
113      */
114     ~EdgeRestriction();
115
116
```

```
117 // OPERATORS
118 // OPERATIONS
119
120     /** Check if this Restriction restricts when the Configuration is applied.
121      * @param   rConfig     Configuration
122      * @throw   RestrictionsException
123      */
124     bool                restricts(const Configuration& rConfig) const;
125
126     /** Set vehicle properties for the specified edge.
127      * Replacing any existing properties with the new ones.
128      * @param   pVehicleProperties  The properties to install for the edge.
129      */
130     void                setVehiclePropertyRestriction(
131                             VehicleProperties* pVehicleProperties);
132
133     /** Set access restrictions for this edge, that is restrictions for all.
134      * @param   pGeneralAccess  The access to set.
135      */
136     void                setGeneralAccessRestriction(
137                             OsmAccess*  pGeneralAccess);
138
139     /** Set access restrictions for this edge, that is restrictions for all.
140      * @param   generalAccessType  The access type to set.
141      */
142     void                setGeneralAccessRestriction(
143                             OsmAccess::AccessType  generalAccessType);
144
145     /** Set access restrictions for edge based on vehicle type.
146      * There can be several vehicle restrictions for each edge.
147      * @param   vehicleType The type of vehicle to restrict on the edge.
148      * @param   pAccess     The access restriction for that vehicle type
149      *                      on this edge.
150      */
151     void                addVehicleTypeAccessRestriction(
152                             OsmVehicle::VehicleType  vehicleType,
153                             OsmAccess*               pAccess);
154
155     /** Set access restrictions for edge based on vehicle type.
156      * There can be several vehicle restrictions for each edge.
157      * @param   vehicleType The type of vehicle to restrict on the edge.
158      * @param   accessType  The access restriction for that vehicle type
159      *                      on this edge.
160      */
161     void                addVehicleTypeAccessRestriction(
162                             OsmVehicle::VehicleType  vehicleType,
163                             OsmAccess::AccessType    accessType);
164
165     /** Set barrier restricting this edge.
166      * @param   pBarrier        The barrier to set.
167      */
168     void                setBarrierRestriction(
169                             OsmBarrier* pBarrier);
170
171     /** Set barrier restricting this edge.
172      * @param   barrierType    The barrier type to set.
173      */
```

```
174     void                setBarrierRestriction(
175                             OsmBarrier::BarrierType barrierType);
176 
177     /** Add turning restrictions from this edge.
178      * Actually just adds the restriction without checking if there already is
179      * a restriction between those two edges.
180      * @param   pTurningRestriction  The turning restriction to set.
181      */
182     void                addTurningRestriction(
183                             OsmTurningRestriction* pTurningRestriction);
184 
185     /** Set disused flag on this edge.
186      */
187     void                setDisusedRestriction();
188 
189     /** Set no exit flag on this edge.
190      */
191     void                setNoExitRestriction();
192 
193     /** Flag this edge as part of a via way restriction that needs attention
194      * when routing.
195      */
196     void                setViaWayRestriction();
197 
198 // ACCESS
199     /** Get which kinds of restrictions this edge has.
200      * @return  A vector with all types of restrictions.
201      */
202     std::vector<RestrictionType>
203                         restrictionTypes() const;
204 
205     /** Try to fetch the vehicle property restrictions for an Edge.
206      * @return  The Vehicle properties
207      * @throw   RestrictionsException if no entry exists for Edge.
208      */
209     const VehicleProperties&
210                         vehicleProperties() const;
211 
212     /** Try to fetch the vehicle property restrictions for an Edge.
213      * @return  The Vehicle properties
214      * @throw   RestrictionsException if no entry exists for Edge.
215      */
216     VehicleProperties&  vehicleProperties();
217 
218     /** Fetch the max speed for this edge. If no explicit speed is set it
219      * returns `VehicleProperties::DEFAULT_SPEED_MAX`. One can query to see if
220      * if there exists an explicit limit with  `hasMaxSpeedRestriction()`
221      * @return  Either the explicit speed limit or a default if not set.
222      */
223     Speed               maxSpeed() const;
224 
225     /** Try to fetch the general access restrictions for this edge.
226      * @param   edgeId  The id of the edge.
227      * @return  reference to the OsmAccess object.
228      * @throw   RestrictionsException if no entry exists for Edge.
229      */
230     const OsmAccess&    generalAccess() const;
```

```
231
232     /** Try to fetch the general access restrictions for this edge.
233      * @param   edgeId  The id of the edge.
234      * @return  reference to the OsmAccess object.
235      * @throw   RestrictionsException if no entry exists for Edge.
236      */
237     OsmAccess&          generalAccess();
238
239     /** Try to fetch the vehicle type specific access restrictions for this edge.
240      * @param   vehiceltType    The type of Vehicle to get access restriction
241      * @return  reference to the OsmAccess object.
242      * @throw   RestrictionsException if no entry exists for Edge.
243      */
244     const OsmAccess&    vehicleTypeAccess(
245                             OsmVehicle::VehicleType vehicleType) const;
246
247     /** Try to fetch the vehicle type specific access restrictions for this edge.
248      * @param   vehiceltType    The type of Vehicle to get access restriction
249      * @return  reference to the OsmAccess object.
250      * @throw   RestrictionsException if no entry exists for Edge.
251      */
252     OsmAccess&          vehicleTypeAccess(
253                             OsmVehicle::VehicleType vehicleType);
254
255     /** Get a list of the types of vehicles with restrictions on this edge.
256      * @return  a Vector with restriction types.
257      */
258     std::vector<OsmVehicle::VehicleType>
259                         vehicleTypesWithRestrictions() const;
260
261     /** Fetch the barrier restricting this edge.
262      * @return  reference to a OsmBarrier object.
263      * @throw   RestrictionsException if no entry exists for this Edge.
264      */
265     const OsmBarrier&   barrier() const;
266
267     /** Get a list of the turning restrictions from this edge.
268      * @return  a Vector with turning restrictions.
269      * @throw   RestrictionsException if edge has no turning restrictions.
270      */
271     const std::vector<OsmTurningRestriction*>&
272                         turningRestrictions() const;
273
274     /** Get a list of all edge id's to which travel from edge is not allowed.
275      * @return  A vector of edgeIds to which travel is not allowed.
276      */
277     std::vector<EdgeIdType>
278                         restrictedTargetEdges() const;
279
280 // INQUIRY
281
282     /** Ask if an Edge has restriction of a certain type.
283      * @param   restrictionType     The type of restriction
284      * @return  true if there is a restriction of that type, false if not.
285      */
286     bool                hasRestriction(
287                             RestrictionType restrictionType) const;
```

```
288     /**
289      * @return true if there is a VehicleProperty restriction for edge.
290      */
291     bool                hasVehiclePropertyRestriction() const;
292
293     /** Convenience method to query for max speed.
294      * @return true if there is a max speed restriction for the edge.
295      */
296     bool                hasMaxSpeedRestriction() const;
297
298     /**
299      * @return true if there is a General Access restriction for the edge.
300      */
301     bool                hasGeneralAccessRestriction() const;
302
303     /**
304      * @return true if there are any Vehicle Type Access restrictions for the edge.
305      */
306     bool                hasVehicleTypeAccessRestriction() const;
307
308     /**
309      * @return true if there are Vehicle Type Access restrictions for the edge
310      *         for that specific type of vehicle.
311      */
312     bool                hasVehicleTypeAccessRestriction(
313                             OsmVehicle::VehicleType vehicleType) const;
314
315     /**
316      * @return true if there are any barriers restricting access to the edge.
317      */
318     bool                hasBarrierRestriction() const;
319
320     /**
321      * @return true if there are any turning restrictions traveling from edge.
322      */
323     bool                hasTurningRestriction() const;
324
325     /**
326      * @return true if the edge is 'disused'.
327      */
328     bool                hasDisusedRestriction() const;
329
330     /**
331      * @return true if the edge has no exit.
332      */
333     bool                hasNoExitRestriction() const;
334
335     /**
336      * @return true if the edge is part of a turning restriction via another way.
337      */
338     bool                hasViaWayRestriction() const;
339
340
341 protected:
342 private:
343     VehicleProperties*      mpVehicleProperties {nullptr};
344     OsmAccess*              mpGeneralAccess {nullptr};
```

```
345     std::map<OsmVehicle::VehicleType,  OsmAccess*>
346                             mVehicleTypeAccessMap;
347     OsmBarrier*             mpBarrier {nullptr};
348     std::vector<OsmTurningRestriction*>
349                             mTurningRestrictions;
350     bool                    mIsDisusedEdge {false};
351     bool                    mIsNoExitEdge {false};
352     bool                    mHasViaWayRestriction {false};
353 };
354
355 // INLINE METHODS
356 //
357
358 // EXTERNAL REFERENCES
359 //
360
361 #endif /* GRAPH_EDGERESTRICTION_H_ */
```

D.7.15 EdgeRestriction.cc

```
1  /*
2   * EdgeRestriction.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "EdgeRestriction.h"  // class implemented
8
9  // STATIC INITIALIZATION
10 /*static*/ double EdgeRestriction::VehicleProperties::DEFAULT_DIMENSION_MAX
11     = std::numeric_limits<double>::max();
12
13 /*static*/ Speed  EdgeRestriction::VehicleProperties::DEFAULT_SPEED_MAX
14     = std::numeric_limits<unsigned>::max();
15
16 /*static*/ Speed  EdgeRestriction::VehicleProperties::DEFAULT_SPEED_MIN
17     = 0;
18
19 ////////////////////////////// PUBLIC //////////////////////////////////////
20 //============================ LIFECYCLE ====================================
21 EdgeRestriction::~EdgeRestriction()
22 {
23     delete mpVehicleProperties;
24     delete mpGeneralAccess;
25     delete mpBarrier;
26     for(auto it : mVehicleTypeAccessMap)
27     {
28         delete it.second;
29     }
30     for(auto it : mTurningRestrictions)
31     {
32         delete it;
33     }
34 }
35 //============================ OPERATORS ====================================
36 //============================ OPERATIONS ===================================
37 bool
```

```
38  EdgeRestriction::restricts(const Configuration& rConfig) const
39  {
40      bool is_restricted = false;
41      bool is_generally_restricted = false;
42      bool is_vehicle_banned = false;
43
44      for(const auto& r : restrictionTypes())
45      {
46          switch (r)
47          {
48              case EdgeRestriction::DISUSED:
49                  is_restricted = true; break;
50              case EdgeRestriction::VEHICLE_PROPERTIES:
51                  if(vehicleProperties()
52                      .restrictsAccess(rConfig.getVehicleConfig()))
53                  {
54                      is_restricted = true;
55                  }
56                  break;
57              case EdgeRestriction::BARRIER:
58                  if(barrier()
59                      .restrictsAccess(rConfig.getBarrierRestrictionsRule()))
60                  {
61                      is_restricted = true;
62                  }
63                  break;
64              case EdgeRestriction::GENERAL_ACCESS:
65                  if(!generalAccess()
66                      .allowsAccess(rConfig.getAccessRule()))
67                  {
68                      is_generally_restricted = true;
69                  }
70                  continue;
71              case EdgeRestriction::VEHICLE_TYPE_ACCESS:
72              {
73                  OsmVehicle::VehicleType type =
74                      rConfig.getVehicleConfig().category;
75                  if(hasVehicleTypeAccessRestriction(type))
76                  {
77                      if(!vehicleTypeAccess(type)
78                          .allowsAccess(rConfig.getAccessRule()))
79                      {
80                          is_vehicle_banned = true;
81                      }
82                  }
83              }
84              continue;
85              default:
86                  continue;
87          }
88      }
89
90      if(is_restricted
91          || (is_generally_restricted && is_vehicle_banned)
92          || is_vehicle_banned)
93      {
94          return true;
```

```
95      }
96      return false;
97  }
98
99  void
100 EdgeRestriction::setVehiclePropertyRestriction(
101     EdgeRestriction::VehicleProperties* pVehicleProperties)
102 {
103     delete mpVehicleProperties;
104     mpVehicleProperties = pVehicleProperties;
105 }
106
107 void
108 EdgeRestriction::setGeneralAccessRestriction(
109     OsmAccess*  pGeneralAccess)
110 {
111     delete mpGeneralAccess;
112     mpGeneralAccess = pGeneralAccess;
113 }
114
115 void
116 EdgeRestriction::setGeneralAccessRestriction(
117     OsmAccess::AccessType  generalAccessType)
118 {
119     delete mpGeneralAccess;
120     mpGeneralAccess = new OsmAccess(generalAccessType);
121 }
122
123 void
124 EdgeRestriction::addVehicleTypeAccessRestriction(
125     OsmVehicle::VehicleType  vehicleType,
126     OsmAccess*               pAccess)
127 {
128     if(hasVehicleTypeAccessRestriction(vehicleType))
129     {
130         auto old_access = mVehicleTypeAccessMap.find(vehicleType);
131         delete old_access->second;
132         mVehicleTypeAccessMap.erase(vehicleType);
133     }
134     mVehicleTypeAccessMap.insert({vehicleType, pAccess});
135 }
136
137 void
138 EdgeRestriction::addVehicleTypeAccessRestriction(
139     OsmVehicle::VehicleType  vehicleType,
140     OsmAccess::AccessType    accessType)
141 {
142     addVehicleTypeAccessRestriction(vehicleType, new OsmAccess(accessType));
143 }
144
145 void
146 EdgeRestriction::setBarrierRestriction(
147     OsmBarrier* pBarrier)
148 {
149     delete mpBarrier;
150     mpBarrier = pBarrier;
151 }
```

```
152
153 void
154 EdgeRestriction::setBarrierRestriction(
155     OsmBarrier::BarrierType barrierType)
156 {
157     delete mpBarrier;
158     mpBarrier = new OsmBarrier(barrierType);
159 }
160
161 void
162 EdgeRestriction::addTurningRestriction(
163     OsmTurningRestriction*   pTurningRestriction)
164 {
165     mTurningRestrictions.push_back(pTurningRestriction);
166 }
167
168 void
169 EdgeRestriction::setDisusedRestriction()
170 {
171     mIsDisusedEdge = true;
172 }
173
174 void
175 EdgeRestriction::setNoExitRestriction()
176 {
177     mIsNoExitEdge = true;
178 }
179
180 void
181 EdgeRestriction::setViaWayRestriction()
182 {
183     mHasViaWayRestriction = true;
184 }
185 //============================== ACESS      ===================================
186 std::vector<EdgeRestriction::RestrictionType>
187 EdgeRestriction::restrictionTypes() const
188 {
189     std::vector<EdgeRestriction::RestrictionType> rest_types;
190
191     for(int i = EdgeRestriction::VEHICLE_PROPERTIES;
192         i < EdgeRestriction::NR_RESTRICTION_TYPES;
193         ++i)
194     {
195         RestrictionType type = static_cast<RestrictionType>(i);
196         if(hasRestriction(type))
197         {
198             rest_types.push_back(type);
199         }
200     }
201
202     return rest_types;
203 }
204
205 const EdgeRestriction::VehicleProperties&
206 EdgeRestriction::vehicleProperties() const
207 {
208     if(!hasVehiclePropertyRestriction())
```

```
209     {
210         throw RestrictionsException(
211             "Restrictions:vehicleProperties: no restriction for edge");
212     }
213     return *mpVehicleProperties;
214 }
215 
216 EdgeRestriction::VehicleProperties&
217 EdgeRestriction::vehicleProperties()
218 {
219     if(!hasVehiclePropertyRestriction())
220     {
221         throw RestrictionsException(
222             "Restrictions:vehicleProperties: no restriction for edge");
223     }
224     return *mpVehicleProperties;
225 }
226 
227 Speed
228 EdgeRestriction::maxSpeed() const
229 {
230     if(hasVehiclePropertyRestriction())
231     {
232         return mpVehicleProperties->maxSpeed;
233     }
234     return VehicleProperties::DEFAULT_SPEED_MAX;
235 }
236 
237 const OsmAccess&
238 EdgeRestriction::generalAccess() const
239 {
240     if(!hasGeneralAccessRestriction())
241     {
242         throw RestrictionsException(
243             "Restrictions:generalAccess: no restriction for edge");
244     }
245     return *mpGeneralAccess;
246 }
247 
248 OsmAccess&
249 EdgeRestriction::generalAccess()
250 {
251     if(!hasGeneralAccessRestriction())
252     {
253         throw RestrictionsException(
254             "Restrictions:generalAccess: no restriction for edge");
255     }
256     return *mpGeneralAccess;
257 }
258 
259 const OsmAccess&
260 EdgeRestriction::vehicleTypeAccess(
261     OsmVehicle::VehicleType vehicleType) const
262 {
263     if(!hasVehicleTypeAccessRestriction(vehicleType))
264     {
265         throw RestrictionsException(
```

```
266             std::string("Restrictions:vehicleTypeAccess: no restriction for")
267             + " vehicle type " + OsmVehicle::toString(vehicleType)
268             + " for edge ");
269     }
270     return *(mVehicleTypeAccessMap.find(vehicleType)->second);
271 }
272
273 OsmAccess&
274 EdgeRestriction::vehicleTypeAccess(
275     OsmVehicle::VehicleType vehicleType)
276 {
277     return const_cast<OsmAccess&>
278         (static_cast<const EdgeRestriction&>
279             (*this).vehicleTypeAccess(vehicleType)
280         );
281 }
282
283 std::vector<OsmVehicle::VehicleType>
284 EdgeRestriction::vehicleTypesWithRestrictions() const
285 {
286     std::vector<OsmVehicle::VehicleType> types;
287
288     for( int i = 0; i < OsmVehicle::NR_VEHICLE_TYPES; ++i)
289     {
290         OsmVehicle::VehicleType type = static_cast<OsmVehicle::VehicleType>(i);
291
292         if(hasVehicleTypeAccessRestriction(type))
293         {
294             types.push_back(type);
295         }
296     }
297
298     return types;
299 }
300
301 const OsmBarrier&
302 EdgeRestriction::barrier() const
303 {
304     if(!hasBarrierRestriction())
305     {
306         throw RestrictionsException(
307             "Restrictions:barrier: no restriction for edge");
308     }
309     return *mpBarrier;
310 }
311
312 const std::vector<OsmTurningRestriction*>&
313 EdgeRestriction::turningRestrictions() const
314 {
315     if(!hasTurningRestriction())
316     {
317         throw RestrictionsException(
318             "Restriction:turningRestriction: no turning restriction for edge");
319     }
320     return mTurningRestrictions;
321 }
322
```

```
323 std::vector<EdgeIdType>
324 EdgeRestriction::restrictedTargetEdges() const
325 {
326     std::vector<EdgeIdType> restricted_targets;
327 
328     try
329     {
330         const auto& r_vec = this->turningRestrictions();
331 
332         for(const auto& restr : r_vec)
333         {
334             restricted_targets.push_back(restr->toEdgeId());
335         }
336     }
337     catch (RestrictionsException& re)
338     {
339         // never mind
340     }
341     return restricted_targets;
342 }
343 
344 //============================== INQUIRY    ==================================
345 bool
346 EdgeRestriction::hasRestriction(
347     EdgeRestriction::RestrictionType type) const
348 {
349     switch (type)
350     {
351         case VEHICLE_PROPERTIES:
352             return hasVehiclePropertyRestriction(); break;
353         case GENERAL_ACCESS:
354             return hasGeneralAccessRestriction(); break;
355         case VEHICLE_TYPE_ACCESS:
356             return hasVehicleTypeAccessRestriction(); break;
357         case BARRIER:
358             return hasBarrierRestriction(); break;
359         case TURNING:
360             return hasTurningRestriction(); break;
361         case DISUSED:
362             return hasDisusedRestriction(); break;
363         case NO_EXIT:
364             return hasNoExitRestriction(); break;
365         default:
366             return false;
367     }
368 }
369 
370 bool
371 EdgeRestriction::hasVehiclePropertyRestriction() const
372 {
373     return mpVehicleProperties != nullptr;
374 }
375 
376 bool
377 EdgeRestriction::hasMaxSpeedRestriction() const
378 {
379     if(hasVehiclePropertyRestriction())
```

```
380     {
381         return mpVehicleProperties->maxSpeed != VehicleProperties::DEFAULT_SPEED_MAX;
382     }
383     return false;
384 }
385 
386 bool
387 EdgeRestriction::hasGeneralAccessRestriction() const
388 {
389     return mpGeneralAccess != nullptr;
390 }
391 
392 bool
393 EdgeRestriction::hasVehicleTypeAccessRestriction() const
394 {
395     return mVehicleTypeAccessMap.size() > 0;
396 }
397 
398 bool
399 EdgeRestriction::hasVehicleTypeAccessRestriction(
400     OsmVehicle::VehicleType vehicleType) const
401 {
402     auto it = mVehicleTypeAccessMap.find(vehicleType);
403     if (it != mVehicleTypeAccessMap.end())
404     {
405         return true;
406     }
407     return false;
408 }
409 
410 bool
411 EdgeRestriction::hasBarrierRestriction() const
412 {
413     return  mpBarrier != nullptr;
414 }
415 
416 bool
417 EdgeRestriction::hasTurningRestriction() const
418 {
419     return mTurningRestrictions.size() > 0;
420 }
421 
422 bool
423 EdgeRestriction::hasDisusedRestriction() const
424 {
425     return mIsDisusedEdge;
426 }
427 
428 bool
429 EdgeRestriction::hasNoExitRestriction() const
430 {
431     return mIsNoExitEdge;
432 }
433 
434 bool
435 EdgeRestriction::hasViaWayRestriction() const
436 {
```

```
437     return mHasViaWayRestriction;
438 }
439 ////////////////////////////// PROTECTED  /////////////////////////////////
440
441 ////////////////////////////// PRIVATE    /////////////////////////////////
```

D.7.16 TurnCostCalculator.h

```
1  /** Calculate the turn cost for making a turn between to edges (roads).
2   *
3   * #include "TurnCostCalculator.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef GRAPH_TURNCOSTCALCULATOR_H_
9  #define GRAPH_TURNCOSTCALCULATOR_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <initializer_list>
14 #include <cstdlib> // abs()
15
16 // PROJECT INCLUDES
17 //
18
19 // LOCAL INCLUDES
20 //
21 #include "Cost.h"
22 #include "Edge.h"
23 #include "Speed.h"
24 #include "../config/Configuration.h"
25
26 // FORWARD REFERENCES
27 //
28
29 // TYPES
30 //
31
32 /**
33  * Calculate the cost for making turns.
34  * Based on "Route Planning in Road Networks with Turn Costs"
35  * by Lars Volker. Universitat Karslruhe 2008.
36  * http://algo2.iti.kit.edu/documents/routeplanning/volker_sa.pdf
37  *
38  * The cost is dependent on
39  * - angle between roads
40  * - size of vehicle
41  * - road category precedence
42  */
43 class TurnCostCalculator
44 {
45 public:
46 // LIFECYCLE
47     TurnCostCalculator() = delete;
48     virtual ~TurnCostCalculator() = delete;
49
```

```
50  // OPERATORS
51  // OPERATIONS
52  // ACCESS
53      /**
54       * @return  The cost of the turn.
55       */
56      static double   getTurnCost(
57                          const Edge& rSource,
58                          const Edge& rTarget,
59                          const Configuration& rConfig);
60  // INQUIRY
61  private:
62  // HELPERS
63      /**
64       * @param   rSource     Source edge.
65       * @param   rTarget     Target edge.
66       * @return  The speed dependent on the angle between edges.
67       */
68      static Speed    getAngleSpeed(
69                          const Edge& rSource,
70                          const Edge& rTarget);
71
72      /**
73       * @param   rSource         Source edge.
74       * @param   rTarget         Target edge.
75       * @param   vehicle_length  Length of the vehicle.
76       * @param   angleSpeed      The angle dependent speed
77       * @return  The speed dependent on the size of the routed vehicle.
78       */
79      static Speed    getVehicleSizeSpeed(
80                          const Edge& rSource,
81                          const Edge& rTarget,
82                          double      vehicle_length,
83                          Speed       angleSpeed);
84
85      /**
86       * @param   speeds      A set of speeds.
87       * @return  The smallest speed
88       */
89      static Speed    getSmallestSpeed(std::initializer_list<Speed> speeds);
90
91      /** Get the angle between source and target as
92       * -180 < angle < 180
93       * That means that 0 is straight ahead, > 0 to the right
94       * and < 0 to the left.
95       */
96      static int      getTurnAngle(
97                          const Edge& rSource,
98                          const Edge& rTarget);
99
100     /** Calculate a penalty for making sharp right turns with long vehicles.
101      * @param   turnAngle       The turning angle in degrees (-180 < a < 180).
102      * @param   vehicleLength   The length of the vehicle.
103      * @return  A factor 0.33 - 1.0
104      */
105     static double   calculateLengthPenaltyFactor(
106                         int     turnAngle,
```

```
107                         double  vehicleLength);
108
109     /** Look if target is of a more important highway type than the source,
110      * in that case we must add a penalty for giving way when entering
111      * the target road.
112      * @param   rSource     The source edge.
113      * @param   rTarget     The target edge.
114      * @return  A cost for giving way.
115      */
116     static Cost     giveWayToHigherRoadCategoryCost(
117                         const Edge& rSource,
118                         const Edge& rTarget);
119
120 // ATTRIBUTES
121 // CONSTANTS
122     static constexpr double VEHICLE_PENALTY_LENGTH = 4.5;
123
124 };
125
126 // INLINE METHODS
127 //
128
129 // EXTERNAL REFERENCES
130 //
131
132 #endif /* GRAPH_TURNCOSTCALCULATOR_H_ */
```

D.7.17 TurnCostCalculator.cc

```
1  /*
2   * TurnCostCalculator.cc
3   */
4
5  #include "TurnCostCalculator.h" // class implemented
6
7  ////////////////////////////// PUBLIC //////////////////////////////////////
8  //============================== LIFECYCLE ====================================
9  //============================== OPERATORS ====================================
10 //============================== OPERATIONS ===================================
11 //============================== ACESS      ===================================
12 /* static */
13 double
14 TurnCostCalculator::getTurnCost(
15     const Edge& rSource,
16     const Edge& rTarget,
17     const Configuration& rConfig)
18 {
19     double vehicle_length = rConfig.getVehicleConfig().length;
20
21     Speed angle_speed = getAngleSpeed(rSource, rTarget);
22     Speed size_speed =
23         getVehicleSizeSpeed(rSource, rTarget, vehicle_length, angle_speed);
24     Speed turn_speed = getSmallestSpeed({angle_speed, size_speed});
25
26     double decel_factor = rConfig.getVehicleConfig().acceleration / 100.0;
27     double accel_factor = rConfig.getVehicleConfig().deceleration / 100.0;
28
```

```
29      Cost deceleration_cost = decel_factor * (rSource.speed() - turn_speed);
30      Cost acceleration_cost = accel_factor * (rTarget.speed() - turn_speed);
31      Cost additional_cost = giveWayToHigherRoadCategoryCost(rSource, rTarget);
32
33      Cost turn_cost = deceleration_cost + acceleration_cost + additional_cost;
34
35      return static_cast<double>(turn_cost);
36  }
37  //============================== INQUIRY    ==================================
38  ////////////////////////////////// PROTECTED  ////////////////////////////////////
39
40  ////////////////////////////////// PRIVATE    ////////////////////////////////////
41  /* static */
42  Speed
43  TurnCostCalculator::getAngleSpeed(const Edge& rSource, const Edge& rTarget)
44  {
45      int turn_angle = getTurnAngle(rSource, rTarget);
46      // make sure there is some speed and not 0
47      if(abs(turn_angle) > 175)
48      {
49          turn_angle = 175;
50      }
51      double reduction_factor(1 - (abs(turn_angle)/180.0));
52      double speed =
53          reduction_factor * getSmallestSpeed({rSource.speed(), rTarget.speed()});
54      return static_cast<Speed>(speed);
55  }
56
57  /* static */
58  Speed
59  TurnCostCalculator::getVehicleSizeSpeed(
60      const Edge& rSource,
61      const Edge& rTarget,
62      double      vehicle_length,
63      Speed       angleSpeed)
64  {
65      int turn_angle (getTurnAngle(rSource, rTarget));
66      double length_penalty_factor =
67          calculateLengthPenaltyFactor(turn_angle, vehicle_length);
68
69      Speed speed = angleSpeed
70                    * (VEHICLE_PENALTY_LENGTH / vehicle_length)
71                    * length_penalty_factor;
72      return speed;
73  }
74
75  /* static */
76  Speed
77  TurnCostCalculator::getSmallestSpeed(std::initializer_list<Speed> speeds)
78  {
79      Speed min {1000};
80
81      if(speeds.size() > 0)
82      {
83          for(const auto& s : speeds)
84          {
85              if(s < min)
```

```
86              {
87                  min = s;
88              }
89          }
90      }
91      return min;
92  }
93
94  /* static */
95  int
96  TurnCostCalculator::getTurnAngle(const Edge& rSource, const Edge& rTarget)
97  {
98      int angle =
99          rSource.geomData().targetBearing - rTarget.geomData().sourceBearing;
100     if(angle < -180)
101     {
102         angle += 360;
103     }
104     if(angle > 180)
105     {
106         angle -= 360;
107     }
108     return angle;
109 }
110
111 /* static */
112 double
113 TurnCostCalculator::calculateLengthPenaltyFactor(
114     int turnAngle,
115     double vehicleLength)
116 {
117     double factor(1.0);
118
119     if(vehicleLength > VEHICLE_PENALTY_LENGTH)
120     {
121         if(turnAngle > 0)
122         {
123             factor = 1.0 - ((2.0/3.0) * (turnAngle/180.0));
124         }
125     }
126     return factor;
127 }
128
129 /* static */
130 Cost
131 TurnCostCalculator::giveWayToHigherRoadCategoryCost(
132     const Edge& rSource,
133     const Edge& rTarget)
134 {
135     if(rSource.roadData().roadType > rTarget.roadData().roadType)
136     {
137         return 5;
138     }
139     return 0;
140 }
```

D.7.18 GraphException.h

```
/**  Exception thrown by the Graph package.
 *
 * #include "GraphException.h"
 *
 * @author  Jonas Bergman
 */

#ifndef GRAPH_GRAPHEXCEPTION_H_
#define GRAPH_GRAPHEXCEPTION_H_

// SYSTEM INCLUDES
//
#include <exception>
#include <string>

// PROJECT INCLUDES
//

// LOCAL INCLUDES
//

// FORWARD REFERENCES
//

/**
 * Exception to throw from the 'graph' package.
 * More information of the type of exception is given in the 'what()' message.
 */
class GraphException : public std::exception
{
public:
// LIFECYCLE
    /** Default constructor.
     */
    GraphException() = delete;

    /** Constructor taking a message to display.
     *
     * @param    message    The message to prepend when 'what()' is called.
     */
    GraphException(const std::string& rMessage) noexcept
        : std::exception(), mMessage(rMessage)
    {}

// OPERATORS
// OPERATIONS
// ACCESS
// INQUIRY
    const char* what() const noexcept
    { return mMessage.c_str(); }

protected:
private:
// ATTRIBUTES
    std::string        mMessage;
};
```

```
57
58 // INLINE METHODS
59 //
60
61 // EXTERNAL REFERENCES
62 //
63
64 #endif /* GRAPH_GRAPHEXCEPTION_H_ */
```

D.7.19 RestrictionsException.h

```
1  /**  Exception thrown by the Restrictions.
2   *
3   * #include "RestrictionsException.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef GRAPH_RESTRICTIONSEXCEPTION_H_
9  #define GRAPH_RESTRICTIONSEXCEPTION_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <exception>
14 #include <string>
15
16 // PROJECT INCLUDES
17 //
18
19 // LOCAL INCLUDES
20 //
21
22 // FORWARD REFERENCES
23 //
24
25 /**
26  * Exception to throw from Restrictions class.
27  * More information of the type of exception is given in the 'what()' message.
28  */
29 class RestrictionsException : public std::exception
30 {
31 public:
32 // LIFECYCLE
33     /** Default constructor.
34      */
35     RestrictionsException() = delete;
36
37     /** Constructor taking a message to display.
38      *
39      * @param   message     The message to prepend when 'what()' is called.
40      */
41     RestrictionsException(const std::string& rMessage) noexcept
42         : std::exception(), mMessage(rMessage)
43     {}
44
45 // OPERATORS
46     void    addEdgeId(std::string edgeIdString) { mMessage += edgeIdString; }
```

```
47  // OPERATIONS
48  // ACCESS
49  // INQUIRY
50      const char* what() const noexcept
51      { return mMessage.c_str(); }
52
53  protected:
54  private:
55  // ATTRIBUTES
56      std::string     mMessage;
57  };
58
59  // INLINE METHODS
60  //
61
62  // EXTERNAL REFERENCES
63  //
64
65  #endif /* GRAPH_RESTRICTIONSEXCEPTION_H_ */
```

D.7.20 TopologyException.h

```
1   /**  Exception thrown by the Topology package.
2    *
3    * #include "TopologyException.h"
4    *
5    * @author  Jonas Bergman
6    */
7
8   #ifndef GRAPH_TOPOLOGYEXCEPTION_H_
9   #define GRAPH_TOPOLOGYEXCEPTION_H_
10
11  // SYSTEM INCLUDES
12  //
13  #include <exception>
14  #include <string>
15
16  // PROJECT INCLUDES
17  //
18
19  // LOCAL INCLUDES
20  //
21
22  // FORWARD REFERENCES
23  //
24
25  /**
26   * Exception to throw from the 'topology' package.
27   * More information of the type of exception is given in the 'what()' message.
28   */
29  class TopologyException : public std::exception
30  {
31  public:
32  // LIFECYCLE
33      /** Default constructor.
34       */
35      TopologyException() = delete;
```

```
36
37      /** Constructor taking a message to display.
38       *
39       * @param    message        The message to prepend when 'what()' is called.
40       */
41      TopologyException(const std::string& rMessage) noexcept
42          : std::exception(), mMessage(rMessage)
43      {}
44
45  // OPERATORS
46  // OPERATIONS
47  // ACCESS
48  // INQUIRY
49      const char* what() const noexcept
50      { return mMessage.c_str(); }
51
52  protected:
53  private:
54  // ATTRIBUTES
55      std::string        mMessage;
56  };
57
58  // INLINE METHODS
59  //
60
61  // EXTERNAL REFERENCES
62  //
63
64  #endif /* GRAPH_TOPOLOGYEXCEPTION_H_ */
```

D.7.21 EdgeCost_test.cc

```
1   /* Tests for EdgeCost class
2    * EdgeCost_test.cc
3    *
4    * @author  Jonas Bergman
5    */
6
7   #include "../../catchtest/catch.hpp"
8
9   #include "../EdgeCost.h"
10
11  SCENARIO ("Keeping track of costs for an Edge", "[graph][edgecost]")
12  {
13      EdgeCost costs;
14
15      GIVEN ("an EdgeCost object")
16      {
17          WHEN ("no costs are added")
18          {
19              THEN ("there should be no costs")
20              {
21                  REQUIRE (costs.getCost() == 0);
22                  REQUIRE (costs.hasCost(EdgeCost::TRAVEL_TIME) == false);
23                  REQUIRE (costs.getCost(EdgeCost::BARRIER) == 0);
24              }
25          }
```

```
26
27          WHEN ("travel cost is added")
28          {
29              costs.addCost(EdgeCost::TRAVEL_TIME, 10);
30
31              THEN ("there should be costs")
32              {
33                  REQUIRE (costs.getCost() > 0);
34                  REQUIRE (costs.hasCost(EdgeCost::TRAVEL_TIME) == true);
35                  REQUIRE (costs.getCost(EdgeCost::TRAVEL_TIME) == Approx(10.0));
36                  REQUIRE (costs.getCost(EdgeCost::BARRIER) == 0);
37              }
38          }
39
40          WHEN ("two travel costs are added")
41          {
42              costs.addCost(EdgeCost::TRAVEL_TIME, 10);
43              costs.addCost(EdgeCost::TRAVEL_TIME, 20);
44
45              THEN ("only the last should be reported")
46              {
47                  REQUIRE (costs.hasCost(EdgeCost::TRAVEL_TIME) == true);
48                  REQUIRE (costs.getCost(EdgeCost::TRAVEL_TIME) == Approx(20.0));
49              }
50          }
51
52          WHEN ("travel a travel and a barrier cost are added")
53          {
54              costs.addCost(EdgeCost::TRAVEL_TIME, 10);
55              costs.addCost(EdgeCost::BARRIER, 20);
56
57              THEN ("the costs should be added")
58              {
59                  REQUIRE (costs.getCost() == Approx(30.0));
60                  REQUIRE (costs.hasCost(EdgeCost::TRAVEL_TIME) == true);
61                  REQUIRE (costs.hasCost(EdgeCost::BARRIER) == true);
62                  REQUIRE (costs.hasCost(EdgeCost::OTHER) == false);
63              }
64          }
65      }
66  }
```

D.7.22 GraphBuilder_test.cc

```
1   /*
2    * Graph_test.cc
3    *
4    * @author  Jonas Bergman
5    */
6
7   #include <iostream>
8
9   #include "../GraphBuilder.h"
10  #include "../Topology.h"
11  #include "../../catchtest/catch.hpp"
12  #include "../../config/ConfigurationReader.h"
13  #include "../../mapprovider/postgis/PostGisProvider.h"
```

```
14
15  SCENARIO ("Building a small graph", "[graph][basic]")
16  {
17      // ---------------------------------------------------------------------
18      GIVEN ("Three points and two edges for a topology")
19      {
20          size_t nr_vertices = 3;
21          size_t nr_edges = 2;
22          OsmIdType osm_id(1);
23
24          Topology topology;
25          const Vertex& v1 = topology.addVertex(1, Point(0,0));
26          const Vertex& v2 = topology.addVertex(2, Point(1,2));
27          const Vertex& v3 = topology.addVertex(3, Point(3,1));
28          Edge& e1 = topology.addEdge(1,osm_id,1,2);
29          Edge& e2 = topology.addEdge(2,osm_id,2,3);
30
31          Configuration config;
32
33          // .................................................................
34          WHEN ("we try create a Graph from the Topology")
35          {
36              THEN ("we should not get an Exception")
37              {
38                  INFO ("calling Graph constructor");
39                  REQUIRE_NOTHROW (GraphBuilder g(topology, config));
40              }
41          }
42
43          // .................................................................
44          WHEN ("building a graph from the topology")
45          {
46              GraphBuilder g(topology, config);
47              const auto& boost_graph = g.getBoostGraph();
48              LineGraphType& r_boost_line_graph = g.getBoostLineGraph();
49
50              // . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
51              THEN ("the # of vertices in the graph representation"
52                  " should be as in the topology"
53                  " and the # edges the double") // default is bidirectional
54              {
55                  REQUIRE (boost::num_vertices(boost_graph) == nr_vertices);
56                  REQUIRE (boost::num_edges(boost_graph) == nr_edges * 2);
57              }
58
59              // . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
60              THEN ("the number of nodes in the LineGraph"
61                    " should be as many as edges in the graph")
62              {
63                  REQUIRE (boost::num_vertices(r_boost_line_graph) ==
64                           boost::num_edges(boost_graph));
65              }
66          }
67
68          // .................................................................
69          WHEN ("we try print out a Graph from the Topology")
70          {
```

```
71              GraphBuilder g(topology, config);
72
73              THEN ("we should get a print out")
74              {
75                  INFO (g);
76                  REQUIRE (true);
77              }
78          }
79
80          // ....................................................................
81          WHEN ("adding unidirectional information to edges before"
82                " building graph")
83          {
84
85              Edge::RoadData rd1;
86              rd1.direction = Edge::DirectionType::FROM_TO;
87              e1.setRoadData(rd1);
88
89              Edge::RoadData rd2;
90              rd2.direction = Edge::DirectionType::FROM_TO;
91              e2.setRoadData(rd2);
92
93              GraphBuilder g2(topology, config);
94
95              THEN ("the # of edges in the graph representation"
96                    " should as many as in the topology")
97              {
98                  INFO (g2);
99                  const auto& boost_graph = g2.getBoostGraph();
100                 REQUIRE (boost::num_edges(boost_graph) == topology.nrEdges());
101             }
102         }
103
104         // ....................................................................
105         WHEN ("adding an extra lane to an edge before"
106               " building graph")
107         {
108
109             Edge::RoadData rd1;
110             rd1.direction = Edge::DirectionType::FROM_TO;
111             rd1.nrLanes = 2;
112             e1.setRoadData(rd1);
113
114             Edge::RoadData rd2;
115             rd2.direction = Edge::DirectionType::FROM_TO;
116             e2.setRoadData(rd2);
117
118             GraphBuilder g2(topology, config);
119
120             THEN ("the # of edges in the graph representation"
121                 " should be one more than in the topology")
122             {
123                 INFO (g2);
124                 const auto& boost_graph = g2.getBoostGraph();
125                 REQUIRE (boost::num_edges(boost_graph) == topology.nrEdges() + 1);
126             }
127         }
```

```
128     }
129 }
130
131 SCENARIO ("Building graph with restrictions", "[graph][restrictions]")
132 {
133     try
134     {
135
136         // ============================================================
137         GIVEN ("Configuration to build a Graph with restrictions ")
138         {
139             std::string config_file(
140                 "catchtest/testsettings/mikh_restr_0617-testsettings.json");
141             ConfigurationReader config_reader(config_file);
142             Configuration config;
143             config_reader.fillConfiguration(config);
144
145             PostGisProvider pgp(config);
146
147             Topology topology;
148             pgp.getTopology(topology);
149             pgp.setRestrictionsAndCosts(topology);
150
151             GraphBuilder graph_restr(topology, config);
152
153             GraphBuilder graph_unrestr(topology, config, false);
154
155             // ..........................................................
156             WHEN ("Adding a turning restriction and a point restriction (barrier)")
157             {
158
159                 THEN ("there should be equally many vertices "
160                       "in restricted and unrestricted")
161                 {
162                     INFO ("  Restricted # Vertices: " << graph_restr.nrVertices());
163                     INFO ("UNRestricted # Vertices: " << graph_unrestr.nrVertices());
164                     REQUIRE (graph_restr.nrVertices() == graph_unrestr.nrVertices());
165                 }
166
167                 THEN ("there should be 2 less edges "
168                       "in restricted and unrestricted")
169                 {
170                     INFO ("  Restricted # Edges:    " << graph_restr.nrEdges());
171                     INFO ("UNRestricted # Edges:    " << graph_unrestr.nrEdges());
172                     REQUIRE (graph_restr.nrEdges() == graph_unrestr.nrEdges() - 2);
173                 }
174
175                 THEN ("there should be 2 less nodes "
176                       "in restricted and unrestricted")
177                 {
178                     INFO ("  Restricted # Nodes:    " << graph_restr.nrNodes());
179                     INFO ("UNRestricted # Nodes:    " << graph_unrestr.nrNodes());
180                     REQUIRE (graph_restr.nrNodes() == graph_unrestr.nrNodes() - 2);
181                 }
182                 THEN ("there should be 9 lines less "
183                       "in restricted than unrestricted")
184                 {
```

```
185                     // 1 turn restriction
186                     // 3*2 where target is restricted by barrier (lift gate)
187                     // 2 u-turns on restricted edge
188                     INFO ("  Restricted # Lines:    " << graph_restr.nrLines());
189                     INFO ("UNRestricted # Lines:    " << graph_unrestr.nrLines());
190                     REQUIRE (graph_restr.nrLines() == graph_unrestr.nrLines() - 9);
191                 }
192                 THEN ("we can print the info for an edge and it should have a cost")
193                 {
194                     EdgeIdType id = 270;
195                     const Edge& edge = topology.getEdge(id);
196                     INFO ("Edge " << id << ": " << edge);
197                     REQUIRE (true);
198                 }
199             }
200         }
201     }
202     catch (ConfigurationException& e)
203     {
204         INFO(e.what());
205         REQUIRE (false);    // force output of error and failure
206     }
207     catch (MapProviderException& dbe)
208     {
209         INFO(dbe.what());
210         REQUIRE (false);    // force output of error and failure
211 
212     }
213 }
```

D.7.23 RestrictionsAndCosts_test.cc

```
1  /* Tests for the different kind of restrictions
2   *
3   * Graph_test.cc
4   * @author  Jonas Bergman
5   */
6 
7  #include <iostream>
8 
9  #include "../../catchtest/catch.hpp"
10 
11 #include "../Topology.h"
12 #include "../../config/ConfigurationReader.h"
13 #include "../../mapprovider/postgis/PostGisProvider.h"
14 #include "../GraphBuilder.h"
15 
16 // TURN RESTRICTION ////////////////////////////////////////////////////////
17 
18 SCENARIO ("Building graph of Mikhailovsk with turn restriction",
19     "[graph][r_and_c][turn][mikhailovsk]")
20 {
21     // block on node 1706164751 on way 158421713
22     try
23     {
24         std::string orig_config_file("catchtest/testsettings/"
25             "restrictions/mikhailovsk-original.json");
```

```
26          ConfigurationReader orig_config_reader(orig_config_file);
27          Configuration orig_config;
28          orig_config_reader.fillConfiguration(orig_config);
29          PostGisProvider orig_pgp(orig_config);
30          Topology orig_topology;
31          orig_pgp.getTopology(orig_topology);
32          orig_pgp.setRestrictionsAndCosts(orig_topology);
33          GraphBuilder orig_graph(orig_topology, orig_config);
34
35          // ================================================================
36          GIVEN ("Configuration to build a Graph with turn restriction ")
37          {
38              std::string config_file("catchtest/testsettings/"
39                  "restrictions/mikhailovsk-turn_no_right.json");
40              ConfigurationReader config_reader(config_file);
41              Configuration config;
42              config_reader.fillConfiguration(config);
43
44              PostGisProvider pgp(config);
45
46              Topology topology;
47              pgp.getTopology(topology);
48              pgp.setRestrictionsAndCosts(topology);
49
50              GraphBuilder graph(topology, config);
51
52              // ........................................................
53              WHEN ("Comparing original to graph with turn restrictions")
54              {
55
56                  THEN ("there should be equally many vertices "
57                        "in original and restricted")
58                  {
59                      INFO ("  Original # Vertices: " << orig_graph.nrVertices());
60                      INFO ("Restricted # Vertices: " << graph.nrVertices());
61                      REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
62                  }
63
64                  THEN ("there should be equally many edges "
65                        "in original as in restricted")
66                  {
67                      INFO ("  Original # Edges: " << orig_graph.nrEdges());
68                      INFO ("Restricted # Edges: " << graph.nrEdges());
69                      REQUIRE ((orig_graph.nrEdges()) == graph.nrEdges());
70                  }
71
72                  THEN ("there should be equally many nodes "
73                        "in original as in restricted")
74                  {
75                      INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
76                      INFO ("Restricted # Nodes:    " << graph.nrNodes());
77                      REQUIRE ((orig_graph.nrNodes()) == graph.nrNodes());
78                  }
79
80                  THEN ("there should be 1 less line "
81                        "in original than in restricted")
82                  {
```

```
                    // 1 right turn
                    INFO ("  Original # Lines:    " << orig_graph.nrLines());
                    INFO ("Restricted # Lines:    " << graph.nrLines());
                    REQUIRE ((orig_graph.nrLines() - 1) == graph.nrLines());
                }
            }
        }
    }
    catch (ConfigurationException& e)
    {
        INFO(e.what());
        REQUIRE (false);    // force output of error and failure
    }
    catch (MapProviderException& dbe)
    {
        INFO(dbe.what());
        REQUIRE (false);    // force output of error and failure

    }
}

SCENARIO ("Building graph of Partille with turn restriction",
    "[graph][r_and_c][turn][partille]")
{
    // block on node 1706164751 on way 158421713
    try
    {
        std::string orig_config_file("catchtest/testsettings/"
            "restrictions/partille-original.json");
        ConfigurationReader orig_config_reader(orig_config_file);
        Configuration orig_config;
        orig_config_reader.fillConfiguration(orig_config);
        PostGisProvider orig_pgp(orig_config);
        Topology orig_topology;
        orig_pgp.getTopology(orig_topology);
        orig_pgp.setRestrictionsAndCosts(orig_topology);
        GraphBuilder orig_graph(orig_topology, orig_config);

        // ========================================================================
        GIVEN ("Configuration to build a Graph with turn restriction ")
        {
            std::string config_file("catchtest/testsettings/"
                "restrictions/partille-turn_no_left.json");
            ConfigurationReader config_reader(config_file);
            Configuration config;
            config_reader.fillConfiguration(config);

            PostGisProvider pgp(config);

            Topology topology;
            pgp.getTopology(topology);
            pgp.setRestrictionsAndCosts(topology);

            GraphBuilder graph(topology, config);

            // ------------------------------------------------------------
            WHEN ("Comparing original to graph with turn restrictions")
```

```
140             {
141
142                 THEN ("there should be equally many vertices "
143                       "in original and restricted")
144                 {
145                     INFO ("  Original # Vertices: " << orig_graph.nrVertices());
146                     INFO ("Restricted # Vertices: " << graph.nrVertices());
147                     REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
148                 }
149
150                 THEN ("there should be equally many edges "
151                       "in original as in restricted")
152                 {
153                     INFO ("  Original # Edges: " << orig_graph.nrEdges());
154                     INFO ("Restricted # Edges: " << graph.nrEdges());
155                     REQUIRE ((orig_graph.nrEdges()) == graph.nrEdges());
156                 }
157
158                 THEN ("there should be equally many nodes "
159                       "in original as in restricted")
160                 {
161                     INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
162                     INFO ("Restricted # Nodes:    " << graph.nrNodes());
163                     REQUIRE ((orig_graph.nrNodes()) == graph.nrNodes());
164                 }
165
166                 THEN ("there should be 1 less line "
167                       "in original than in restricted")
168                 {
169                     // 1 right turn
170                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
171                     INFO ("Restricted # Lines:    " << graph.nrLines());
172                     REQUIRE ((orig_graph.nrLines() - 1) == graph.nrLines());
173                 }
174             }
175         }
176     }
177     catch (ConfigurationException& e)
178     {
179         INFO(e.what());
180         REQUIRE (false);    // force output of error and failure
181     }
182     catch (MapProviderException& dbe)
183     {
184         INFO(dbe.what());
185         REQUIRE (false);    // force output of error and failure
186
187     }
188 }
189
190 // BARRIER BLOCK ///////////////////////////////////////////////////////
191
192 SCENARIO ("Building graph of Mikhailovsk with barrier block",
193     "[graph][r_and_c][block][mikhailovsk]")
194 {
195     // block on node 1706164751 on way 158421713
196     try
```

```
197     {
198         std::string orig_config_file("catchtest/testsettings/"
199             "restrictions/mikhailovsk-original.json");
200         ConfigurationReader orig_config_reader(orig_config_file);
201         Configuration orig_config;
202         orig_config_reader.fillConfiguration(orig_config);
203         PostGisProvider orig_pgp(orig_config);
204         Topology orig_topology;
205         orig_pgp.getTopology(orig_topology);
206         orig_pgp.setRestrictionsAndCosts(orig_topology);
207         GraphBuilder orig_graph(orig_topology, orig_config);
208 
209         // ========================================================================
210         GIVEN ("Configuration that restricts barrier block ")
211         {
212             std::string config_file("catchtest/testsettings/"
213                 "restrictions/mikhailovsk-barrier_block.json");
214             ConfigurationReader config_reader(config_file);
215             Configuration config;
216             config_reader.fillConfiguration(config);
217 
218             PostGisProvider pgp(config);
219 
220             Topology topology;
221             pgp.getTopology(topology);
222             pgp.setRestrictionsAndCosts(topology);
223 
224             GraphBuilder graph(topology, config);
225 
226             // ....................................................................
227             WHEN ("Comparing original to graph with barrier block")
228             {
229 
230                 THEN ("there should be equally many vertices "
231                       "in original and restricted")
232                 {
233                     INFO ("  Original # Vertices: " << orig_graph.nrVertices());
234                     INFO ("Restricted # Vertices: " << graph.nrVertices());
235                     REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
236                 }
237 
238                 THEN ("there should be 2 more edges "
239                       "in original than in restricted")
240                 {
241                     INFO ("  Original # Edges: " << orig_graph.nrEdges());
242                     INFO ("Restricted # Edges: " << graph.nrEdges());
243                     REQUIRE ((orig_graph.nrEdges() - 2) == graph.nrEdges());
244                 }
245 
246                 THEN ("there should be 2 more nodes "
247                       "in original than in restricted")
248                 {
249                     INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
250                     INFO ("Restricted # Nodes:    " << graph.nrNodes());
251                     REQUIRE ((orig_graph.nrNodes() - 2) == graph.nrNodes());
252                 }
253 
```

```
254                 THEN ("there should be 10 more lines "
255                       "in original than in restricted")
256                 {
257                     // 4 bidirectional edges connecting = 8 lines
258                     // 2 u-turns = 2 lines
259                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
260                     INFO ("Restricted # Lines:    " << graph.nrLines());
261                     REQUIRE ((orig_graph.nrLines() - 10) == graph.nrLines());
262                 }
263             }
264         }
265     }
266     catch (ConfigurationException& e)
267     {
268         INFO(e.what());
269         REQUIRE (false);    // force output of error and failure
270     }
271     catch (MapProviderException& dbe)
272     {
273         INFO(dbe.what());
274         REQUIRE (false);    // force output of error and failure
275 
276     }
277 }
278 
279 SCENARIO ("Building graph of Partille with barrier block",
280     "[graph][r_and_c][block][partille]")
281 {
282     // block on node 249292683 on way 28050664
283     try
284     {
285         std::string orig_config_file("catchtest/testsettings/"
286             "restrictions/partille-original.json");
287         ConfigurationReader orig_config_reader(orig_config_file);
288         Configuration orig_config;
289         orig_config_reader.fillConfiguration(orig_config);
290         PostGisProvider orig_pgp(orig_config);
291         Topology orig_topology;
292         orig_pgp.getTopology(orig_topology);
293         orig_pgp.setRestrictionsAndCosts(orig_topology);
294         GraphBuilder orig_graph(orig_topology, orig_config);
295 
296         // ========================================================================
297         GIVEN ("Configuration that restricts barrier block ")
298         {
299             std::string config_file("catchtest/testsettings/"
300                 "restrictions/partille-barrier_block.json");
301             ConfigurationReader config_reader(config_file);
302             Configuration config;
303             config_reader.fillConfiguration(config);
304 
305             PostGisProvider pgp(config);
306 
307             Topology topology;
308             pgp.getTopology(topology);
309             pgp.setRestrictionsAndCosts(topology);
310 
```

```
311             GraphBuilder graph(topology, config);
312
313             // ..........................................................
314             WHEN ("Comparing original to graph with barrier block")
315             {
316
317                 THEN ("there should be equally many vertices "
318                       "in original and restricted")
319                 {
320                     INFO ("  Original # Vertices: " << orig_graph.nrVertices());
321                     INFO ("Restricted # Vertices: " << graph.nrVertices());
322                     REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
323                 }
324
325                 THEN ("there should be 2 more edges "
326                       "in original than in restricted")
327                 {
328                     INFO ("  Original # Edges: " << orig_graph.nrEdges());
329                     INFO ("Restricted # Edges: " << graph.nrEdges());
330                     REQUIRE ((orig_graph.nrEdges() - 2) == graph.nrEdges());
331                 }
332
333                 THEN ("there should be 2 more nodes "
334                       "in original than in restricted")
335                 {
336                     INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
337                     INFO ("Restricted # Nodes:    " << graph.nrNodes());
338                     REQUIRE ((orig_graph.nrNodes() - 2) == graph.nrNodes());
339                 }
340
341                 THEN ("there should be 8 more lines "
342                       "in original than in restricted")
343                 {
344                     // 3 bidirectional edges connecting = 6 lines
345                     // (1 cycleway = 0 lines)
346                     // 2 u-turns = 2 lines
347                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
348                     INFO ("Restricted # Lines:    " << graph.nrLines());
349                     REQUIRE ((orig_graph.nrLines() - 8) == graph.nrLines());
350                 }
351             }
352         }
353     }
354     catch (ConfigurationException& e)
355     {
356         INFO(e.what());
357         REQUIRE (false);    // force output of error and failure
358     }
359     catch (MapProviderException& dbe)
360     {
361         INFO(dbe.what());
362         REQUIRE (false);    // force output of error and failure
363
364     }
365 }
366
367
```

```
368 // BARRIER BOLLARD ////////////////////////////////////////////////////////
369
370 SCENARIO ("Building graph of Mikhailovsk with barrier bollard",
371     "[graph][r_and_c][bollard][mikhailovsk]")
372 {
373     // block on node 1706164751 on way 158421713
374     try
375     {
376         std::string orig_config_file("catchtest/testsettings/"
377             "restrictions/mikhailovsk-original.json");
378         ConfigurationReader orig_config_reader(orig_config_file);
379         Configuration orig_config;
380         orig_config_reader.fillConfiguration(orig_config);
381         PostGisProvider orig_pgp(orig_config);
382         Topology orig_topology;
383         orig_pgp.getTopology(orig_topology);
384         orig_pgp.setRestrictionsAndCosts(orig_topology);
385         GraphBuilder orig_graph(orig_topology, orig_config);
386
387         // ====================================================================
388         GIVEN ("Configuration that restricts barrier bollard ")
389         {
390             std::string config_file("catchtest/testsettings/"
391                 "restrictions/mikhailovsk-barrier_bollard.json");
392             ConfigurationReader config_reader(config_file);
393             Configuration config;
394             config_reader.fillConfiguration(config);
395
396             PostGisProvider pgp(config);
397
398             Topology topology;
399             pgp.getTopology(topology);
400             pgp.setRestrictionsAndCosts(topology);
401
402             GraphBuilder graph(topology, config);
403
404             // ................................................................
405             WHEN ("Comparing original to graph with barrier bollard")
406             {
407
408                 THEN ("there should be equally many vertices "
409                       "in original and restricted")
410                 {
411                     INFO ("  Original # Vertices: " << orig_graph.nrVertices());
412                     INFO ("Restricted # Vertices: " << graph.nrVertices());
413                     REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
414                 }
415
416                 THEN ("there should be 2 more edges "
417                       "in original than in restricted")
418                 {
419                     INFO ("  Original # Edges: " << orig_graph.nrEdges());
420                     INFO ("Restricted # Edges: " << graph.nrEdges());
421                     REQUIRE ((orig_graph.nrEdges() - 2) == graph.nrEdges());
422                 }
423
424                 THEN ("there should be 2 more nodes "
```

```
425                       "in original than in restricted")
426                 {
427                     INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
428                     INFO ("Restricted # Nodes:    " << graph.nrNodes());
429                     REQUIRE ((orig_graph.nrNodes() - 2) == graph.nrNodes());
430                 }
431 
432                 THEN ("there should be 10 more lines "
433                       "in original than in restricted")
434                 {
435                     // 4 bidirectional edges connecting = 8 lines
436                     // 2 u-turns = 2 lines
437                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
438                     INFO ("Restricted # Lines:    " << graph.nrLines());
439                     REQUIRE ((orig_graph.nrLines() - 10) == graph.nrLines());
440                 }
441             }
442         }
443     }
444     catch (ConfigurationException& e)
445     {
446         INFO(e.what());
447         REQUIRE (false);    // force output of error and failure
448     }
449     catch (MapProviderException& dbe)
450     {
451         INFO(dbe.what());
452         REQUIRE (false);    // force output of error and failure
453 
454     }
455 }
456 
457 SCENARIO ("Building graph of Partille with barrier bollard",
458     "[graph][r_and_c][bollard][partille]")
459 {
460     // block on node 249292683 on way 28050664
461     try
462     {
463         std::string orig_config_file("catchtest/testsettings/"
464             "restrictions/partille-original.json");
465         ConfigurationReader orig_config_reader(orig_config_file);
466         Configuration orig_config;
467         orig_config_reader.fillConfiguration(orig_config);
468         PostGisProvider orig_pgp(orig_config);
469         Topology orig_topology;
470         orig_pgp.getTopology(orig_topology);
471         orig_pgp.setRestrictionsAndCosts(orig_topology);
472         GraphBuilder orig_graph(orig_topology, orig_config);
473 
474         // ================================================================
475         GIVEN ("Configuration that restricts barrier bollard ")
476         {
477             std::string config_file("catchtest/testsettings/"
478                 "restrictions/partille-barrier_bollard.json");
479             ConfigurationReader config_reader(config_file);
480             Configuration config;
481             config_reader.fillConfiguration(config);
```

```
482
483             PostGisProvider pgp(config);
484
485             Topology topology;
486             pgp.getTopology(topology);
487             pgp.setRestrictionsAndCosts(topology);
488
489             GraphBuilder graph(topology, config);
490
491             // ..............................................................
492             WHEN ("Comparing original to graph with barrier bollard")
493             {
494
495                 THEN ("there should be equally many vertices "
496                       "in original and restricted")
497                 {
498                     INFO ("  Original # Vertices: " << orig_graph.nrVertices());
499                     INFO ("Restricted # Vertices: " << graph.nrVertices());
500                     REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
501                 }
502
503                 THEN ("there should be 2 more edges "
504                       "in original than in restricted")
505                 {
506                     INFO ("  Original # Edges: " << orig_graph.nrEdges());
507                     INFO ("Restricted # Edges: " << graph.nrEdges());
508                     REQUIRE ((orig_graph.nrEdges() - 2) == graph.nrEdges());
509                 }
510
511                 THEN ("there should be 2 more nodes "
512                       "in original than in restricted")
513                 {
514                     INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
515                     INFO ("Restricted # Nodes:    " << graph.nrNodes());
516                     REQUIRE ((orig_graph.nrNodes() - 2) == graph.nrNodes());
517                 }
518
519                 THEN ("there should be 8 more lines "
520                       "in original than in restricted")
521                 {
522                     // 3 bidirectional edges connecting = 6 lines
523                     // (1 cycleway = 0 lines)
524                     // 2 u-turns = 2 lines
525                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
526                     INFO ("Restricted # Lines:    " << graph.nrLines());
527                     REQUIRE ((orig_graph.nrLines() - 8) == graph.nrLines());
528                 }
529             }
530         }
531     }
532     catch (ConfigurationException& e)
533     {
534         INFO(e.what());
535         REQUIRE (false);    // force output of error and failure
536     }
537     catch (MapProviderException& dbe)
538     {
```

```
539         INFO(dbe.what());
540         REQUIRE (false);    // force output of error and failure
541
542     }
543 }
544
545 // BARRIER LIFT GATE ////////////////////////////////////////////////////////
546
547 SCENARIO ("Building graph of Mikhailovsk with barrier lift gate",
548     "[graph][r_and_c][lift_gate][mikhailovsk]")
549 {
550     // block on node 1706164751 on way 158421713 (topo edge 649)
551     try
552     {
553         std::string orig_config_file("catchtest/testsettings/"
554             "restrictions/mikhailovsk-original.json");
555         ConfigurationReader orig_config_reader(orig_config_file);
556         Configuration orig_config;
557         orig_config_reader.fillConfiguration(orig_config);
558         PostGisProvider orig_pgp(orig_config);
559         Topology orig_topology;
560         orig_pgp.getTopology(orig_topology);
561         orig_pgp.setRestrictionsAndCosts(orig_topology);
562         GraphBuilder orig_graph(orig_topology, orig_config);
563
564         // ====================================================================
565         GIVEN ("Configuration that does not restrict barrier lift gate ")
566         {
567             std::string config_file("catchtest/testsettings/"
568                 "restrictions/mikhailovsk-barrier_lift_gate.json");
569             ConfigurationReader config_reader(config_file);
570             Configuration config;
571             config_reader.fillConfiguration(config);
572
573             PostGisProvider pgp(config);
574
575             Topology topology;
576             pgp.getTopology(topology);
577             pgp.setRestrictionsAndCosts(topology);
578
579             GraphBuilder graph(topology, config);
580
581             // ................................................................
582             WHEN ("Comparing original to graph with barrier lift gate")
583             {
584
585                 THEN ("there should be equally many vertices "
586                       "in original and restricted")
587                 {
588                     INFO ("  Original # Vertices: " << orig_graph.nrVertices());
589                     INFO ("Restricted # Vertices: " << graph.nrVertices());
590                     REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
591                 }
592
593                 THEN ("there should be equally many edges "
594                       "in original as in restricted")
595                 {
```

```
596                     INFO ("  Original # Edges: " << orig_graph.nrEdges());
597                     INFO ("Restricted # Edges: " << graph.nrEdges());
598                     REQUIRE ((orig_graph.nrEdges()) == graph.nrEdges());
599                 }
600
601                 THEN ("there should be equally many nodes "
602                       "in original as in restricted")
603                 {
604                     INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
605                     INFO ("Restricted # Nodes:    " << graph.nrNodes());
606                     REQUIRE ((orig_graph.nrNodes()) == graph.nrNodes());
607                 }
608
609                 THEN ("there should be equally many lines "
610                       "in original as in restricted")
611                 {
612                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
613                     INFO ("Restricted # Lines:    " << graph.nrLines());
614                     REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
615                 }
616
617                 THEN ("there should be an extra cost of 60 on edge 649")
618                 {
619                     EdgeIdType id = 649;
620                     Cost orig_cost = orig_topology.getEdge(id).cost();
621                     Cost rest_cost = topology.getEdge(id).cost();
622                     INFO ("  Original cost:    " << orig_cost);
623                     INFO ("Restricted cost:    " << rest_cost);
624                     REQUIRE (60.0 == Approx((rest_cost - orig_cost)));
625                 }
626             }
627         }
628     }
629     catch (ConfigurationException& e)
630     {
631         INFO(e.what());
632         REQUIRE (false);    // force output of error and failure
633     }
634     catch (MapProviderException& dbe)
635     {
636         INFO(dbe.what());
637         REQUIRE (false);    // force output of error and failure
638
639     }
640 }
641
642
643 SCENARIO ("Building graph of Partille with barrier lift gate",
644     "[graph][r_and_c][lift_gate][partille]")
645 {
646     // lift gate on node 249292683 on way 28050664 (topo edge 267)
647     try
648     {
649         std::string orig_config_file("catchtest/testsettings/"
650             "restrictions/partille-original.json");
651         ConfigurationReader orig_config_reader(orig_config_file);
652         Configuration orig_config;
```

```
653         orig_config_reader.fillConfiguration(orig_config);
654         PostGisProvider orig_pgp(orig_config);
655         Topology orig_topology;
656         orig_pgp.getTopology(orig_topology);
657         orig_pgp.setRestrictionsAndCosts(orig_topology);
658         GraphBuilder orig_graph(orig_topology, orig_config);
659
660         // ================================================================
661         GIVEN ("Configuration that does not restrict barrier lift gate ")
662         {
663             std::string config_file("catchtest/testsettings/"
664                 "restrictions/partille-barrier_lift_gate.json");
665             ConfigurationReader config_reader(config_file);
666             Configuration config;
667             config_reader.fillConfiguration(config);
668
669             PostGisProvider pgp(config);
670
671             Topology topology;
672             pgp.getTopology(topology);
673             pgp.setRestrictionsAndCosts(topology);
674
675             GraphBuilder graph(topology, config);
676
677             // ..............................................................
678             WHEN ("Comparing original to graph with barrier lift gate")
679             {
680
681                 THEN ("there should be equally many vertices "
682                       "in original and restricted")
683                 {
684                     INFO ("  Original # Vertices: " << orig_graph.nrVertices());
685                     INFO ("Restricted # Vertices: " << graph.nrVertices());
686                     REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
687                 }
688
689                 THEN ("there should be equally many edges "
690                       "in original as in restricted")
691                 {
692                     INFO ("  Original # Edges: " << orig_graph.nrEdges());
693                     INFO ("Restricted # Edges: " << graph.nrEdges());
694                     REQUIRE ((orig_graph.nrEdges()) == graph.nrEdges());
695                 }
696
697                 THEN ("there should be equally many nodes "
698                       "in original as in restricted")
699                 {
700                     INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
701                     INFO ("Restricted # Nodes:    " << graph.nrNodes());
702                     REQUIRE ((orig_graph.nrNodes()) == graph.nrNodes());
703                 }
704
705                 THEN ("there should be equally many lines "
706                       "in original as in restricted")
707                 {
708                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
709                     INFO ("Restricted # Lines:    " << graph.nrLines());
```

```
710                     REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
711                 }
712
713                 THEN ("there should be an extra cost of 60 on edge 267")
714                 {
715                     EdgeIdType id = 267;
716                     Cost orig_cost = orig_topology.getEdge(id).cost();
717                     Cost rest_cost = topology.getEdge(id).cost();
718                     INFO ("  Original cost:    " << orig_cost);
719                     INFO ("Restricted cost:    " << rest_cost);
720                     REQUIRE (60.0 == Approx((rest_cost - orig_cost)));
721                 }
722             }
723         }
724     }
725     catch (ConfigurationException& e)
726     {
727         INFO(e.what());
728         REQUIRE (false);    // force output of error and failure
729     }
730     catch (MapProviderException& dbe)
731     {
732         INFO(dbe.what());
733         REQUIRE (false);    // force output of error and failure
734
735     }
736 }
737
738 // TRAFFIC LIGHTS ////////////////////////////////////////////////////////////
739
740 SCENARIO ("Building graph of Mikhailovsk with traffic signals",
741     "[graph][r_and_c][traffic_signals][mikhailovsk]")
742 {
743     // additional node on way 158421713 (topo edge id 649)
744     try
745     {
746         std::string orig_config_file("catchtest/testsettings/"
747             "restrictions/mikhailovsk-original.json");
748         ConfigurationReader orig_config_reader(orig_config_file);
749         Configuration orig_config;
750         orig_config_reader.fillConfiguration(orig_config);
751         PostGisProvider orig_pgp(orig_config);
752         Topology orig_topology;
753         orig_pgp.getTopology(orig_topology);
754         orig_pgp.setRestrictionsAndCosts(orig_topology);
755         GraphBuilder orig_graph(orig_topology, orig_config);
756
757         // ============================================================================
758         GIVEN ("Configuration that has cost for traffic signals ")
759         {
760             std::string config_file("catchtest/testsettings/"
761                 "restrictions/mikhailovsk-highway_traffic_signals.json");
762             ConfigurationReader config_reader(config_file);
763             Configuration config;
764             config_reader.fillConfiguration(config);
765
766             PostGisProvider pgp(config);
```

```
767
768             Topology topology;
769             pgp.getTopology(topology);
770             pgp.setRestrictionsAndCosts(topology);
771
772             GraphBuilder graph(topology, config);
773
774             // ..............................................................
775             WHEN ("Comparing original to graph with traffic lights")
776             {
777
778                 THEN ("there should be equally many vertices "
779                       "in original and restricted")
780                 {
781                     INFO ("  Original # Vertices: " << orig_graph.nrVertices());
782                     INFO ("Restricted # Vertices: " << graph.nrVertices());
783                     REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
784                 }
785
786                 THEN ("there should be equally many edges "
787                       "in original as in restricted")
788                 {
789                     INFO ("  Original # Edges: " << orig_graph.nrEdges());
790                     INFO ("Restricted # Edges: " << graph.nrEdges());
791                     REQUIRE ((orig_graph.nrEdges()) == graph.nrEdges());
792                 }
793
794                 THEN ("there should be equally many nodes "
795                       "in original as in restricted")
796                 {
797                     INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
798                     INFO ("Restricted # Nodes:    " << graph.nrNodes());
799                     REQUIRE ((orig_graph.nrNodes()) == graph.nrNodes());
800                 }
801
802                 THEN ("there should be equally many lines "
803                       "in original as in restricted")
804                 {
805                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
806                     INFO ("Restricted # Lines:    " << graph.nrLines());
807                     REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
808                 }
809
810                 THEN ("there should be an extra cost of 30 on edge 649")
811                 {
812                     EdgeIdType id = 649;
813                     Cost orig_cost = orig_topology.getEdge(id).cost();
814                     Cost rest_cost = topology.getEdge(id).cost();
815                     INFO ("  Original cost:    " << orig_cost);
816                     INFO ("Restricted cost:    " << rest_cost);
817                     REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
818                 }
819             }
820         }
821     }
822     catch (ConfigurationException& e)
823     {
```

```
824         INFO(e.what());
825         REQUIRE (false);    // force output of error and failure
826     }
827     catch (MapProviderException& dbe)
828     {
829         INFO(dbe.what());
830         REQUIRE (false);    // force output of error and failure
831 
832     }
833 }
834 
835 SCENARIO ("Building graph of Partille with traffic signals",
836     "[graph][r_and_c][traffic_signals][partille]")
837 {
838     // additional node on way 28050664 (topo edge id 267)
839     try
840     {
841         std::string orig_config_file("catchtest/testsettings/"
842             "restrictions/partille-original.json");
843         ConfigurationReader orig_config_reader(orig_config_file);
844         Configuration orig_config;
845         orig_config_reader.fillConfiguration(orig_config);
846         PostGisProvider orig_pgp(orig_config);
847         Topology orig_topology;
848         orig_pgp.getTopology(orig_topology);
849         orig_pgp.setRestrictionsAndCosts(orig_topology);
850         GraphBuilder orig_graph(orig_topology, orig_config);
851 
852         // ========================================================================
853         GIVEN ("Configuration that has cost for traffic signals ")
854         {
855             std::string config_file("catchtest/testsettings/"
856                 "restrictions/partille-highway_traffic_signals.json");
857             ConfigurationReader config_reader(config_file);
858             Configuration config;
859             config_reader.fillConfiguration(config);
860 
861             PostGisProvider pgp(config);
862 
863             Topology topology;
864             pgp.getTopology(topology);
865             pgp.setRestrictionsAndCosts(topology);
866 
867             GraphBuilder graph(topology, config);
868 
869             // ....................................................................
870             WHEN ("Comparing original to graph with traffic lights")
871             {
872 
873                 THEN ("there should be equally many vertices "
874                       "in original and restricted")
875                 {
876                     INFO ("  Original # Vertices: " << orig_graph.nrVertices());
877                     INFO ("Restricted # Vertices: " << graph.nrVertices());
878                     REQUIRE (orig_graph.nrVertices() == graph.nrVertices());
879                 }
880 
```

```
881                 THEN ("there should be equally many edges "
882                       "in original as in restricted")
883                 {
884                     INFO ("  Original # Edges: " << orig_graph.nrEdges());
885                     INFO ("Restricted # Edges: " << graph.nrEdges());
886                     REQUIRE ((orig_graph.nrEdges()) == graph.nrEdges());
887                 }
888
889                 THEN ("there should be equally many nodes "
890                       "in original as in restricted")
891                 {
892                     INFO ("  Original # Nodes:    " << orig_graph.nrNodes());
893                     INFO ("Restricted # Nodes:    " << graph.nrNodes());
894                     REQUIRE ((orig_graph.nrNodes()) == graph.nrNodes());
895                 }
896
897                 THEN ("there should be equally many lines "
898                       "in original as in restricted")
899                 {
900                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
901                     INFO ("Restricted # Lines:    " << graph.nrLines());
902                     REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
903                 }
904
905                 THEN ("there should be an extra cost of 30 on edge 267")
906                 {
907                     EdgeIdType id = 267;
908                     Cost orig_cost = orig_topology.getEdge(id).cost();
909                     Cost rest_cost = topology.getEdge(id).cost();
910                     INFO ("  Original cost:    " << orig_cost);
911                     INFO ("Restricted cost:    " << rest_cost);
912                     REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
913                 }
914             }
915         }
916     }
917     catch (ConfigurationException& e)
918     {
919         INFO(e.what());
920         REQUIRE (false);    // force output of error and failure
921     }
922     catch (MapProviderException& dbe)
923     {
924         INFO(dbe.what());
925         REQUIRE (false);    // force output of error and failure
926
927     }
928 }
929
930 // STOP NODE ////////////////////////////////////////////////////////////////
931
932 SCENARIO ("Building graph of Mikhailovsk with stop at node before crossing",
933     "[graph][r_and_c][stop_node][mikhailovsk]")
934 {
935     // additional node on way 158421713 (topo edge id 649)
936     try
937     {
```

```
938         std::string orig_config_file("catchtest/testsettings/"
939             "restrictions/mikhailovsk-original.json");
940         ConfigurationReader orig_config_reader(orig_config_file);
941         Configuration orig_config;
942         orig_config_reader.fillConfiguration(orig_config);
943         PostGisProvider orig_pgp(orig_config);
944         Topology orig_topology;
945         orig_pgp.getTopology(orig_topology);
946         orig_pgp.setRestrictionsAndCosts(orig_topology);
947         GraphBuilder orig_graph(orig_topology, orig_config);
948 
949         // ============================================================================
950         GIVEN ("Configuration that has cost for stops ")
951         {
952             std::string config_file("catchtest/testsettings/"
953                 "restrictions/mikhailovsk-highway_stop_node.json");
954             ConfigurationReader config_reader(config_file);
955             Configuration config;
956             config_reader.fillConfiguration(config);
957 
958             PostGisProvider pgp(config);
959 
960             Topology topology;
961             pgp.getTopology(topology);
962             pgp.setRestrictionsAndCosts(topology);
963 
964             GraphBuilder graph(topology, config);
965 
966             // ....................................................................
967             WHEN ("Comparing original to graph with stop signs")
968             {
969                 THEN ("there should be equally many lines "
970                       "in original as in restricted")
971                 {
972                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
973                     INFO ("Restricted # Lines:    " << graph.nrLines());
974                     REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
975                 }
976 
977                 THEN ("there should be an extra cost of 30 on edge 649")
978                 {
979                     EdgeIdType id = 649;
980                     Cost orig_cost = orig_topology.getEdge(id).cost();
981                     Cost rest_cost = topology.getEdge(id).cost();
982                     INFO ("  Original cost:    " << orig_cost);
983                     INFO ("Restricted cost:    " << rest_cost);
984                     REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
985                 }
986             }
987         }
988     }
989     catch (ConfigurationException& e)
990     {
991         INFO(e.what());
992         REQUIRE (false);    // force output of error and failure
993     }
994     catch (MapProviderException& dbe)
```

```
995     {
996         INFO(dbe.what());
997         REQUIRE (false);    // force output of error and failure
998 
999     }
1000 }
1001 
1002 SCENARIO ("Building graph of Partille with stop at node before crossing",
1003     "[graph][r_and_c][stop_node][partille]")
1004 {
1005     // additional node on way 28050664 (topo edge id 267)
1006     try
1007     {
1008         std::string orig_config_file("catchtest/testsettings/"
1009             "restrictions/partille-original.json");
1010         ConfigurationReader orig_config_reader(orig_config_file);
1011         Configuration orig_config;
1012         orig_config_reader.fillConfiguration(orig_config);
1013         PostGisProvider orig_pgp(orig_config);
1014         Topology orig_topology;
1015         orig_pgp.getTopology(orig_topology);
1016         orig_pgp.setRestrictionsAndCosts(orig_topology);
1017         GraphBuilder orig_graph(orig_topology, orig_config);
1018 
1019         // ============================================================
1020         GIVEN ("Configuration that has cost for traffic signals ")
1021         {
1022             std::string config_file("catchtest/testsettings/"
1023                 "restrictions/partille-highway_stop_node.json");
1024             ConfigurationReader config_reader(config_file);
1025             Configuration config;
1026             config_reader.fillConfiguration(config);
1027 
1028             PostGisProvider pgp(config);
1029 
1030             Topology topology;
1031             pgp.getTopology(topology);
1032             pgp.setRestrictionsAndCosts(topology);
1033 
1034             GraphBuilder graph(topology, config);
1035 
1036             // ........................................................
1037             WHEN ("Comparing original to graph with stop signs")
1038             {
1039                 THEN ("there should be equally many lines "
1040                       "in original as in restricted")
1041                 {
1042                     INFO ("  Original # Lines:    " << orig_graph.nrLines());
1043                     INFO ("Restricted # Lines:    " << graph.nrLines());
1044                     REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
1045                 }
1046 
1047                 THEN ("there should be an extra cost of 30 on edge 267")
1048                 {
1049                     EdgeIdType id = 267;
1050                     Cost orig_cost = orig_topology.getEdge(id).cost();
1051                     Cost rest_cost = topology.getEdge(id).cost();
```

```
1052                    INFO ("  Original cost:    " << orig_cost);
1053                    INFO ("Restricted cost:    " << rest_cost);
1054                    REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
1055                }
1056            }
1057        }
1058    }
1059    catch (ConfigurationException& e)
1060    {
1061        INFO(e.what());
1062        REQUIRE (false);    // force output of error and failure
1063    }
1064    catch (MapProviderException& dbe)
1065    {
1066        INFO(dbe.what());
1067        REQUIRE (false);    // force output of error and failure
1068
1069    }
1070 }
1071
1072 // STOP ALL ////////////////////////////////////////////////////////////////
1073
1074 SCENARIO ("Building graph of Mikhailovsk with stop for all at crossing",
1075     "[graph][r_and_c][stop_all][mikhailovsk]")
1076 {
1077     // stop at vertex 1706164758 (topo 460)
1078     // affecting edges with topo id 611, 649, 661
1079     try
1080     {
1081         std::string orig_config_file("catchtest/testsettings/"
1082             "restrictions/mikhailovsk-original.json");
1083         ConfigurationReader orig_config_reader(orig_config_file);
1084         Configuration orig_config;
1085         orig_config_reader.fillConfiguration(orig_config);
1086         PostGisProvider orig_pgp(orig_config);
1087         Topology orig_topology;
1088         orig_pgp.getTopology(orig_topology);
1089         orig_pgp.setRestrictionsAndCosts(orig_topology);
1090         GraphBuilder orig_graph(orig_topology, orig_config);
1091
1092         // ============================================================
1093         GIVEN ("Configuration that has cost for stops ")
1094         {
1095             std::string config_file("catchtest/testsettings/"
1096                 "restrictions/mikhailovsk-highway_stop_all.json");
1097             ConfigurationReader config_reader(config_file);
1098             Configuration config;
1099             config_reader.fillConfiguration(config);
1100
1101             PostGisProvider pgp(config);
1102
1103             Topology topology;
1104             pgp.getTopology(topology);
1105             pgp.setRestrictionsAndCosts(topology);
1106
1107             GraphBuilder graph(topology, config);
1108
```

```
1109            // ..........................................................
1110            WHEN ("Comparing original to graph with stop signs")
1111            {
1112                THEN ("there should be equally many lines "
1113                      "in original as in restricted")
1114                {
1115                    INFO ("  Original # Lines:    " << orig_graph.nrLines());
1116                    INFO ("Restricted # Lines:    " << graph.nrLines());
1117                    REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
1118                }
1119
1120                THEN ("there should be an extra cost of 30 on edge 611")
1121                {
1122                    EdgeIdType id = 611;
1123                    Cost orig_cost = orig_topology.getEdge(id).cost();
1124                    Cost rest_cost = topology.getEdge(id).cost();
1125                    INFO ("  Original cost:    " << orig_cost);
1126                    INFO ("Restricted cost:    " << rest_cost);
1127                    REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
1128                }
1129
1130                THEN ("there should be an extra cost of 30 on edge 649")
1131                {
1132                    EdgeIdType id = 649;
1133                    Cost orig_cost = orig_topology.getEdge(id).cost();
1134                    Cost rest_cost = topology.getEdge(id).cost();
1135                    INFO ("  Original cost:    " << orig_cost);
1136                    INFO ("Restricted cost:    " << rest_cost);
1137                    REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
1138                }
1139
1140                THEN ("there should be an extra cost of 30 on edge 661")
1141                {
1142                    EdgeIdType id = 661;
1143                    Cost orig_cost = orig_topology.getEdge(id).cost();
1144                    Cost rest_cost = topology.getEdge(id).cost();
1145                    INFO ("  Original cost:    " << orig_cost);
1146                    INFO ("Restricted cost:    " << rest_cost);
1147                    REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
1148                }
1149            }
1150        }
1151    }
1152    catch (ConfigurationException& e)
1153    {
1154        INFO(e.what());
1155        REQUIRE (false);    // force output of error and failure
1156    }
1157    catch (MapProviderException& dbe)
1158    {
1159        INFO(dbe.what());
1160        REQUIRE (false);    // force output of error and failure
1161
1162    }
1163 }
1164
1165 SCENARIO ("Building graph of Partille with stop for all at crossing",
```

```
1166        "[graph][r_and_c][stop_all][partille]")
1167    {
1168        // stop at vertex 308018343 (topo 229)
1169        // affecting edges with topo id 265, 266, 267
1170        try
1171        {
1172            std::string orig_config_file("catchtest/testsettings/"
1173                "restrictions/partille-original.json");
1174            ConfigurationReader orig_config_reader(orig_config_file);
1175            Configuration orig_config;
1176            orig_config_reader.fillConfiguration(orig_config);
1177            PostGisProvider orig_pgp(orig_config);
1178            Topology orig_topology;
1179            orig_pgp.getTopology(orig_topology);
1180            orig_pgp.setRestrictionsAndCosts(orig_topology);
1181            GraphBuilder orig_graph(orig_topology, orig_config);
1182
1183            // ========================================================================
1184            GIVEN ("Configuration that has cost for traffic signals ")
1185            {
1186                std::string config_file("catchtest/testsettings/"
1187                    "restrictions/partille-highway_stop_all.json");
1188                ConfigurationReader config_reader(config_file);
1189                Configuration config;
1190                config_reader.fillConfiguration(config);
1191
1192                PostGisProvider pgp(config);
1193
1194                Topology topology;
1195                pgp.getTopology(topology);
1196                pgp.setRestrictionsAndCosts(topology);
1197
1198                GraphBuilder graph(topology, config);
1199
1200                // ....................................................................
1201                WHEN ("Comparing original to graph with stop signs")
1202                {
1203                    THEN ("there should be equally many lines "
1204                          "in original as in restricted")
1205                    {
1206                        INFO ("  Original # Lines:    " << orig_graph.nrLines());
1207                        INFO ("Restricted # Lines:    " << graph.nrLines());
1208                        REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
1209                    }
1210
1211                    THEN ("there should be an extra cost of 30 on edge 265")
1212                    {
1213                        EdgeIdType id = 265;
1214                        Cost orig_cost = orig_topology.getEdge(id).cost();
1215                        Cost rest_cost = topology.getEdge(id).cost();
1216                        INFO ("  Original cost:    " << orig_cost);
1217                        INFO ("Restricted cost:    " << rest_cost);
1218                        REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
1219                    }
1220
1221                    THEN ("there should be an extra cost of 30 on edge 266")
1222                    {
```

```
1223                    EdgeIdType id = 266;
1224                    Cost orig_cost = orig_topology.getEdge(id).cost();
1225                    Cost rest_cost = topology.getEdge(id).cost();
1226                    INFO ("  Original cost:    " << orig_cost);
1227                    INFO ("Restricted cost:    " << rest_cost);
1228                    REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
1229                }
1230
1231                THEN ("there should be an extra cost of 30 on edge 267")
1232                {
1233                    EdgeIdType id = 267;
1234                    Cost orig_cost = orig_topology.getEdge(id).cost();
1235                    Cost rest_cost = topology.getEdge(id).cost();
1236                    INFO ("  Original cost:    " << orig_cost);
1237                    INFO ("Restricted cost:    " << rest_cost);
1238                    REQUIRE (30.0 == Approx((rest_cost - orig_cost)));
1239                }
1240            }
1241        }
1242    }
1243    catch (ConfigurationException& e)
1244    {
1245        INFO(e.what());
1246        REQUIRE (false);    // force output of error and failure
1247    }
1248    catch (MapProviderException& dbe)
1249    {
1250        INFO(dbe.what());
1251        REQUIRE (false);    // force output of error and failure
1252
1253    }
1254}
1255
1256// TRAFFIC CALMING ////////////////////////////////////////////////////////
1257
1258SCENARIO ("Building graph of Mikhailovsk with speed bump at node",
1259    "[graph][r_and_c][traffic_calming][mikhailovsk]")
1260{
1261    // additional node on way 158421713 (topo edge id 649)
1262    try
1263    {
1264        std::string orig_config_file("catchtest/testsettings/"
1265            "restrictions/mikhailovsk-original.json");
1266        ConfigurationReader orig_config_reader(orig_config_file);
1267        Configuration orig_config;
1268        orig_config_reader.fillConfiguration(orig_config);
1269        PostGisProvider orig_pgp(orig_config);
1270        Topology orig_topology;
1271        orig_pgp.getTopology(orig_topology);
1272        orig_pgp.setRestrictionsAndCosts(orig_topology);
1273        GraphBuilder orig_graph(orig_topology, orig_config);
1274
1275        // ============================================================
1276        GIVEN ("Configuration that has cost for stops ")
1277        {
1278            std::string config_file("catchtest/testsettings/"
1279                "restrictions/mikhailovsk-traffic_calming_bump.json");
```

```
1280            ConfigurationReader config_reader(config_file);
1281            Configuration config;
1282            config_reader.fillConfiguration(config);
1283
1284            PostGisProvider pgp(config);
1285
1286            Topology topology;
1287            pgp.getTopology(topology);
1288            pgp.setRestrictionsAndCosts(topology);
1289
1290            GraphBuilder graph(topology, config);
1291
1292            // ..............................................................
1293            WHEN ("Comparing original to graph with stop signs")
1294            {
1295                THEN ("there should be equally many lines "
1296                      "in original as in restricted")
1297                {
1298                    INFO ("  Original # Lines:    " << orig_graph.nrLines());
1299                    INFO ("Restricted # Lines:    " << graph.nrLines());
1300                    REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
1301                }
1302
1303                THEN ("there should be an extra cost of 10 on edge 649")
1304                {
1305                    EdgeIdType id = 649;
1306                    Cost orig_cost = orig_topology.getEdge(id).cost();
1307                    Cost rest_cost = topology.getEdge(id).cost();
1308                    INFO ("  Original cost:    " << orig_cost);
1309                    INFO ("Restricted cost:    " << rest_cost);
1310                    REQUIRE (10.0 == Approx((rest_cost - orig_cost)));
1311                }
1312            }
1313        }
1314    }
1315    catch (ConfigurationException& e)
1316    {
1317        INFO(e.what());
1318        REQUIRE (false);    // force output of error and failure
1319    }
1320    catch (MapProviderException& dbe)
1321    {
1322        INFO(dbe.what());
1323        REQUIRE (false);    // force output of error and failure
1324
1325    }
1326}
1327
1328SCENARIO ("Building graph of Partille with speed bump at node",
1329    "[graph][r_and_c][traffic_calming][partille]")
1330{
1331    // additional node on way 28050664 (topo edge id 267)
1332    try
1333    {
1334        std::string orig_config_file("catchtest/testsettings/"
1335            "restrictions/partille-original.json");
1336        ConfigurationReader orig_config_reader(orig_config_file);
```

```
1337          Configuration orig_config;
1338          orig_config_reader.fillConfiguration(orig_config);
1339          PostGisProvider orig_pgp(orig_config);
1340          Topology orig_topology;
1341          orig_pgp.getTopology(orig_topology);
1342          orig_pgp.setRestrictionsAndCosts(orig_topology);
1343          GraphBuilder orig_graph(orig_topology, orig_config);
1344
1345          // ================================================================
1346          GIVEN ("Configuration that has cost for stops ")
1347          {
1348              std::string config_file("catchtest/testsettings/"
1349                  "restrictions/partille-traffic_calming_hump.json");
1350              ConfigurationReader config_reader(config_file);
1351              Configuration config;
1352              config_reader.fillConfiguration(config);
1353
1354              PostGisProvider pgp(config);
1355
1356              Topology topology;
1357              pgp.getTopology(topology);
1358              pgp.setRestrictionsAndCosts(topology);
1359
1360              GraphBuilder graph(topology, config);
1361
1362              // ....................................................
1363              WHEN ("Comparing original to graph with stop signs")
1364              {
1365                  THEN ("there should be equally many lines "
1366                        "in original as in restricted")
1367                  {
1368                      INFO ("  Original # Lines:    " << orig_graph.nrLines());
1369                      INFO ("Restricted # Lines:    " << graph.nrLines());
1370                      REQUIRE ((orig_graph.nrLines()) == graph.nrLines());
1371                  }
1372
1373                  THEN ("there should be an extra cost of 10 on edge 267")
1374                  {
1375                      EdgeIdType id = 267;
1376                      Cost orig_cost = orig_topology.getEdge(id).cost();
1377                      Cost rest_cost = topology.getEdge(id).cost();
1378                      INFO ("  Original cost:    " << orig_cost);
1379                      INFO ("Restricted cost:    " << rest_cost);
1380                      REQUIRE (10.0 == Approx((rest_cost - orig_cost)));
1381                  }
1382              }
1383          }
1384      }
1385      catch (ConfigurationException& e)
1386      {
1387          INFO(e.what());
1388          REQUIRE (false);    // force output of error and failure
1389      }
1390      catch (MapProviderException& dbe)
1391      {
1392          INFO(dbe.what());
1393          REQUIRE (false);    // force output of error and failure
```

```
1394     }
1395 }
```

D.7.24 Topology_test.cc

```
1  /*
2   * Topology_test.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "../../catchtest/catch.hpp"
8
9  #include "../Topology.h"
10
11 SCENARIO ("Storing topology edges and vertices in Topology", "[topology]")
12 {
13     OsmIdType osm_id(std::numeric_limits<OsmIdType>::max());
14     // ------------------------------------------------------------------------
15     GIVEN ("a Topology object and data for a vertex")
16         {
17         Topology topo;
18         const VertexIdType id(1);
19         const double x(2);
20         const double y(3);
21         const Point point(x, y);
22
23         WHEN ("we try to add vertex to Topology")
24         {
25             const Vertex& r_vertex = topo.addVertex(id, point);
26
27             THEN ("we should get a reference to a TopologyVertex object")
28             {
29                 REQUIRE (r_vertex.id() == id);
30                 REQUIRE (r_vertex.point() == point);
31                 REQUIRE (r_vertex.point().x == Approx(x));
32                 REQUIRE (r_vertex.point().y == Approx(y));
33             }
34         }
35         }
36
37     // ------------------------------------------------------------------------
38     GIVEN ("a Topology object and data for 2 vertices with same id")
39     {
40         Topology topo;
41         const VertexIdType v1(1);
42         const Point p1(2,3);
43         const Point p2(4,5);
44
45         WHEN ("we try to add second vertex to Topology")
46         {
47             const Vertex& r_v1 = topo.addVertex(v1, p1);
48             const Vertex& r_v2 = topo.addVertex(v1, p2);
49
50             THEN ("we should get a reference to first TopologyVertex object")
51             {
52                 REQUIRE (r_v2 == r_v1);
```

```
53              }
54          }
55      }
56
57      // ---------------------------------------------------------------------
58      GIVEN ("a Topology object and data for two vertices and an edge")
59      {
60          Topology topo;
61
62          const VertexIdType v1(1);
63          const Point p1(2,3);
64
65          const VertexIdType v2(2);
66          const Point p2(4,5);
67
68          const EdgeIdType e1(1);
69
70          WHEN ("we try to add edge to Topology with existing vertices")
71          {
72              const Vertex& r_v1 = topo.addVertex(v1, p1);
73              const Vertex& r_v2 = topo.addVertex(v2, p2);
74
75              const Edge& r_edge = topo.addEdge(e1, osm_id, v1, v2);
76
77              THEN ("we should get a reference to a TopologyEdge object")
78              {
79                  REQUIRE (r_edge.id() == e1);
80                  REQUIRE (r_edge.sourceId() == r_v1.id());
81                  REQUIRE (r_edge.targetId() == r_v2.id());
82              }
83          }
84      }
85
86      // ---------------------------------------------------------------------
87      GIVEN ("a Topology object and data for two vertices and an edge")
88      {
89          Topology topo;
90
91          const VertexIdType v1(1);
92          const Point p1(2,3);
93
94          const VertexIdType v2(2);
95          const Point p2(4,5);
96
97          const EdgeIdType e1(1);
98
99          WHEN ("we try to add edge to Topology with non-existing vertices")
100         {
101             THEN ("we should get a TopologyException")
102                     {
103             REQUIRE_THROWS_AS(
104                 const Edge& r_edge = topo.addEdge(e1, osm_id, v1, v2),
105                 TopologyException&
106             );
107                     }
108         }
109     }
```

```
110
111     // ------------------------------------------------------------------------
112     GIVEN ("Three points and to edges for a topology")
113     {
114         Topology topology;
115         topology.addVertex(1, Point(0,0));
116         topology.addVertex(2, Point(1,2));
117         topology.addVertex(3, Point(3,4));
118         topology.addEdge(1,osm_id,1,2);
119         topology.addEdge(2,osm_id,2,3);
120
121         // ....................................................................
122         WHEN ("counting nr of edges")
123         {
124             size_t nr_edges = topology.nrEdges();
125
126             THEN ("we should get 2")
127                 {
128                 REQUIRE (nr_edges == 2);
129                 }
130         }
131
132         // ....................................................................
133         WHEN ("counting nr of vertices")
134         {
135             size_t nr_vertices = topology.nrVertices();
136
137             THEN ("we should get 3")
138             {
139                 REQUIRE (nr_vertices == 3);
140             }
141         }
142     }
143 }
```

D.7.25 TurnCostCalculator_test.cc

```
1  /* Tests for TurnCostCalclulator class.
2   *
3   * To run these tests one needs to comment the
4   * `private` label in the TurnCostCalculator.
5   *
6   * TurnCostCalculator_test.cc
7   *
8   *  @author  Jonas Bergman
9   */
10
11 #include "../../catchtest/catch.hpp"
12
13 #include "../TurnCostCalculator.h"
14 #include "../Edge.h"
15 #include "../../config/ConfigurationReader.h"
16 #include "../../config/Configuration.h"
17
18 SCENARIO ("Keeping track of costs for Turn", "[turncost]")
19 {
20     Edge source(1,1,1,1);
```

```
21      Edge target(2,2,2,2);
22
23      Edge::GeomData source_geom;
24      Edge::GeomData target_geom;
25
26      Edge::RoadData primary;
27      Edge::RoadData secondary;
28
29      primary.roadType = OsmHighway::HighwayType::PRIMARY;
30      secondary.roadType = OsmHighway::HighwayType::SECONDARY;
31
32
33      std::string config_file(
34          "catchtest/testsettings/mikh_restr_0617-testsettings.json");
35      ConfigurationReader config_reader(config_file);
36      Configuration config;
37      config_reader.fillConfiguration(config);
38
39      GIVEN ("two edges and a configuration")
40      {
41          WHEN ("asking for turn cost for turn between bearing 80 and 350")
42          {
43              source_geom.targetBearing = 80;
44              source.setGeomData(source_geom);
45              source.setSpeed(90);
46
47              target_geom.sourceBearing = 350;
48              target.setGeomData(target_geom);
49              target.setSpeed(60);
50
51              THEN ("we should get a cost")
52              {
53                  double cost =
54                      TurnCostCalculator::getTurnCost(
55                          source, target, config);
56                  INFO ("Turn cost " << cost);
57                  REQUIRE (cost > 0);
58              }
59          }
60
61          /*****************************************************************
62           * Must comment the `private` part of the TurnCostCalculator class
63           * to run the following tests.
64           *****************************************************************/
65  //        WHEN ("getting cost for lower priority source turning into"
66  //            " higher priority target")
67  //        {
68  //            source.setRoadData(secondary);
69  //            target.setRoadData(primary);
70  //
71  //            THEN ("we should get a cost")
72  //            {
73  //                double cost =
74  //                    TurnCostCalculator::giveWayToHigherRoadCategoryCost(
75  //                    source, target);
76  //                REQUIRE (cost > 0);
77  //            }
```

```
78  //        }
79  //
80  //        WHEN ("getting cost for higher priority source turning into"
81  //            " lower priority target")
82  //        {
83  //            source.setRoadData(primary);
84  //            target.setRoadData(secondary);
85  //
86  //            THEN ("we should not get a cost")
87  //            {
88  //                double cost =
89  //                    TurnCostCalculator::giveWayToHigherRoadCategoryCost(
90  //                    source, target);
91  //                REQUIRE (cost == Approx(0.0));
92  //            }
93  //        }
94  //
95  //        WHEN ("getting cost for source turning into"
96  //            " equal priority target")
97  //        {
98  //            source.setRoadData(primary);
99  //            target.setRoadData(primary);
100 //
101 //            THEN ("we should not get a cost")
102 //            {
103 //                double cost =
104 //                    TurnCostCalculator::giveWayToHigherRoadCategoryCost(
105 //                    source, target);
106 //                REQUIRE (cost == Approx(0.0));
107 //            }
108 //        }
109 //
110 //        WHEN ("asking for angle between bearing 80 and bearing 20")
111 //        {
112 //            source_geom.targetBearing = 80;
113 //            source.setGeomData(source_geom);
114 //
115 //            target_geom.sourceBearing = 20;
116 //            target.setGeomData(target_geom);
117 //
118 //            THEN ("we should get an angle of 60")
119 //            {
120 //                int angle = TurnCostCalculator::getTurnAngle(source, target);
121 //                INFO ("turn angle in 80, out 20 = " << angle);
122 //                REQUIRE (angle == 60);
123 //            }
124 //        }
125 //
126 //        WHEN ("asking for angle between bearing 80 and bearing 350")
127 //        {
128 //            source_geom.targetBearing = 80;
129 //            source.setGeomData(source_geom);
130 //
131 //            target_geom.sourceBearing = 350;
132 //            target.setGeomData(target_geom);
133 //
134 //            THEN ("we should get an angle of 90")
```

```
135 //            {
136 //                int angle = TurnCostCalculator::getTurnAngle(source, target);
137 //                INFO ("turn angle in 80, out 350 = " << angle);
138 //                REQUIRE (angle == 90);
139 //            }
140 //        }
141 //
142 //        WHEN ("asking for angle between bearing 80 and bearing 125")
143 //        {
144 //            source_geom.targetBearing = 80;
145 //            source.setGeomData(source_geom);
146 //
147 //            target_geom.sourceBearing = 125;
148 //            target.setGeomData(target_geom);
149 //
150 //            THEN ("we should get an angle of -45")
151 //            {
152 //                int angle = TurnCostCalculator::getTurnAngle(source, target);
153 //                INFO ("turn angle in 80, out 125 = " << angle);
154 //                REQUIRE (angle == -45);
155 //            }
156 //        }
157 //
158 //        WHEN ("asking for angle between bearing 80 and bearing 260")
159 //        {
160 //            source_geom.targetBearing = 80;
161 //            source.setGeomData(source_geom);
162 //
163 //            target_geom.sourceBearing = 260;
164 //            target.setGeomData(target_geom);
165 //
166 //            THEN ("we should get an angle of -180")
167 //            {
168 //                int angle = TurnCostCalculator::getTurnAngle(source, target);
169 //                INFO ("turn angle in 80, out 260 = " << angle);
170 //                REQUIRE (angle == -180);
171 //            }
172 //        }
173 //
174 //        WHEN ("asking for length penalty factor for length 4.5 at angle 35 degrees")
175 //        {
176 //            int angle = 35;
177 //
178 //            THEN ("we should get a factor of 1")
179 //            {
180 //                double len = 4.5;
181 //                double factor =
182 //                    TurnCostCalculator::calculateLengthPenaltyFactor(angle, len);
183 //                REQUIRE (factor == Approx(1.0));
184 //            }
185 //        }
186 //
187 //        WHEN ("asking for length penalty factor for length 6.0 at angle 35 degrees")
188 //        {
189 //            int angle = 35;
190 //
191 //            THEN ("we should get a factor less than 1")
```

```
192 //            {
193 //                double len = 6.0;
194 //                double factor =
195 //                    TurnCostCalculator::calculateLengthPenaltyFactor(angle, len);
196 //                REQUIRE (factor < 1.0);
197 //            }
198 //        }
199 //
200 //        WHEN ("asking for length penalty factor for length 6.0 at angle -130 degrees")
201 //        {
202 //            int angle = -130;
203 //
204 //            THEN ("we should get a factor equal to 1")
205 //            {
206 //                double len = 6.0;
207 //                double factor =
208 //                    TurnCostCalculator::calculateLengthPenaltyFactor(angle, len);
209 //                REQUIRE (factor == Approx(1.0));
210 //            }
211 //        }
212 //
213 //        WHEN ("asking for smallest speed of 20, 40, 60, 80")
214 //        {
215 //            THEN ("we should get 20")
216 //            {
217 //                Speed smallest = TurnCostCalculator::getSmallestSpeed({80,40,20,60});
218 //                REQUIRE (smallest == 20);
219 //            }
220 //        }
221 //
222 //        WHEN ("asking for angle speed between bearing 80 and 350")
223 //        {
224 //            source_geom.targetBearing = 80;
225 //            source.setGeomData(source_geom);
226 //            source.setSpeed(90);
227 //
228 //            target_geom.sourceBearing = 350;
229 //            target.setGeomData(target_geom);
230 //            target.setSpeed(60);
231 //
232 //            THEN ("we should get a speed")
233 //            {
234 //                Speed speed = TurnCostCalculator::getAngleSpeed(source, target);
235 //                INFO ("Angle speed " << speed);
236 //                REQUIRE ((speed > 0 && speed < 60));
237 //            }
238 //        }
239 //
240 //        WHEN ("asking for angle speed between bearing 80 and 260")
241 //        {
242 //            source_geom.targetBearing = 80;
243 //            source.setGeomData(source_geom);
244 //            source.setSpeed(90);
245 //
246 //            target_geom.sourceBearing = 260;
247 //            target.setGeomData(target_geom);
248 //            target.setSpeed(60);
```

```
249 //
250 //            THEN ("we should get a speed")
251 //            {
252 //                Speed speed = TurnCostCalculator::getAngleSpeed(source, target);
253 //                INFO ("Angle speed " << speed);
254 //                REQUIRE ((speed > 0 && speed < 60));
255 //            }
256 //        }
257 //
258 //        WHEN ("asking for angle speed between bearing 80 and 20")
259 //        {
260 //            source_geom.targetBearing = 80;
261 //            source.setGeomData(source_geom);
262 //            source.setSpeed(90);
263 //
264 //            target_geom.sourceBearing = 20;
265 //            target.setGeomData(target_geom);
266 //            target.setSpeed(60);
267 //
268 //            THEN ("we should get a speed")
269 //            {
270 //                Speed speed = TurnCostCalculator::getAngleSpeed(source, target);
271 //                INFO ("Angle speed " << speed);
272 //                REQUIRE ((speed > 0 && speed < 60));
273 //            }
274 //        }
275 //
276 //        WHEN ("asking for vehicle size speed between bearing 80 and 20, vehicle_length 4.5")
277 //        {
278 //            source_geom.targetBearing = 80;
279 //            source.setGeomData(source_geom);
280 //            source.setSpeed(90);
281 //
282 //            target_geom.sourceBearing = 20;
283 //            target.setGeomData(target_geom);
284 //            target.setSpeed(60);
285 //
286 //            double length {4.5};
287 //            int angle_speed {40};
288 //
289 //
290 //            THEN ("we should not get a speed reduction from angle speed")
291 //            {
292 //                Speed speed =
293 //                    TurnCostCalculator::getVehicleSizeSpeed(
294 //                        source, target, length, angle_speed);
295 //                INFO ("Vehicle size speed " << speed);
296 //                REQUIRE (speed == angle_speed);
297 //            }
298 //        }
299 //
300 //        WHEN ("asking for vehicle size speed between bearing 80 and 20, vehicle_length 8.5")
301 //        {
302 //            source_geom.targetBearing = 80;
303 //            source.setGeomData(source_geom);
304 //            source.setSpeed(90);
305 //
```

```
306 //            target_geom.sourceBearing = 20;
307 //            target.setGeomData(target_geom);
308 //            target.setSpeed(60);
309 //
310 //            double length {8.5};
311 //            int angle_speed {40};
312 //
313 //
314 //            THEN ("we should get a speed reduction from angle speed")
315 //            {
316 //                Speed speed =
317 //                    TurnCostCalculator::getVehicleSizeSpeed(
318 //                        source, target, length, angle_speed);
319 //                INFO ("Vehicle size speed " << speed);
320 //                REQUIRE (speed < angle_speed);
321 //            }
322 //        }
323 //
324 //        WHEN ("asking for turn cost for turn between bearing 80 and 20")
325 //        {
326 //            source_geom.targetBearing = 80;
327 //            source.setGeomData(source_geom);
328 //            source.setSpeed(90);
329 //
330 //            target_geom.sourceBearing = 20;
331 //            target.setGeomData(target_geom);
332 //            target.setSpeed(60);
333 //
334 //            THEN ("we should get a cost")
335 //            {
336 //                double cost =
337 //                    TurnCostCalculator::getTurnCost(
338 //                        source, target, config);
339 //                INFO ("Turn cost " << cost);
340 //                REQUIRE (cost > 0);
341 //            }
342 //        }
343     }
344 }
```

D.8 lgu

D.8.1 README.md

```
Line Graph Utility
==================

The main class in this utility.

Given a configuration file it picks a `MapProvider` and fetches a `Topology`,
↪ which is passed to the `GraphBuilder` along with the `Configuration`. The
↪ goal is to fetch a linegraph that is built according to the data found in the
↪ database and the configuration settings found in the configuration file.
```

```
A requirement for this utility was to be able to update data in the database
↪ which means this utility can also be requested to re-read the topology if
↪ there has been a change to them, or the restrictions and costs if there has
↪ been a change to them.
```

D.8.2 LineGraphUtility.h

```
/**  The class to call to request a linegraph for routing.
 *
 * #include "LineGraphUtility.h"
 *
 * @author  Jonas Bergman
 */

#ifndef LGU_LINEGRAPHUTILITY_H_
#define LGU_LINEGRAPHUTILITY_H_

// SYSTEM INCLUDES
//

// PROJECT INCLUDES
//
#include <boost/graph/copy.hpp>

// LOCAL INCLUDES
//
#include "LineGraphUtilityException.h"
#include "../config/ConfigurationReader.h"
#include "../graph/GraphBuilder.h"
#include "../mapprovider/MapProvider.h"

// FORWARD REFERENCES
//


/** A class to run the fetching of data from database, through to complete
 * weighted linegraph.
 */
class LineGraphUtility
{
public:
// LIFECYCLE

    /** Default constructor.
     */
    LineGraphUtility() = delete;

    /** Constructor.
     * @param   rFilename   The path to the configuration file.
     */
    LineGraphUtility(const std::string& rFilename);

    /** Copy constructor.
     *
     * @param from The value to copy to this object.
     */
    LineGraphUtility(const LineGraphUtility& from) = delete;
```

```
51
52
53      /** Destructor.
54       */
55      ~LineGraphUtility(void);
56
57
58  // OPERATORS
59  // OPERATIONS
60      /** Return a LineGraph
61       */
62      LineGraphType*    getLineGraph();
63
64      /** Re-read the topology if there has been a change in the database.
65       */
66      void              updateTopology();
67
68      /** Re-apply restrictions and costs on the topology fi there has been changes.
69       */
70      void              updateRestrictionsAndCosts();
71
72      /** Save the LineGraph to storage.
73       * This is a hack to be able to demo the line graph in PostGis and JOSM.
74       */
75      void              persistLineGraph();
76
77      /** Output information about # vertices, edges, nodes, lines.
78       */
79      void              printGraphInformation(
80                           std::string propmt,
81                           std::ostream& os) const;
82
83  // ACCESS
84  // INQUIRY
85
86  protected:
87  private:
88  // HELPERS
89      void    init();
90      void    initConfiguration();
91      void    initMapProvider();
92      void    initTopology();
93      void    initRestrictionsAndCosts();
94      void    buildGraph();
95
96  // ATTRIBUTES
97      const std::string&  mrSettingsfile;
98      Configuration       mConfig;
99      MapProvider*        mpMapProvider;
100     Topology            mTopology;
101     GraphBuilder*       mpGraphBuilder;
102
103 // CONSTANTS
104 };
105
106 // INLINE METHODS
107 //
```

```
108
109 // EXTERNAL REFERENCES
110 //
111
112 #endif /* LGU_LINEGRAPHUTILITY_H_ */
```

D.8.3 LineGraphUtility.cc

```
1  /*
2   * LineGraphUtility.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "../mapprovider/postgis/PostGisProvider.h"
8  #include "../mapprovider/jsontest/JsonTestProvider.h"
9
10 #include <string>
11 #include "LineGraphUtility.h"  // class implemented
12
13 ////////////////////////////// PUBLIC /////////////////////////////////////
14
15 //============================= LIFECYCLE ====================================
16 LineGraphUtility::LineGraphUtility(const std::string& rFilename)
17     : mrSettingsfile(rFilename),
18       mConfig(),
19       mpMapProvider(nullptr),
20       mTopology(),
21       mpGraphBuilder(nullptr)
22 {
23     try
24     {
25         init();
26     }
27     catch (const std::exception& e)
28     {
29         throw LineGraphUtilityException(
30             std::string("Error initializing LineGraphUtility: ") + e.what());
31     }
32 }
33
34 LineGraphUtility::~LineGraphUtility()
35 {
36     delete mpMapProvider;
37     delete mpGraphBuilder;
38 }
39
40 //============================= OPERATORS ====================================
41 //============================= OPERATIONS ===================================
42 LineGraphType*
43 LineGraphUtility::getLineGraph()
44 {
45     LineGraphType& r_orig = mpGraphBuilder->getBoostLineGraph();
46     LineGraphType* p_new = new LineGraphType();
47
48     // make a copy of the old graph into a new
49     boost::copy_graph(r_orig, *p_new);
```

```
50
51      return p_new;
52  }
53
54  void
55  LineGraphUtility::updateTopology()
56  {
57      mTopology.clearTopology();
58      initTopology();
59      initRestrictionsAndCosts();
60      buildGraph();
61  }
62
63  void
64  LineGraphUtility::updateRestrictionsAndCosts()
65  {
66      mTopology.clearEdgeCostAndRestrictions();
67      initRestrictionsAndCosts();
68      buildGraph();
69  }
70
71  void
72  LineGraphUtility::persistLineGraph()
73  {
74      try
75      {
76          mpMapProvider->persistLineGraph(*mpGraphBuilder);
77      }
78      catch(MapProviderException& mpe)
79      {
80          throw LineGraphUtilityException(mpe.what());
81      }
82  }
83
84  void
85  LineGraphUtility::printGraphInformation(
86      std::string prompt,
87      std::ostream& os) const
88  {
89      os << prompt;
90      mpGraphBuilder->printGraphInformation(os);
91  }
92  //============================== ACESS      ===================================
93  //============================== INQUIRY    ===================================
94  ////////////////////////////////// PROTECTED  ////////////////////////////////////
95
96  ////////////////////////////////// PRIVATE    ////////////////////////////////////
97  void
98  LineGraphUtility::init()
99  {
100     initConfiguration();
101     initMapProvider();
102     initTopology();
103     initRestrictionsAndCosts();
104     buildGraph();
105 }
106
```

```
107 void
108 LineGraphUtility::initConfiguration()
109 {
110     try
111     {
112         ConfigurationReader config_reader(mrSettingsfile);
113         config_reader.fillConfiguration(mConfig);
114     }
115     catch (ConfigurationException& ce)
116     {
117         delete mpMapProvider;
118         delete mpGraphBuilder;
119         throw LineGraphUtilityException(
120             std::string("LineGraphUtility:initConfiguration: ") + ce.what());
121     }
122 }
123
124 void
125 LineGraphUtility::initMapProvider()
126 {
127     try
128     {
129         const TopologyConfig& r_topo_config = mConfig.getTopologyConfig();
130         const std::string& r_provider_name = r_topo_config.providerName;
131
132         if(r_provider_name == TopologyConfig::PROVIDER_POSTGIS)
133         {
134             mpMapProvider = new PostGisProvider(mConfig);
135         }
136         else if(r_provider_name == TopologyConfig::PROVIDER_JSONTEST)
137         {
138             mpMapProvider = new JsonTestProvider(mConfig);
139         }
140         else
141         {
142             throw MapProviderException("No valid MapProvider found");
143         }
144     }
145     catch (MapProviderException& mpe)
146     {
147         delete mpMapProvider;
148         delete mpGraphBuilder;
149
150         throw LineGraphUtilityException(
151             std::string("LineGraphUtility:initMapProvider: ") + mpe.what());
152     }
153 }
154
155 void
156 LineGraphUtility::initTopology()
157 {
158     try
159     {
160         mpMapProvider->getTopology(mTopology);
161     }
162     catch (MapProviderException& mpe)
163     {
```

```
164             delete mpMapProvider;
165             delete mpGraphBuilder;
166 
167             throw LineGraphUtilityException(
168                 std::string("LineGraphUtility:initTopology ") + mpe.what());
169         }
170     }
171 
172     void
173     LineGraphUtility::initRestrictionsAndCosts()
174     {
175         try
176         {
177             mpMapProvider->setRestrictionsAndCosts(mTopology);
178         }
179         catch (MapProviderException& mpe)
180         {
181             delete mpMapProvider;
182             delete mpGraphBuilder;
183 
184             throw LineGraphUtilityException(
185                 std::string("LineGraphUtility:initRestrictionsAndCosts ") + mpe.what());
186         }
187     }
188 
189     void
190     LineGraphUtility::buildGraph()
191     {
192         try
193         {
194             delete mpGraphBuilder;
195             mpGraphBuilder = new GraphBuilder(mTopology, mConfig);
196         }
197         catch (const std::exception& e)
198         {
199             delete mpMapProvider;
200             delete mpGraphBuilder;
201 
202             throw LineGraphUtilityException(
203                 std::string("LineGraphUtility:buildGraph: ") + e.what());
204         }
205     }
```

D.8.4 LineGraphUtilityException.h

```
1   /**  Exception thrown by the 'lgu' package.
2    *
3    * #include "LineGraphUtilityException.h"
4    *
5    * @author  Jonas Bergman
6    */
7 
8   #ifndef LGU_LINEGRAPHUTILITYEXCEPTION_H_
9   #define LGU_LINEGRAPHUTILITYEXCEPTION_H_
10 
11  // SYSTEM INCLUDES
12  //
```

```
13 #include <exception>
14 #include <string>
15
16 // PROJECT INCLUDES
17 //
18
19 // LOCAL INCLUDES
20 //
21
22 // FORWARD REFERENCES
23 //
24
25 /**
26  * Exception to throw from the 'graph' package.
27  * More information of the type of exception is given in the 'what()' message.
28  */
29 class LineGraphUtilityException : public std::exception
30 {
31 public:
32 // LIFECYCLE
33     /** Default constructor.
34      */
35     LineGraphUtilityException() = delete;
36
37     /** Constructor taking a message to display.
38      *
39      * @param   message     The message to prepend when 'what()' is called.
40      */
41     LineGraphUtilityException(const std::string& rMessage) noexcept
42         : std::exception(), mMessage(rMessage)
43     {}
44
45 // OPERATORS
46 // OPERATIONS
47 // ACCESS
48 // INQUIRY
49     const char* what() const noexcept
50     { return (mMessage.c_str()); }
51
52 protected:
53 private:
54 // ATTRIBUTES
55     std::string     mMessage;
56 };
57
58 // INLINE METHODS
59 //
60
61 // EXTERNAL REFERENCES
62 //
63
64 #endif /* LGU_LINEGRAPHUTILITYEXCEPTION_H_ */
```

D.8.5 LineGraphUtility_test.cc

```
1 /*
2  * LineGraphUtility_test.cc
```

```
3   *
4   * @author  Jonas Bergman
5   */
6
7   #include "../LineGraphUtility.h"
8
9   #include <chrono>
10
11  using namespace std::chrono;
12
13
14  #include "../../catchtest/catch.hpp"
15
16
17  SCENARIO ("LineGraphUtility construction", "[lgu][construction]")
18  {
19      try
20      {
21          GIVEN ("a valid config file set up to use jsontest as provider")
22          {
23              std::string config_file(
24                  "catchtest/testsettings/mikh_restr_0617-testsettings.json");
25              //.....................................................................
26              WHEN ("creating a LineGraphUtility")
27              {
28                  THEN ("we should not receive an exception")
29                  {
30                      REQUIRE_NOTHROW(LineGraphUtility lgu(config_file));
31                  }
32              }
33          }
34      }
35      catch (LineGraphUtilityException& lgue)
36      {
37          INFO(lgue.what());
38          REQUIRE (false);    // force output of error and failure
39      }
40      catch (const std::exception& e)
41      {
42          INFO(e.what());
43          REQUIRE (false);    // force output of error and failure
44      }
45  }
46
47  SCENARIO ("LineGraphUtility operation", "[lgu][operation]")
48  {
49      try
50      {
51          GIVEN ("a valid config file")
52          {
53              std::string config_file(
54                  "catchtest/testsettings/mikh_restr_0617-testsettings.json");
55              LineGraphUtility lgu(config_file);
56
57              WHEN ("asking for a LineGraph")
58              {
59                  LineGraphType* p_lg = lgu.getLineGraph();
```

```
60
61                  THEN ("we should have a line graph")
62                  {
63                      REQUIRE (p_lg != nullptr);
64                      REQUIRE (boost::num_edges(*p_lg) > 0);
65                  }
66
67                  THEN ("we should be able to print info for lines")
68                  {
69                      int i = 0;
70                      for(auto it = boost::edges(*p_lg);
71                          (i < 10) && (it.first != it.second) ;
72                          ++it.first, ++i)
73                      {
74                          const auto& line = *(it.first);
75                          NodeIdType lg_source_id =
76                              boost::get(&LineGraphLine::lgSourceNodeId, *p_lg, line);
77                          NodeIdType lg_target_id =
78                              boost::get(&LineGraphLine::lgTargetNodeId, *p_lg, line);
79                          VertexIdType topo_via_vertex_id =
80                              boost::get(&LineGraphLine::topoViaVertexId, *p_lg, line);
81                          double cost =
82                              boost::get(&LineGraphLine::cost, *p_lg, line);
83
84                          INFO ("LINE:   lg_source_id: " << lg_source_id
85                              << ", lg_target_id: " << lg_target_id
86                              << ", topo_via_vertex_id: " << topo_via_vertex_id
87                              << ", cost: " << cost << "\n");
88                          REQUIRE (true);
89
90                      }
91                  }
92
93                  THEN ("we can try to persist line graph")
94                  {
95                      try
96                      {
97                          lgu.persistLineGraph();
98                          INFO ("Persisted line graph");
99                          REQUIRE (true);
100                     }
101                     catch (LineGraphUtilityException& lgue)
102                     {
103                         INFO (lgue.what());
104                         REQUIRE (false);
105                     }
106                 }
107
108                 delete p_lg;
109             }
110
111             WHEN ("asking to update restrictions and costs")
112             {
113                 lgu.updateRestrictionsAndCosts();
114                 LineGraphType* p_lg {nullptr};
115                 p_lg = lgu.getLineGraph();
116
```

```
117                 THEN ("we should still be able to have a line graph")
118                 {
119                     REQUIRE (p_lg != nullptr);
120                     REQUIRE (boost::num_edges(*p_lg) > 0);
121                 }
122 
123                 delete p_lg;
124             }
125 
126             WHEN ("asking to update topology")
127             {
128                 lgu.updateTopology();
129                 LineGraphType* p_lg {nullptr};
130                 p_lg = lgu.getLineGraph();
131 
132                 THEN ("we should still be able to have a line graph")
133                 {
134                     REQUIRE (p_lg != nullptr);
135                     REQUIRE (boost::num_edges(*p_lg) > 0);
136                 }
137 
138                 delete p_lg;
139             }
140         }
141     }
142     catch (LineGraphUtilityException& lgue)
143     {
144         INFO(lgue.what());
145         REQUIRE (false);    // force output of error and failure
146     }
147     catch (const std::exception& e)
148     {
149         INFO(e.what());
150         REQUIRE (false);    // force output of error and failure
151     }
152 }
153 
154 
155 SCENARIO ("LineGraphUtility timing", "[lgu][timing]")
156 {
157     try
158     {
159         GIVEN ("a valid config file for Mikhailovsk without temporary topology")
160         {
161             std::string config_file(
162                 "catchtest/testsettings/mikhailovsk-original.json");
163 
164             WHEN ("when timing request for a LineGraph")
165             {
166                 // start timing
167                 std::chrono::high_resolution_clock::time_point t1 =
168                     std::chrono::high_resolution_clock::now();
169 
170                 LineGraphType* p_lg;
171 
172                 int nr_rounds = 10;
173                 for(int i = 0; i < nr_rounds; ++i)
```

```
174                 {
175                     LineGraphUtility lgu(config_file);
176                     p_lg = lgu.getLineGraph();
177                     delete p_lg;
178                     p_lg = nullptr;
179                 }
180 
181                 // end timing;
182                 std::chrono::high_resolution_clock::time_point t2 =
183                     std::chrono::high_resolution_clock::now();
184 
185                 auto duration =
186                     std::chrono::duration_cast<std::chrono::microseconds>
187                         ( t2 - t1 ).count();
188 
189 
190                 THEN ("we should have an average timing")
191                 {
192                     INFO ("Average duration over " << nr_rounds << " rounds for "
193                         "Mikhailovsk without temporary topology: "
194                         << duration / nr_rounds << " microseconds");
195                     REQUIRE (true);
196                 }
197 
198                 delete p_lg;
199             }
200         }
201 
202         GIVEN ("a valid config file for Mikhailovsk WITH temporary topology")
203         {
204             std::string config_file(
205                 "catchtest/testsettings/mikhailovsk-original-temp.json");
206 
207             WHEN ("when timing request for a LineGraph")
208             {
209                 // start timing
210                 std::chrono::high_resolution_clock::time_point t1 =
211                     std::chrono::high_resolution_clock::now();
212 
213                 LineGraphType* p_lg;
214 
215                 int nr_rounds = 10;
216                 for(int i = 0; i < nr_rounds; ++i)
217                 {
218                     LineGraphUtility lgu(config_file);
219                     p_lg = lgu.getLineGraph();
220                     delete p_lg;
221                     p_lg = nullptr;
222                 }
223 
224                 // end timing;
225                 std::chrono::high_resolution_clock::time_point t2 =
226                     std::chrono::high_resolution_clock::now();
227 
228                 auto duration =
229                     std::chrono::duration_cast<std::chrono::microseconds>
230                         ( t2 - t1 ).count();
```

```
231
232
233                 THEN ("we should have an average timing")
234                 {
235                     INFO ("Average duration over " << nr_rounds << " rounds for "
236                         "Mikhailovsk WITH temporary topology: "
237                         << duration / nr_rounds << " microseconds");
238                     REQUIRE (true);
239                 }
240
241                 delete p_lg;
242             }
243         }
244
245
246         GIVEN ("a valid config file for Partille without temporary topology")
247         {
248             std::string config_file(
249                 "catchtest/testsettings/partille-original.json");
250
251             WHEN ("when timing request for a LineGraph")
252             {
253                 // start timing
254                 std::chrono::high_resolution_clock::time_point t1 =
255                     std::chrono::high_resolution_clock::now();
256
257                 LineGraphType* p_lg;
258
259                 int nr_rounds = 10;
260                 for(int i = 0; i < nr_rounds; ++i)
261                 {
262                     LineGraphUtility lgu(config_file);
263                     p_lg = lgu.getLineGraph();
264                     delete p_lg;
265                     p_lg = nullptr;
266                 }
267
268                 // end timing;
269                 std::chrono::high_resolution_clock::time_point t2 =
270                     std::chrono::high_resolution_clock::now();
271
272                 auto duration =
273                     std::chrono::duration_cast<std::chrono::microseconds>
274                         ( t2 - t1 ).count();
275
276
277                 THEN ("we should have an average timing")
278                 {
279                     INFO ("Average duration over " << nr_rounds << " rounds for "
280                         "Partille without temporary topology: "
281                         << duration / nr_rounds << " microseconds");
282                     REQUIRE (true);
283                 }
284
285                 delete p_lg;
286             }
287         }
```

```
288
289         GIVEN ("a valid config file for Partille WITH temporary topology")
290         {
291             std::string config_file(
292                 "catchtest/testsettings/partille-original-temp.json");
293
294             WHEN ("when timing request for a LineGraph")
295             {
296                 // start timing
297                 std::chrono::high_resolution_clock::time_point t1 =
298                     std::chrono::high_resolution_clock::now();
299
300                 LineGraphType* p_lg;
301
302                 int nr_rounds = 10;
303                 for(int i = 0; i < nr_rounds; ++i)
304                 {
305                     LineGraphUtility lgu(config_file);
306                     p_lg = lgu.getLineGraph();
307                     delete p_lg;
308                     p_lg = nullptr;
309                 }
310
311                 // end timing;
312                 std::chrono::high_resolution_clock::time_point t2 =
313                     std::chrono::high_resolution_clock::now();
314
315                 auto duration =
316                     std::chrono::duration_cast<std::chrono::microseconds>
317                         ( t2 - t1 ).count();
318
319                 THEN ("we should have an average timing")
320                 {
321                     INFO ("Average duration over " << nr_rounds << " rounds for "
322                         "Partille WITH temporary topology: "
323                         << duration / nr_rounds << " microseconds");
324                     REQUIRE (true);
325                 }
326
327                 delete p_lg;
328             }
329         }
330
331     }
332     catch (LineGraphUtilityException& lgue)
333     {
334         INFO(lgue.what());
335         REQUIRE (false);    // force output of error and failure
336     }
337     catch (const std::exception& e)
338     {
339         INFO(e.what());
340         REQUIRE (false);    // force output of error and failure
341     }
342 }
343
344 SCENARIO ("LineGraphUtility size and order", "[lgu][print_size]")
```

```
345 {
346     try
347     {
348         GIVEN ("a valid config file for Mikhailovsk")
349         {
350             std::string config_file(
351                 "catchtest/testsettings/mikhailovsk-original.json");
352 
353             WHEN ("asking for size and order of graphs")
354             {
355                 LineGraphUtility lgu(config_file);
356                 THEN ("we should get a print out of sizes")
357                 {
358                     lgu.printGraphInformation("Mikhailovsk: ", std::cout);
359                     REQUIRE(true);
360                 }
361             }
362         }
363 
364         GIVEN ("a valid config file for Partille")
365         {
366             std::string config_file(
367                 "catchtest/testsettings/partille-original.json");
368 
369             WHEN ("asking for size and order of graphs")
370             {
371                 LineGraphUtility lgu(config_file);
372                 THEN ("we should get a print out of sizes")
373                 {
374                     lgu.printGraphInformation("Partille: ", std::cout);
375                     REQUIRE(true);
376                 }
377             }
378         }
379     }
380     catch (LineGraphUtilityException& lgue)
381     {
382         INFO(lgue.what());
383         REQUIRE (false);    // force output of error and failure
384     }
385     catch (const std::exception& e)
386     {
387         INFO(e.what());
388         REQUIRE (false);    // force output of error and failure
389     }
390 }
```

D.9 mapprovider

D.9.1 README.md

```
MapProvider
===========

The `mapprovider` package exists to implement different classes that can provide
↪  access to OpenStreetMap data.
```

```
#### Background
There exists several solutions to import OpenStreetMap data into a database, and
 ↪  the different solutions all creates different schemas and tables. To keep the
 ↪  flexibility to change how we get the OSM data, the `mapprovider` exists to
 ↪  provide an abstract interface that providers must implement.

#### `jsontest`
The `JsonTestProvider` is a small map provider that was implemented to be able to
 ↪  read in a small set of well-known edges and vertices (such as the [pgRouting
 ↪  sample
 ↪  data](http://docs.pgrouting.org/dev/doc/src/developer/sampledata.html)), to
 ↪  be used under development of the `Graph` class.

#### `postgis`
The `PostGisProvider` exists for working with topologies built with the
 ↪  `postgis_topology` extension. This is the course taken during development of
 ↪  the `LineGraphUtility` so far. (Using `pgRouting` seemed to be useful only
 ↪  for building topology and not to get access to the other map data, and
 ↪  `osm2po` is not open source. But with the `MapProvider` interface it is
 ↪  possible to implement another if desirable.)

The `PostGisProvider` uses classes `TopologyQueries` and `RestrictionQueries` for
 ↪  querying the database.

### Exceptions
Each subpackage throws its own exception: `MapProviderException`, and
 ↪  `PostGisProviderException`.
```

D.9.2 MapProvider.h

```
1  /**  Abstract base class giving the interface for different sources of
2   * topology map data.
3   *
4   * #include "MapProvider.h"
5   *
6   * @author  Jonas Bergman
7   */
8
9  #ifndef MAPPROVIDER_MAPPROVIDER_H_
10 #define MAPPROVIDER_MAPPROVIDER_H_
11
12 // SYSTEM INCLUDES
13 //
14 #include <map>
15
16 // PROJECT INCLUDES
17 //
18
19 // LOCAL INCLUDES
20 //
21 #include "MapProviderException.h"
22 #include "../graph/Topology.h"
23 #include "../config/Configuration.h"
24 #include "../graph/Edge.h"
25 #include "../graph/GraphBuilder.h"
26 #include "../graph/Vertex.h"
27
```

```
28  // FORWARD REFERENCES
29  //
30
31  /**
32   * Interface for getting map data from file or database.
33   */
34  class MapProvider
35  {
36  public:
37  // LIFECYCLE
38
39      /** Default constructor.
40       */
41      MapProvider() = delete;
42
43      /** Constructor.
44       * Construct a MapProvider based on the configurations given.
45       */
46      MapProvider(const Configuration& rConfig)
47          : mBuildTempTopology(false), mrConfig(rConfig)
48      {}
49
50      /** Copy constructor.
51       *
52       * @param from The value to copy to this object.
53       */
54      MapProvider(const MapProvider& from) = delete;
55
56
57      /** Destructor.
58       */
59      virtual ~MapProvider(void) {}
60
61
62  // OPERATORS
63  // OPERATIONS
64      /** Fill the topology with data from the MapProvider.
65       * @param   rTopology   The Topology to fill with data.
66       * @throws  MapProviderException, TopologyException
67       */
68      virtual void    getTopology(Topology& rTopology) = 0;
69
70      /** Read tags that might impose restrictions and costs and add them to
71       * the edges in the topology.
72       * @param   rTopology       The Topology with edges to get updated.
73       * @throws  MapProviderException, RestrictionsException
74       */
75      virtual void    setRestrictionsAndCosts(Topology& rTopology) = 0;
76
77      /** Save the line graph to persistent storage or throw exception if not
78       * implemented.
79       * @param   rGraph  The GraphBuilder with the LineGraph and topology to save.
80       * @throws  MapProviderException
81       */
82      virtual void    persistLineGraph(const GraphBuilder& rGraph) = 0;
83
84  // ACCESS
```

```
85  // INQUIRY
86
87  protected:
88      bool                 mBuildTempTopology;
89      const Configuration& mrConfig;
90  private:
91  };
92
93  // INLINE METHODS
94  //
95
96  // EXTERNAL REFERENCES
97  //
98
99  #endif /* MAPPROVIDER_MAPPROVIDER_H_ */
```

D.9.3 MapProviderException.h

```
1   /**  Exception thrown by the MapProvider package.
2    *
3    * #include "MapProviderException.h"
4    *
5    * @author  Jonas Bergman
6    */
7
8   #ifndef MAPPROVIDER_MAPPROVIDEREXCEPTION_H_
9   #define MAPPROVIDER_MAPPROVIDEREXCEPTION_H_
10
11  // SYSTEM INCLUDES
12  //
13  #include <exception>
14  #include <string>
15
16  // PROJECT INCLUDES
17  //
18
19  // LOCAL INCLUDES
20  //
21
22  // FORWARD REFERENCES
23  //
24
25  /**
26   * Exception to throw from the 'MapProvider' package.
27   * More information of the type of exception is given in the 'what()' message.
28   */
29  class MapProviderException : public std::exception
30  {
31  public:
32  // LIFECYCLE
33
34      /** Default constructor.
35       */
36      MapProviderException() = delete;
37
38      /** Constructor taking a message to display.
39       *
```

```
40      * @param    message        The message to prepend when 'what()' is called.
41      */
42     MapProviderException(const std::string& rMessage) noexcept
43         : std::exception(), mMessage(rMessage)
44     {}
45
46 // OPERATORS
47 // OPERATIONS
48 // ACCESS
49 // INQUIRY
50     const char* what() const noexcept
51     { return mMessage.c_str(); }
52
53 protected:
54 private:
55 // ATTRIBUTES
56     std::string        mMessage;
57 };
58
59 // INLINE METHODS
60 //
61
62 // EXTERNAL REFERENCES
63 //
64
65 #endif /* MAPPROVIDER_MAPPROVIDEREXCEPTION_H_ */
```

D.9.4 MapProvider_test.cc

```
1  /*
2   * MapProvider_test.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "../MapProvider.h"
8
9  #include <string>
10 #include <map>
11
12 #include "../../catchtest/catch.hpp"
13 #include "../../config/ConfigurationReader.h"
14 #include "../../config/Configuration.h"
15 #include "../../config/DatabaseConfig.h"
16 #include "../postgis/PostGisProvider.h"
17 #include "../../graph/Edge.h"
18 #include "../../graph/Vertex.h"
19
20
21 SCENARIO ("MapProvider queries", "[mp-query]")
22 {
23     try
24     {
25         std::string config_file("catchtest/testsettings/testsettings.json");
26         ConfigurationReader config_reader(config_file);
27
28         GIVEN ("a valid database configuration structure and "
```

```
29                  "name to existing topology")
30          {
31              Configuration config;
32              config_reader.fillConfiguration(config);
33              const DatabaseConfig& db_config = config.getDatabaseConfig();
34              std::string topo_name("test");
35
36              MapProvider* p_mp(nullptr);
37              if(config.getTopologyConfig().providerName ==
38                  TopologyConfig::PROVIDER_POSTGIS)
39              {
40                  p_mp = new PostGisProvider(config);
41              }
42
43              REQUIRE (p_mp != nullptr);
44
45              // ..........................................................
46              WHEN ("we try to fetch topology")
47              {
48                  Topology topology;
49                  p_mp->getTopology(topology);
50
51                  THEN ("we should receive topology vertices and edges")
52                  {
53                      REQUIRE (topology.nrVertices() > 0);
54                      REQUIRE (topology.nrEdges() > 0);
55                  }
56              }
57              delete p_mp;
58          }
59      }
60      catch (ConfigurationException& e)
61      {
62          INFO(e.what());
63          REQUIRE (false);    // force output of error and failure
64      }
65      catch (MapProviderException& dbe)
66      {
67          INFO(dbe.what());
68          REQUIRE (false);    // force output of error and failure
69
70      }
71  }
```

D.10 jsontest

D.10.1 README.md

```
JsonTest
========

This package exists for the sole purpose of testing in the initial stages of
↪ development, so that we easily could read in a small set of well-known edges
↪ and vertices.

But the test is now commented out, as it has not been updated to keep up with
↪ advances in the development.
```

Format of test file in json is:

```json
    {
        "vertices": [
            [1,2,0],
            [2,2,1]
        ],
        "edges": [
            [1,1,2,0]
        ]
    }
```

where each row in `vertices` are `[id,x,y]` and each row in `edges` are `[id, source vertex id, target vertex id, direction]`. Values for `direction` is `0 = BOTH`, `1 = FROM_TO`, `2 = TO_FROM`.

D.10.2 JsonTestProvider.h

```cpp
/**  Read in sample topology from a json file.
 *
 * #include "JsonTestProvider.h"
 *
 * @author  Jonas Bergman
 */
#ifndef MAPPROVIDER_JSONTEST_JSONTESTPROVIDER_H_
#define MAPPROVIDER_JSONTEST_JSONTESTPROVIDER_H_

// SYSTEM INCLUDES
//

// PROJECT INCLUDES
//

// LOCAL INCLUDES
//
#include "../MapProvider.h"

// FORWARD REFERENCES
//

/**
 * Interface for getting map data from file or database.
 */
class JsonTestProvider : public MapProvider
{
public:
// LIFECYCLE

    /** Default constructor.
     */
    JsonTestProvider() = delete;

    /** Constructor.
```

```
36      * Construct a MapProvider based on the configurations given.
37      */
38     JsonTestProvider(const Configuration& rConfig);
39
40     /** Copy constructor.
41      *
42      * @param from The value to copy to this object.
43      */
44     JsonTestProvider(const JsonTestProvider& from) = delete;
45
46
47     /** Destructor.
48      */
49     virtual ~JsonTestProvider(void);
50
51
52 // OPERATORS
53 // OPERATIONS
54     virtual void    getTopology(Topology& rTopology);
55
56     virtual void    setRestrictionsAndCosts(Topology& rTopology);
57
58     virtual void    persistLineGraph(const GraphBuilder& rGraph);
59
60 // ACCESS
61 // INQUIRY
62
63 protected:
64 private:
65 };
66
67 // INLINE METHODS
68 //
69
70 // EXTERNAL REFERENCES
71 //
72
73 #endif /* MAPPROVIDER_JSONTEST_JSONTESTPROVIDER_H_ */
```

D.10.3 JsonTestProvider.cc

```
1  /*
2   * JsonTestProvider.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "JsonTestProvider.h"  // class implemented
8  #include "../../graph/Vertex.h"
9  #include "../../graph/Edge.h"
10
11 #include <string>
12 #include <boost/property_tree/ptree.hpp>
13 #include <boost/property_tree/json_parser.hpp>
14
15
16
```

```
17  ////////////////////////////// PUBLIC //////////////////////////////////////
18
19  //============================= LIFECYCLE ====================================
20
21  JsonTestProvider::JsonTestProvider(const Configuration& rConfig)
22      : MapProvider(rConfig)
23  {
24  }
25
26  JsonTestProvider::~JsonTestProvider()
27  {
28  }
29
30  //============================= OPERATORS ====================================
31
32  //============================= OPERATIONS ===================================
33  void
34  JsonTestProvider::getTopology(Topology& rTopology)
35  {
36      using namespace boost::property_tree;
37
38      const std::string& filename = mrConfig.getTopologyConfig().testFile;
39      ptree              pt;
40
41      try
42      {
43          read_json(filename, pt);
44
45          // vertices
46          int v_row[3];
47          for(auto& row : pt.get_child("vertices"))
48          {
49              int i = 0;
50              for(auto& item : row.second)
51              {
52                  v_row[i] = item.second.get_value<int>();
53                  ++i;
54              }
55              rTopology.addVertex(v_row[0], Point(v_row[1], v_row[2]));
56          }
57
58          // edges
59          int e_row[4];
60          for(auto& row : pt.get_child("edges"))
61          {
62              int i = 0;
63              for(auto& item : row.second)
64              {
65                  e_row[i] = item.second.get_value<int>();
66                  ++i;
67              }
68              Edge::DirectionType direction;
69              OsmIdType osm_id(std::numeric_limits<OsmIdType>::max());
70              switch(e_row[3])
71              {
72                  case 0:
73                      direction = Edge::DirectionType::BOTH; break;
```

```
74                  case 1:
75                      direction = Edge::DirectionType::FROM_TO; break;
76                  case 2:
77                      direction = Edge::DirectionType::TO_FROM; break;
78                  default:
79                      direction = Edge::DirectionType::BOTH;
80              }
81              Edge& e = rTopology.addEdge(e_row[0], osm_id, e_row[1], e_row[2]);
82              Edge::RoadData rd;
83              rd.direction = direction;
84              e.setRoadData(rd);
85          }
86      }
87      catch (boost::property_tree::ptree_error& e)
88      {
89          throw TopologyException("Could not read file " + filename);
90      }
91  }
92
93  void
94  JsonTestProvider::setRestrictionsAndCosts(Topology& rTopology)
95  {
96      //none
97  }
98
99  void
100 JsonTestProvider::persistLineGraph(const GraphBuilder& rGraph)
101 {
102     throw MapProviderException("JsonTestProvider has not "
103         "implemented persisting a Line graph");
104 }
105
106 //============================== ACESS      ====================================
107 //============================== INQUIRY    ====================================
108 ////////////////////////////////// PROTECTED  ////////////////////////////////////
109
110 ////////////////////////////////// PRIVATE    ////////////////////////////////////
```

D.10.4 JsonTestProvider_test.cc

```
1  ///*
2  // * JsonTestProvider_test.cc
3  // *
4  // * @author  Jonas Bergman
5  // */
6  //
7  //#include "../JsonTestProvider.h"
8  //
9  //#include <string>
10 //#include <vector>
11 //
12 //#include "../../../catchtest/catch.hpp"
13 //#include "../../../config/ConfigurationReader.h"
14 //#include "../../../config/DatabaseConfig.h"
15 //#include "../../../graph/Edge.h"
16 //#include "../../../graph/Vertex.h"
17 //#include "../../../graph/Graph.h"
```

```
18 //
19 //SCENARIO ("JsonTest topology handling", "[jsontest]")
20 //{
21 //    try
22 //    {
23 //        // ============================================================
24 //        GIVEN ("a valid configuration structure with a jsontest filename")
25 //        {
26 //            std::string config_file("mapprovider/jsontest"
27 //                "/catchtest/jsontest-settings.json");
28 //            ConfigurationReader config_reader(config_file);
29 //            Configuration config;
30 //            config_reader.fillConfiguration(config);
31 //            JsonTestProvider* p_jt(nullptr);
32 //
33 //            // ........................................................
34 //            WHEN ("we try to create topology")
35 //            {
36 //                THEN ("we should not receive an exception")
37 //                {
38 //                    REQUIRE_NOTHROW ( p_jt = new JsonTestProvider(config));
39 //                }
40 //            }
41 //
42 //            // ........................................................
43 //            WHEN ("we try to fetch topology ")
44 //            {
45 //                Topology topology;
46 //                JsonTestProvider jtp(config);
47 //
48 //                // . . . . . . . . . . . . . . . . . . . . . . . . . . .
49 //                THEN ("we should not receive an exception")
50 //                {
51 //                    REQUIRE_NOTHROW (jtp.getTopology(topology));
52 //                }
53 //            }
54 //            // ........................................................
55 //            WHEN ("using topology")
56 //            {
57 //                Topology topology;
58 //                JsonTestProvider jtp(config);
59 //                jtp.getTopology(topology);
60 //
61 //                size_t nr_vertices = 13;
62 //                size_t nr_edges = 16;
63 //
64 //                Configuration config;
65 //
66 //                // . . . . . . . . . . . . . . . . . . . . . . . . . . .
67 //                THEN ("we should have the right number of edges and vertices"
68 //                    " in the topology")
69 //                {
70 //                    REQUIRE (topology.nrVertices() == nr_vertices);
71 //                    REQUIRE (topology.nrEdges() == nr_edges);
72 //                }
73 //
74 //                // . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
//                THEN ("we should be able to create a graph from topology")
//                {
//                    REQUIRE_NOTHROW (Graph graph(topology, config));
//                }
//
//                // . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
//                THEN ("we should be able to create a graph from topology"
//                    " and print out the graph")
//                {
//                    Graph graph(topology, config);
//                    INFO (graph);
//                    REQUIRE (true);
//                }
//            }
//            delete p_jt;
//        }
//
//    }
//    catch (ConfigurationException& e) {
//        INFO(e.what());
//        REQUIRE (false);    // force output of error and failure
//    }
//}
//
//
```

D.10.5 jsontest-settings.json

```
{
    "database":
    {
        "host":         "127.0.0.1",
        "port":          5432,
        "username":     "tester",
        "password":     "tester",
        "database":     "mikh_style"
    },

    "topology":
    {
        "provider":       "jsontest",

        "postgis":
        {

            "topo_name":          "test",
            "roads_prefix":       "highways",
            "schema_prefix":      "topo",
            "build": {
                "temp_topo_name": "epoch_ms",
                "srid":            900913,
                "tolerance":       1.0
            },
            "edge":
            {
                "table":       "edge_data",
                "id_col":      "edge_id",
```

```
                "source_col": "start_node",
                "target_col": "end_node",
                "geom_col":   "geom"
            },
            "vertex":
            {
                "table":      "node",
                "id_col":     "node_id",
                "geom_col":   "geom"
            }
        },

        "pgrouting":
        {
        },

        "jsontest":
        {
            "test_file": "mapprovider/jsontest/catchtest/test-topology.json"
        }
    },

    "vehicle":
    {
        "category":   "motorcar",
        "motorcar":
        {
           "height":  1.6,
           "length":  4.5,
           "width":   1.9,
           "weight":  2.0,
           "maxspeed": 200
        }
    },

    "cost":
    {
       "default_speed":
       {
           "motorway":      {"high": 110, "low":  90},
           "motorway_link": {"high":  90, "low":  90},
           "trunk":         {"high":  90, "low":  60},
           "trunk_link":    {"high":  90, "low":  60},
           "primary":       {"high":  90, "low":  60},
           "primary_link":  {"high":  90, "low":  60},
           "secondary":     {"high":  90, "low":  60},
           "secondary_link":{"high":  90, "low":  60},
           "tertiary":      {"high":  90, "low":  60},
           "tertiary_link": {"high":  90, "low":  60},
           "unclassified":  {"high":  90, "low":  60},
           "residential":   {"high":  90, "low":  60},
           "living_street": {"high":  20, "low":  20}
       }
    }
}
```

D.10.6 test-topology.json

```
{
    "vertices": [
        [1,2,0],
        [2,2,1],
        [3,3,1],
        [4,4,1],
        [5,0,2],
        [6,1,2],
        [7,2,2],
        [8,3,2],
        [9,4,2],
        [10,2,3],
        [11,3,3],
        [12,4,3],
        [13,2,4]
    ],
    "edges": [
        [1,1,2,0],
        [2,2,3,1],
        [3,3,4,1],
        [4,2,7,0],
        [5,3,8,2],
        [6,5,6,0],
        [7,6,7,0],
        [8,7,8,0],
        [9,8,9,0],
        [10,7,10,0],
        [11,8,11,2],
        [12,10,11,2],
        [13,11,12,2],
        [14,10,13,0],
        [15,9,12,0],
        [16,4,9,0]
    ]
}
```

D.11 postgis

D.11.1 README.md

```
PostGisProvider
===============

This is a concrete class that implements the MapProviders interface. It fetches
↪  map data from a PostGis database where the OSM data has been imported with
↪  `osm2pgsql`, and topology has been built with the `postgis_topology`
↪  extension, converts it to valid `Vertex` and `Edge` types and stores them in
↪  a `Topology`.

The handling of the queries are factored out in different static classes:
- `CostQueries` for handling costs.
- `RestrictionQueries` for handling restrictions.
- `TopologyQueries` for handling topology related stuff.
- `LineGraphSaveQueries` for persisting a LineGraph back to the database.
```

D.11.2 PostGisProvider.h

```
/**  Handle connections with PostGis database to get map data.
 *
 * #include "PostGisProvider.h"
 *
 * @author  Jonas Bergman
 */

#ifndef MAPPROVIDER_POSTGIS_POSTGISPROVIDER_H_
#define MAPPROVIDER_POSTGIS_POSTGISPROVIDER_H_

// SYSTEM INCLUDES
//
#include <algorithm>
#include <sstream>
#include <string>
#include <vector>

// PROJECT INCLUDES
//
#include <pqxx/pqxx>        // link with -lpqxx -lpq
#include <boost/algorithm/string.hpp>

// LOCAL INCLUDES
//
#include "CostQueries.h"
#include "RestrictionQueries.h"
#include "TopologyQueries.h"
#include "../MapProvider.h"
#include "../MapProviderException.h"
#include "../../config/DatabaseConfig.h"
#include "../../graph/Edge.h"
#include "../../graph/Topology.h"
#include "../../graph/Vertex.h"
#include "../../graph/Speed.h"
#include "../../osm/OsmAccess.h"
#include "../../osm/OsmHighway.h"
#include "../../osm/OsmVehicle.h"
#include "../../util/TimeToStringMaker.h"
#include "LineGraphSaveQueries.h"

// FORWARD REFERENCES
//

/**
 * A class to handle the reading of data from the PostGis database.
 * The configurations for the connection is given.
 */
class PostGisProvider : public MapProvider
{
public:
// LIFECYCLE

    /** Default constructor.
     */
    PostGisProvider() = delete;

```

```
57      /** Constructor.
58       * Establish connection to database.
59       *
60       * @param        rConfig                          The configuration.
61       * @throws        MapProviderException    If connection could not be established.
62       */
63      PostGisProvider(const Configuration& rConfig);
64
65      /** Destructor.
66       * Close connection to database
67       */
68      virtual ~PostGisProvider();
69
70
71  // OPERATORS
72  // OPERATIONS
73      virtual void    getTopology(Topology& rTopology);
74
75      virtual void    setRestrictionsAndCosts(Topology& rTopology);
76
77      virtual void    persistLineGraph(const GraphBuilder& rGraph);
78
79  // INQUIRY
80
81  protected:
82
83  private:
84  // HELPERS
85
86      /** Get edges from database.
87       * @param   rEdgeResult   Result of db query for Edges.
88       * @throws        MapProviderException
89       */
90      void    getTopologyEdges(pqxx::result& rEdgeResult);
91
92      /** Add edges to topology.
93       * @param   rEdgeResult    Result of db query for edges.
94       * @param   rTopology      Topology to fill with edges.
95       * @throws  TopologyException
96       */
97      void    addEdgeResultToTopology(const pqxx::result& rEdgeResult,
98                                      Topology& rTopology);
99
100     /** Get vertices from database.
101      * @throws        MapProviderException
102      */
103     void    getTopologyVertices(pqxx::result& rVertexResult);
104
105     /** Add vertices to topology.
106      * @throws  TopologyException
107      */
108     void    addVertexResultToTopology(
109                 const pqxx::result& rResult,
110                 Topology&           rTopology);
111
112     // Helpers for constructor
113     /** Set the base name for the topology, either a string from config
```

```
114         * or a timestamp.
115         */
116        void    setTopoBaseName(std::string& rTopoBaseName);
117
118        /** Build a PostGIS topology with name given in constructor.
119         * @param        srid               The SRID for the projection to use
120         * @param        tolerance        The distance to look for merging vertices, unit of srid.
121         * @throws        MapProviderException
122         */
123        void        buildTopology(int srid, double tolerance);
124
125        /** Remove PostGIS topology (tables and schema) from the database.
126         * @throws        MapProviderException
127         */
128        void        removeTopology();
129
130
131        // Restriction helpers ------------------------------------------------
132
133        /** Add restrictions to edges.
134         * @param   rTopology  Adding EdgeRestricion to Edges in topology.
135         * @throw   MapProviderException
136         */
137        void    addEdgeRestrictions(Topology& rTopology);
138
139        /** Get VehicleProperty restrictions
140         * Helper for 'getEdgeRestrictions()'
141         * @param   rResult     Store the result of query in here.
142         * @throw   MapProviderException
143         */
144        void    getVehiclePropertyEdgeRestrictions(pqxx::result& rResult);
145
146        /** Add the result of the query for vehicle properties to Edge's restrictions.
147         * Helper for 'getEdgeRestrictions()'
148         * @param   rResult         The results of the query
149         * @param   rTopology       Update affected edges in the topology.
150         * @throw   MapProviderException
151         */
152        void    addVehiclePropertyRestrictionsToEdge(
153                    const pqxx::result&    rResult,
154                    Topology&              rTopology);
155
156        /** Get Access restrictions to edge.
157         * Helper for 'getEdgeRestrictions()'
158         * @param   rResult     Store the result of query in here.
159         * @throw   MapProviderException
160         */
161        void    getAccessRestrictions(pqxx::result& rResult);
162
163        /** Add the result of the query for Access to restrictions.
164         * Helper for 'getEdgeRestrictions()'
165         * @param   rResult         The results of the query
166         * @param   rTopology       Update affected edges in the topology.
167         * @throw   MapProviderException
168         */
169        void    addAccessRestrictionsToEdge(
170                    const pqxx::result&    rResult,
```

```
171                 Topology&               rTopology);
172
173     /** Get Turning restrictions for traveling from edge.
174      * Helper for 'getEdgeRestrictions()'.
175      * Turning restrictions are relations and not easily handled with
176      * osm2pgsql. Therefore we must use 'slim' mode when converting OSM to
177      * PostGis, and use the table 'planet_osm_rels' and column 'tags' to look
178      * for a 'restriction'. If we find one we have to parse the 'members'
179      * column ourselves.
180      * @param   rResult     Store the result of query in here.
181      * @throw   MapProviderException
182      */
183     void    getTurningRestrictions(pqxx::result& rResult);
184
185     /** Add the result of the query for Turning restrictions.
186      * Helper for 'getEdgeRestrictions()'
187      * @param   rResult         The results of the query
188      * @param   rTopology       Update affected edges in the topology.
189      * @throw   MapProviderException
190      */
191     void    addTurningRestrictionsToEdge(
192                 const pqxx::result&     rResult,
193                 Topology&               rTopology);
194
195     /** Get restrictions defined at points but applicable to edges,
196      * such as barriers and railway crossings.
197      * @param   rResult     Store the result of query in here.
198      * @throw   MapProviderException
199      */
200     void    getEdgePointRestrictions(pqxx::result& rResult);
201
202     /** Add the result of the query for Point restrictions on Edges .
203      * Helper for 'getEdgeRestrictions()'
204      * @param   rResult         The results of the query
205      * @param   rTopology       Update affected edges in the topology.
206      * @throw   MapProviderException
207      */
208     void    addPointRestrictionsToEdge(
209                 const pqxx::result&     rResult,
210                 Topology&               rTopology);
211
212     // Costs ---------------------------------------------------------------
213     /** Add costs to the edge.
214      * @param   The Topology with Edges to add cost to.
215      */
216     void    addEdgeCosts(Topology& rTopology);
217
218     /** Get costs for the travel time along an edge.
219      * The length is constant in the topology but we need to find out if
220      * there is a max speed restriction or if there is a bad surface.
221      * If no such restrictions are in the database then the default speed
222      * for the road category is used.
223      * @param   rResult     Store the result of the query here.
224      * @throw   MapProviderException
225      */
226     void    getTravelTimeCosts(pqxx::result& rResult);
227
```

```
228     /** Add costs for travel time along the edge.
229      * First set the speed of those with explicit restrictions in database,
230      * then set the default speed for those without explicit speeds.
231      * @param   rResult     The results of the query.
232      * @param   rTopology   The topology with edges to set cost for.
233      * @throw   MapProviderException
234      */
235     void    addTravelTimeCosts(const pqxx::result& rResult, Topology& rTopology);
236
237     /** Get other costs for the edge other than speed and barriers, those
238      * include slowdown at stop and yield signs, zebra crossings, railway
239      * crossings, bus stops, speed bumps, traffic lights...
240      * @param   rResult     Store the result of the query here.
241      */
242     void    getOtherEdgeCosts(pqxx::result& rResult);
243
244     /** Add costs for speed bumps and such to affected edges.
245      * @param   rResult     The results of the query.
246      * @param   rTopology   The topology with edges to set cost for.
247      * @throw   MapProviderException
248      */
249     void    addOtherCosts(const pqxx::result& rResult, Topology& rTopology);
250
251     // LineGraph persistence ------------------------------------------------
252     /** Set up the schema and tables needed to persist the line graph.
253      */
254     void    setUpSchemaAndTables();
255
256     /** Create a new schema in the database
257      * @throw   MapProviderException
258      */
259     void    createLineGraphSchema();
260
261     /** Create the needed tables in the database
262      * @throw   MapProviderException
263      */
264     void    createLineGraphTables();
265
266     /** Insert the data in the database.
267      * @param   rGraph      The graph with data.
268      * @throw   MapProviderException
269      */
270     void    insertData(const GraphBuilder& rGraph);
271
272     /** Prepare the LineGraph data for inserting into the database.
273      * @param   rTrans      The transaction to operate within.
274      * @param   rGraph      The graph data
275      * @throw   MapProviderException
276      */
277     void    prepareLineGraphData(
278                 pqxx::transaction_base& rTrans,
279                 const GraphBuilder&           rGraph);
280
281     // Generic helpers to clean up the code some ----------------------------
282     /** Check that the connection with the database is up, or throw exception.
283      * @throw   MapProviderException
284      */
```

```
285     void    testConnection();
286 
287 // ATTRIBUTES
288     const Configuration&    mConfig;
289     const DatabaseConfig&   mDbConfig;
290     const TopologyConfig&   mTopoConfig;
291     pqxx::connection        mConnection;
292     std::string             mOsmEdgeTable;
293     std::string             mPointTableName;
294     std::string             mSchemaName;
295     std::string             mTopoEdgeTable;
296     std::string             mEdgeIdCol;
297     std::string             mSourceCol;
298     std::string             mTargetCol;
299     std::string             mEdgeGeomCol;
300     std::string             mTopoVertexTable;
301     std::string             mVertexIdCol;
302     std::string             mVertexGeomCol;
303     std::string             mLineGraphSchema;
304     std::string             mLineGraphNodeTable;
305     std::string             mLineGraphLineTable;
306 
307 // CONSTANTS
308 };
309 
310 // INLINE METHODS
311 //
312 
313 // EXTERNAL REFERENCES
314 //
315 
316 #endif /* MAPPROVIDER_POSTGIS_POSTGISPROVIDER_H_ */
```

D.11.3 PostGisProvider.cc

```
1  /*
2   * PostGisProvider.cc
3   *
4   * @author  Jonas Bergman
5   */
6 
7  #include "PostGisProvider.h"  // class implemented
8  #include "TopologyQueries.h"
9  #include "RestrictionQueries.h"
10 #include "CostQueries.h"
11 
12 #include "../../graph/Edge.h"
13 #include "../../osm/OsmId.h"
14 #include "../../graph/EdgeRestriction.h"
15 #include "../../graph/EdgeCost.h"
16 
17 
18 
19 ////////////////////////////// PUBLIC //////////////////////////////////////
20 
21 //============================ LIFECYCLE ====================================
22 PostGisProvider::PostGisProvider(const Configuration& rConfig)
```

```
23  try
24      : MapProvider(rConfig),
25        mConfig(rConfig),
26        mDbConfig(rConfig.getDatabaseConfig()),
27        mTopoConfig(rConfig.getTopologyConfig()),
28        mConnection(mDbConfig.getConnectionString())
29  {
30      try
31      {
32          testConnection();
33
34          std::string topoBaseName;
35          setTopoBaseName(topoBaseName);
36
37          if(topoBaseName == "")
38          {
39              throw MapProviderException("No topology specified.");
40          }
41
42          pqxx::nontransaction nt(mConnection);
43          mOsmEdgeTable       = nt.esc(mTopoConfig.roadsPrefix + "_" +
44                                    topoBaseName);
45          mPointTableName     = nt.esc("planet_osm_point");
46          mSchemaName         = nt.esc(mTopoConfig.topologySchemaPrefix + "_" +
47                                    topoBaseName);
48          mTopoEdgeTable      = nt.esc(mSchemaName + "." +
49                                    mTopoConfig.edgeTableName);
50          mEdgeIdCol          = nt.esc(mSchemaName + "." +
51                                    mTopoConfig.edgeIdColumnName);
52          mSourceCol          = nt.esc(mSchemaName + "." +
53                                    mTopoConfig.sourceColumnName);
54          mTargetCol          = nt.esc(mSchemaName + "." +
55                                    mTopoConfig.targetColumnName);
56          mEdgeGeomCol        = nt.esc(mSchemaName + "." +
57                                    mTopoConfig.edgeGeomColumnName);
58          mTopoVertexTable    = nt.esc(mSchemaName + "." +
59                                    mTopoConfig.vertexTableName);
60          mVertexIdCol        = nt.esc(mSchemaName + "." +
61                                    mTopoConfig.vertexIdColumnName);
62          mVertexGeomCol      = nt.esc(mSchemaName + "." +
63                                    mTopoConfig.vertexGeomColumnName);
64          mLineGraphSchema    = nt.esc("line_graph_generated");
65          mLineGraphNodeTable = nt.esc(mLineGraphSchema + ".node");
66          mLineGraphLineTable = nt.esc(mLineGraphSchema + ".line");
67          nt.abort();
68
69          if(mBuildTempTopology)
70          {
71              buildTopology(mTopoConfig.srid, mTopoConfig.tolerance);
72          }
73      }
74      catch(const std::exception& e)
75      {
76          throw MapProviderException(
77              std::string("PostGisProvider:ctor(in): ") + e.what());
78      }
79  }
```

```
80  // catch error in initializer list (opening connection)
81  catch(const std::exception& e)
82  {
83          throw MapProviderException(
84              std::string("PostGisProvider:ctor(out): ") + e.what());
85  }
86
87  PostGisProvider::~PostGisProvider()
88  {
89      try
90      {
91          if(mBuildTempTopology)
92          {
93              removeTopology();
94          }
95          if(mConnection.is_open())
96          {
97              mConnection.disconnect();
98          }
99      }
100     catch(const std::exception& e)
101     {
102         throw MapProviderException(
103             std::string("PostGisProvider:dtor: ") + e.what());
104     }
105 }
106
107 //============================== OPERATORS ===================================
108
109 //============================== OPERATIONS ==================================
110 void
111 PostGisProvider::getTopology(Topology& rTopology)
112 {
113     pqxx::result vertex_result;
114     getTopologyVertices(vertex_result);
115     addVertexResultToTopology(vertex_result, rTopology);
116
117     pqxx::result edge_result;
118     getTopologyEdges(edge_result);
119     addEdgeResultToTopology(edge_result, rTopology);
120 }
121
122 void
123 PostGisProvider::setRestrictionsAndCosts(Topology& rTopology)
124 {
125     addEdgeRestrictions(rTopology);
126     addEdgeCosts(rTopology);
127 }
128
129 void
130 PostGisProvider::persistLineGraph(const GraphBuilder& rGraph)
131 {
132     setUpSchemaAndTables();
133     insertData(rGraph);
134 }
135
136 //============================== ACESS       ==================================
```

```
137 //============================ INQUIRY    ==================================
138 ////////////////////////////// PROTECTED  //////////////////////////////////
139 ////////////////////////////// PRIVATE    //////////////////////////////////
140 void
141 PostGisProvider::getTopologyVertices(pqxx::result& rVertexResult)
142 {
143     try
144     {
145         testConnection();
146 
147         // NON-TRANSACTION START
148         pqxx::nontransaction transaction(mConnection);
149 
150         TopologyQueries::getTopologyVertices(
151             transaction,
152             rVertexResult,
153             mTopoVertexTable);
154     }
155     catch(const std::exception& e)
156     {
157         throw MapProviderException(
158             std::string("PostGisProvider:getTopologyVertices: ") + e.what());
159     }
160 }
161 
162 void
163 PostGisProvider::addVertexResultToTopology(
164     const pqxx::result& rResult,
165     Topology&           rTopology)
166 {
167     TopologyQueries::addVertexResultToTopology(rResult, rTopology);
168 }
169 
170 
171 void
172 PostGisProvider::getTopologyEdges(pqxx::result& rEdgeResult)
173 {
174     try
175     {
176         testConnection();
177 
178         // NON-TRANSACTION START
179         pqxx::nontransaction transaction(mConnection);
180 
181         TopologyQueries::getTopologyEdges(
182             transaction,
183             rEdgeResult,
184             mTopoEdgeTable,
185             mSchemaName,
186             mOsmEdgeTable);
187     }
188     catch(const std::exception& e)
189     {
190         throw MapProviderException(
191             std::string("PostGisProvider:getTopoEdges: ") + e.what());
192     }
193 }
```

```
194
195 void
196 PostGisProvider::addEdgeResultToTopology(
197     const pqxx::result& rResult,
198     Topology&           rTopology)
199 {
200     TopologyQueries::addEdgeResultToTopology(rResult, rTopology);
201 }
202
203 void
204 PostGisProvider::buildTopology(int srid, double tolerance)
205 {
206     try
207     {
208         testConnection();
209
210         // TRANSACTION START
211         pqxx::work transaction(mConnection);
212
213         try
214         {
215             TopologyQueries::installPostgisTopology(transaction);
216             TopologyQueries::setSearchPath(transaction);
217             TopologyQueries::createTemporaryTable(transaction, mOsmEdgeTable);
218             TopologyQueries::createTemporarySchema(
219                 transaction, mSchemaName, srid);
220             TopologyQueries::addTopoGeometryColumn(
221                 transaction, mSchemaName, mOsmEdgeTable);
222             TopologyQueries::fillTopoGeometryColumn(
223                 transaction, mSchemaName, mOsmEdgeTable, tolerance);
224
225             // TRANSACTION END
226             transaction.commit();
227         }
228         catch (const std::exception& e)
229         {
230             transaction.abort();
231             throw e;
232         }
233     }
234     catch(const std::exception& e)
235     {
236         throw MapProviderException(
237             std::string("PostGisProvider:buildTopology: ") + e.what());
238     }
239 }
240
241
242 void
243 PostGisProvider::removeTopology()
244 {
245     try
246     {
247         testConnection();
248
249         // TRANSACTION START
250         pqxx::work transaction(mConnection);
```

```
251
252             try
253             {
254                 TopologyQueries::dropTemporaryTable(transaction, mOsmEdgeTable);
255                 TopologyQueries::dropTemporarySchema(transaction, mSchemaName);
256                 TopologyQueries::deleteTemporaryLayerRecord(transaction, mOsmEdgeTable);
257                 TopologyQueries::deleteTemporaryTopoRecord(transaction, mSchemaName);
258
259                 // TRANSACTION END
260                 transaction.commit();
261             }
262             catch (const std::exception& e)
263             {
264                 transaction.abort();
265                 throw e;
266             }
267         }
268         catch(const std::exception& e)
269         {
270             throw MapProviderException(std::string(
271                 "PostGisProvider:removeTopology: ") + e.what());
272         }
273     }
274
275     void
276     PostGisProvider::setTopoBaseName(std::string& rTopoBaseName)
277     {
278         if(mTopoConfig.tempTopoName == TopologyConfig::TEMP_TOPO_NAMEBASE)
279         {
280             rTopoBaseName = TimeToStringMaker::getEpochMsTimeString();
281             mBuildTempTopology = true;
282         }
283         else
284         {
285             rTopoBaseName = mTopoConfig.topoName;
286         }
287     }
288
289     // Restrictions ------------------------------------------------------------
290     void
291     PostGisProvider::addEdgeRestrictions(Topology& rTopology)
292     {
293         pqxx::result result;
294
295         getVehiclePropertyEdgeRestrictions(result);
296         addVehiclePropertyRestrictionsToEdge(result, rTopology);
297
298         result.clear();
299         getAccessRestrictions(result);
300         addAccessRestrictionsToEdge(result, rTopology);
301
302         result.clear();
303         getTurningRestrictions(result);
304         addTurningRestrictionsToEdge(result, rTopology);
305
306         result.clear();
307         getEdgePointRestrictions(result);
```

```
308     addPointRestrictionsToEdge(result, rTopology);
309 }
310
311 void
312 PostGisProvider::getVehiclePropertyEdgeRestrictions(pqxx::result& rResult)
313 {
314     try
315     {
316         testConnection();
317
318         // NON-TRANSACTION START
319         pqxx::nontransaction transaction(mConnection);
320
321         RestrictionQueries::getVehiclePropertyEdgeRestrictions(
322             transaction,
323             rResult,
324             mTopoEdgeTable,
325             mOsmEdgeTable,
326             mSchemaName
327         );
328     }
329     catch(const std::exception& e)
330     {
331         throw MapProviderException(
332             std::string("PostGisProvider:getVehiclePropertyEdgeRestrictions: ")
333                         + e.what());
334     }
335 }
336
337 void
338 PostGisProvider::addVehiclePropertyRestrictionsToEdge(
339     const pqxx::result& rResult,
340     Topology&           rTopology)
341 {
342     RestrictionQueries::addVehiclePropertyRestrictionsToEdge(rResult, rTopology);
343 }
344
345 void
346 PostGisProvider::getAccessRestrictions(pqxx::result& rResult)
347 {
348     try
349     {
350         testConnection();
351
352         // NON-TRANSACTION START
353         pqxx::nontransaction transaction(mConnection);
354
355         RestrictionQueries::getAccessRestrictions(
356             transaction,
357             rResult,
358             mTopoEdgeTable,
359             mOsmEdgeTable,
360             mSchemaName);
361     }
362     catch(const std::exception& e)
363     {
364         throw MapProviderException(
```

```
365             std::string("PostGisProvider:getAccessRestrictions: ")
366         + e.what());
367     }
368 }
369
370 void
371 PostGisProvider::addAccessRestrictionsToEdge(
372     const pqxx::result& rResult,
373     Topology&           rTopology)
374 {
375     RestrictionQueries::addAccessRestrictionsToEdge(rResult, rTopology, mConfig);
376 }
377
378 void
379 PostGisProvider::getTurningRestrictions(pqxx::result& rResult)
380 {
381     try
382     {
383         testConnection();
384
385         // TRANSACTION START
386         pqxx::nontransaction transaction(mConnection);
387
388         try
389         {
390             RestrictionQueries::dropCreateTurningRestrictionsTable(transaction);
391             RestrictionQueries::identifyTurningRestrictions(
392                 transaction,
393                 mOsmEdgeTable,
394                 mTopoEdgeTable);
395             RestrictionQueries::getTurningRestrictions(transaction, rResult);
396         }
397         catch (const std::exception& e)
398         {
399             transaction.abort();
400             throw e;
401         }
402     }
403     catch(const std::exception& e)
404     {
405         throw MapProviderException(
406             std::string("PostGisProvider:getTurningRestrictions: ")
407         + e.what());
408     }
409 }
410
411 void
412 PostGisProvider::addTurningRestrictionsToEdge(
413     const pqxx::result&     rResult,
414     Topology&               rTopology)
415 {
416     RestrictionQueries::addTurningRestrictionsToEdge(rResult, rTopology);
417 }
418
419 void
420 PostGisProvider::getEdgePointRestrictions(pqxx::result& rResult)
421 {
```

```
422     try
423     {
424         testConnection();
425 
426         // NON-TRANSACTION START
427         pqxx::nontransaction transaction(mConnection);
428 
429         RestrictionQueries::getEdgePointRestrictions(
430             transaction,
431             rResult,
432             mPointTableName,
433             mTopoEdgeTable,
434             mOsmEdgeTable,
435             mSchemaName);
436     }
437     catch(const std::exception& e)
438     {
439         throw MapProviderException(
440             std::string("PostGisProvider:getEdgePointRestrictions: ") + e.what());
441     }
442 }
443 
444 void
445 PostGisProvider::addPointRestrictionsToEdge(
446     const pqxx::result&     rResult,
447     Topology&               rTopology)
448 {
449     RestrictionQueries::addPointRestrictionsToEdge(rResult, rTopology, mConfig);
450 }
451 
452 // Costs ---------------------------------------------------------------------
453 void
454 PostGisProvider::addEdgeCosts(Topology& rTopology)
455 {
456     pqxx::result result;
457 
458     getTravelTimeCosts(result);
459     addTravelTimeCosts(result, rTopology);
460 
461     // barrier costs are added while looking for restrictions
462 
463     result.clear();
464     getOtherEdgeCosts(result);
465     addOtherCosts(result, rTopology);
466 }
467 
468 void
469 PostGisProvider::getTravelTimeCosts(pqxx::result& rResult)
470 {
471     try
472     {
473         testConnection();
474 
475         // NON-TRANSACTION START
476         pqxx::nontransaction transaction(mConnection);
477 
478         CostQueries::getTravelTimeEdgeCosts(
```

```
479             transaction,
480             rResult,
481             mTopoEdgeTable,
482             mOsmEdgeTable,
483             mSchemaName
484         );
485     }
486     catch(const std::exception& e)
487     {
488         throw MapProviderException(
489             std::string("PostGisProvider:getTravelTimCost: ")
490                         + e.what());
491     }
492 }
493
494 void
495 PostGisProvider::addTravelTimeCosts(
496     const pqxx::result& rResult,
497     Topology&           rTopology)
498 {
499     CostQueries::addTravelTimeCosts(rResult, rTopology, mConfig);
500 }
501
502 void
503 PostGisProvider::getOtherEdgeCosts(pqxx::result& rResult)
504 {
505     try
506     {
507         testConnection();
508
509         // NON-TRANSACTION START
510         pqxx::nontransaction transaction(mConnection);
511
512         CostQueries::getOtherCosts(
513             transaction,
514             rResult,
515             mPointTableName,
516             mTopoEdgeTable,
517             mOsmEdgeTable,
518             mSchemaName);
519     }
520     catch(const std::exception& e)
521     {
522         throw MapProviderException(
523             std::string("PostGisProvider:getOtherEdgeCosts: ") + e.what());
524     }
525 }
526
527 void
528 PostGisProvider::addOtherCosts(
529     const pqxx::result&     rResult,
530     Topology&               rTopology)
531 {
532     CostQueries::addOtherCosts(rResult, rTopology, mConfig);
533 }
534
535 //  LineGraph persistence ------------------------------------------------
```

```
536 void
537 PostGisProvider::setUpSchemaAndTables()
538 {
539     createLineGraphSchema();
540     createLineGraphTables();
541 }
542
543 void
544 PostGisProvider::createLineGraphSchema()
545 {
546     try
547     {
548         testConnection();
549
550         // NON-TRANSACTION START
551         pqxx::nontransaction transaction(mConnection);
552
553         LineGraphSaveQueries::dropCreateSchema(transaction, mLineGraphSchema);
554     }
555     catch(const std::exception& e)
556     {
557         throw MapProviderException(
558             std::string("PostGisProvider:createLineGraphSchema: ") + e.what());
559     }
560 }
561
562 void
563 PostGisProvider::createLineGraphTables()
564 {
565     try
566     {
567         testConnection();
568
569         // NON-TRANSACTION START
570         pqxx::nontransaction transaction(mConnection);
571
572         LineGraphSaveQueries::dropCreateLineTable(transaction, mLineGraphLineTable);
573         LineGraphSaveQueries::dropCreateNodeTable(transaction, mLineGraphNodeTable);
574     }
575     catch(const std::exception& e)
576     {
577         throw MapProviderException(
578             std::string("PostGisProvider:createLineGraphTables: ") + e.what());
579     }
580 }
581
582 void
583 PostGisProvider::insertData(const GraphBuilder& rGraph)
584 {
585     try
586     {
587         testConnection();
588
589         pqxx::work transaction(mConnection);
590
591         try
592         {
```

```
593             prepareLineGraphData(transaction, rGraph);
594 
595             // TRANSACTION END
596             transaction.commit();
597         }
598         catch (const std::exception& e)
599         {
600             transaction.abort();
601             throw e;
602         }
603     }
604     catch(const std::exception& e)
605     {
606         throw MapProviderException(
607             std::string("PostGisProvider:insertData: ") + e.what());
608     }
609 }
610 
611 void
612 PostGisProvider::prepareLineGraphData(
613     pqxx::transaction_base& rTrans,
614     const GraphBuilder&            rGraph)
615 {
616     const LineGraphType& rLineGraph = rGraph.getBoostLineGraph();
617     const Topology&      rTopology  = rGraph.getTopology();
618 
619     for(auto line_it = boost::edges(rLineGraph);
620         line_it.first != line_it.second;
621         ++line_it.first)
622     {
623         const LineType& line = *(line_it.first);
624 
625         NodeIdType source_node_id = rLineGraph[line].lgSourceNodeId;
626         NodeIdType target_node_id = rLineGraph[line].lgTargetNodeId;
627 
628         Cost cost = rLineGraph[line].cost;
629 
630         const NodeType& source_node = rGraph.getLineGraphNode(source_node_id);
631         const NodeType& target_node = rGraph.getLineGraphNode(target_node_id);
632 
633         EdgeIdType source_edge_id = rLineGraph[source_node].topoEdgeId;
634         EdgeIdType target_edge_id = rLineGraph[target_node].topoEdgeId;
635 
636 
637         const Edge& sourceEdge = rTopology.getEdge(source_edge_id);
638         const Edge& targetEdge = rTopology.getEdge(target_edge_id);
639 
640         const Point& sourcePoint = sourceEdge.geomData().centerPoint;
641         const Point& targetPoint = targetEdge.geomData().centerPoint;
642 
643         std::string sourceWKT = "POINT(" + std::to_string(sourcePoint.x) + " "
644                                          + std::to_string(sourcePoint.y) + ")";
645         std::string targetWKT = "POINT(" + std::to_string(targetPoint.x) + " "
646                                          + std::to_string(targetPoint.y) + ")";
647         std::string lineWKT = "LINESTRING(" + std::to_string(sourcePoint.x) + " " +
648                                               std::to_string(sourcePoint.y) + ", " +
649                                               std::to_string(targetPoint.x) + " " +
```

```
650                                                 std::to_string(targetPoint.y) + ")";
651
652         LineGraphSaveQueries::insertNode(
653             rTrans,
654             mLineGraphNodeTable,
655             source_edge_id,
656             sourceWKT);
657
658         LineGraphSaveQueries::insertNode(
659             rTrans,
660             mLineGraphNodeTable,
661             target_edge_id,
662             targetWKT);
663
664         LineGraphSaveQueries::insertLine(
665             rTrans,
666             mLineGraphLineTable,
667             cost,
668             lineWKT);
669     }
670 }
671
672 void
673 PostGisProvider::testConnection()
674 {
675     if(!mConnection.is_open())
676     {
677         throw MapProviderException(
678             std::string("Could not open ") + mDbConfig.database);
679     }
680 }
```

D.11.4 CostQueries.h

```
1  /**  Queries for PostGisProvider to find costs.
2   *
3   * #include "CostQueries.h"
4   *
5   * @author  Jonas Bergman
6   */
7  #ifndef MAPPROVIDER_POSTGIS_COSTQUERIES_H_
8  #define MAPPROVIDER_POSTGIS_COSTQUERIES_H_
9
10 // SYSTEM INCLUDES
11 //
12 #include <string>
13 #include <sstream>
14 #include <vector>
15
16 // PROJECT INCLUDES
17 //
18 #include <boost/algorithm/string.hpp>
19 #include <pqxx/pqxx>
20
21 // LOCAL INCLUDES
22 //
23 #include "../../osm/OsmHighway.h"
```

```
24 #include "../../graph/Edge.h"
25 #include "../../graph/EdgeRestriction.h"
26 #include "../../graph/Topology.h"
27 #include "../../graph/Vertex.h"
28 #include "../MapProviderException.h"
29
30 /** Class for holding static queries about costs,
31  * needed by the PostGisProvider.
32  */
33 class CostQueries
34 {
35 public:
36 // TYPES
37
38     struct TravelTimeCostResult
39     {
40         enum Columns
41         {
42             EDGE_ID,
43             ELEMENT_ID,
44             MAXSPEED,
45             SURFACE
46         };
47     };
48
49     struct OtherCostResult
50     {
51         enum Columns
52         {
53             OSM_ID,
54             HIGHWAY,
55             RAILWAY,
56             PUBLIC_TRANSPORT,
57             TRAFFIC_CALMING,
58             EDGE_ID
59         };
60     };
61
62 // LIFECYCLE
63     CostQueries() = delete;
64     CostQueries(const CostQueries& from) = delete;
65     ~CostQueries() = default;
66
67 // OPERATORS
68 // OPERATIONS
69     /** Query for costs for travel time, meaning we need to find the speed.
70      * @param   rTrans          Transaction to perform query in.
71      * @param   rResult         Store the result of query here.
72      * @param   rTopoEdgeTable  Name of table with topology edges.
73      * @param   rOsmEdgeTable   Name of table with OSM edges.
74      * @param   rSchemaName     Name of the topology schema.
75      * @throw   pqxx::pqxx_exception
76      */
77     static void     getTravelTimeEdgeCosts(
78                         pqxx::transaction_base& rTrans,
79                         pqxx::result&           rResult,
80                         const std::string&      rTopoEdgeTable,
```

```
81                      const std::string&      rOsmEdgeTable,
82                      const std::string&      rSchemaName);
83
84      /** Add costs for travel time along the edge.
85       * First set the speed of those with explicit restrictions in database,
86       * then set the default speed for those without explicit speeds.
87       * @param   rResult     The results of the query.
88       * @param   rTopology   The topology with edges to set cost for.
89       * @param   rConfig         Configuration
90       * @throw   MapProviderException
91       */
92      static void     addTravelTimeCosts(
93                      const pqxx::result&     rResult,
94                      Topology&               rTopology,
95                      const Configuration&    rConfig);
96
97      /** Add cost relating to the maxspeed of the edge.
98       * The cost is the number of seconds to travel the edge.
99       * @param   rEdge           The edge to add cost to
100      * @param   speed           The speed for the edge found in the database.
101      * @param   surfaceString   The surface as string or empty if not specified.
102      * @param   rConfig         Configuration
103      * @throw   MapProviderException
104      */
105     static void     addTravelTimeCostToEdge(
106                     Edge&                   rEdge,
107                     Speed                   speed,
108                     std::string&            surfaceString,
109                     const Configuration&    rConfig);
110
111     /** If the speed in the db was not set we must fetch the default
112      * for this road category from the configuration.
113      * @param   rEdge   The edge to find the default speed for
114      * @param   rConfig         Configuration
115      * @return  The default speed for this type of highway.
116      */
117     static Speed    getDefaultSpeedForEdge(
118                     const Edge&             rEdge,
119                     const Configuration&    rConfig);
120
121     /** Query for costs under the highway and railway tags:
122      * Highway:
123      *  - bus_stop
124      *  - crossing
125      *  - give_way
126      *  - mini_roundabout
127      *  - stop
128      *  - traffic_signals
129      * Railway:
130      *  - level_crossing
131      * Traffic calming
132      *
133      * @param   rTrans          Transaction to perform query in
134      * @param   rResult         Store the result of the query here
135      * @param   rOsmPointTable  Name of table with OSM points (nodes)
136      * @param   rTopoEdgeTable  Name of table with topology edges.
137      * @param   rOsmEdgeTable   Name of table with OSM edges.
```

```
138         * @param   rSchemaName     The name of the schema with topology info.
139         */
140        static void     getOtherCosts(
141                            pqxx::transaction_base&     rTrans,
142                            pqxx::result&               rResult,
143                            const std::string&          rOsmPointTable,
144                            const std::string&          rTopoEdgeTable,
145                            const std::string&          rOsmEdgeTable,
146                            const std::string&          rSchemaName);
147
148        /** Add costs for speed bumps and such to affected edges.
149         * @param   rResult     The results of the query.
150         * @param   rTopology   The topology with edges to set cost for.
151         * @param   rConfig     The Configuration
152         * @throw   MapProviderException
153         */
154        static void     addOtherCosts(
155                            const pqxx::result&     rResult,
156                            Topology&               rTopology,
157                            const Configuration&    rConfig);
158
159        /** Add a cost of an other type to the edge.
160         * Look up the value in the configuration.
161         * @param   rEdge   The Edge to add a cost to.
162         * @param   key     The type of cost as a string
163         * @param   rConfig     The Configuration
164         */
165        static void     addOtherCostToEdge(
166                            Edge&                   rEdge,
167                            const std::string&      key,
168                            const Configuration&    rConfig);
169
170        /** While looking for restrictions and we come across barriers,
171         * add the costs for barriers if they incur costs.
172         * @param   edge    The edge with a barrier.
173         * @param   type    The type of barrier.
174         * @param   rConfig The Configuration to use for the cost.
175         */
176        static void     addBarrierCostToEdge(
177                            Edge&                   rEdge,
178                            OsmBarrier::BarrierType type,
179                            const Configuration&    rConfig);
180
181    // ACCESS
182    // INQUIRY
183
184    protected:
185    private:
186        /** SELECT FROM JOIN */
187        static std::string  startOfQuery(const std::string& rTopoEdgeTable);
188
189        /** Which columns to pick */
190        static std::string  queryColumns(const std::vector<std::string>& rCols);
191
192        /** FROM JOIN ON WHERE */
193        static std::string  midOfQuery(
194            const std::string& rSchemaName,
```

```
195         const std::string& rOsmEdgeTable);
196
197     /** Make sure only to pick rows with content in some column. */
198     static std::string  notNullColumns(const std::vector<std::string>& rCols);
199
200     /** AS ON ORDER BY */
201     static std::string  endOfQuery();
202 };
203
204 #endif /* MAPPROVIDER_POSTGIS_COSTQUERIES_H_ */
```

D.11.5 CostQueries.cc

```
1   /*
2    * CostQueries.cc
3    *
4    * @author  Jonas Bergman
5    */
6
7   #include "CostQueries.h"  // class implemented
8
9   ////////////////////////////// PUBLIC //////////////////////////////////////
10
11  //============================= LIFECYCLE ====================================
12  //============================= OPERATORS ====================================
13  //============================= OPERATIONS ===================================
14  //static
15  void
16  CostQueries::getTravelTimeEdgeCosts(
17      pqxx::transaction_base& rTrans,
18      pqxx::result&           rResult,
19      const std::string&      rTopoEdgeTable,
20      const std::string&      rOsmEdgeTable,
21      const std::string&      rSchemaName)
22  {
23      std::vector<std::string> columns {
24          "maxspeed",
25          "surface"
26      };
27
28      rResult = rTrans.exec(
29          startOfQuery(rTopoEdgeTable) +
30          queryColumns(columns) +
31          midOfQuery(rSchemaName, rOsmEdgeTable) +
32  //        notNullColumns(columns) +
33          endOfQuery()
34      );
35  }
36
37  // static
38  void
39  CostQueries::addTravelTimeCosts(
40      const pqxx::result&     rResult,
41      Topology&               rTopology,
42      const Configuration&    rConfig)
43  {
44      try
```

```
45      {
46          for(const pqxx::tuple& row : rResult)
47          {
48              // throw exception if no edgeId
49              EdgeIdType edgeId =
50                  row[TravelTimeCostResult::EDGE_ID].as<EdgeIdType>();
51
52              Edge& edge = rTopology.getEdge(edgeId);
53
54              Speed speed =
55                  row[TravelTimeCostResult::MAXSPEED].as<Speed>(
56                      EdgeRestriction::VehicleProperties::DEFAULT_SPEED_MAX);
57              std::string surface_string =
58                  row[TravelTimeCostResult::SURFACE].as<std::string>("");
59
60              addTravelTimeCostToEdge(edge, speed, surface_string, rConfig);
61          }
62      }
63      catch (std::exception& e)
64      {
65          throw MapProviderException(
66              std::string("PostGisProvider:addTravelTimeCost: ") + e.what());
67      }
68  }
69
70  //static
71  void
72  CostQueries::addTravelTimeCostToEdge(
73      Edge&                   rEdge,
74      Speed                   speed,
75      std::string&            surfaceString,
76      const Configuration&    rConfig)
77  {
78      bool hasMaxSpeed =
79          (speed != EdgeRestriction::VehicleProperties::DEFAULT_SPEED_MAX);
80      bool hasSurface = surfaceString.length() > 0;
81      if(!(hasMaxSpeed || hasSurface))
82      {
83          speed = getDefaultSpeedForEdge(rEdge, rConfig);
84      }
85      // look if surface restricts speed
86      else if(hasSurface)
87      {
88          try
89          {
90              OsmHighway::SurfaceType surface =
91                  OsmHighway::parseSurfaceString(surfaceString);
92              Speed surfaceSpeed =
93                  rConfig.getCostConfig().surfaceMaxSpeed.getSurfaceMaxSpeed(surface);
94              if(surfaceSpeed < speed)
95              {
96                  speed = surfaceSpeed;
97              }
98          }
99          catch (OsmException& e)
100         {
101             throw MapProviderException(
```

```
102                 std::string("CostQueries:addTravelTime... ") +
103                 "could not parse surface " + surfaceString);
104         }
105     }
106     double speed_mps = speed / 3.6;
107     double travel_time = rEdge.geomData().length/ speed_mps;
108     rEdge.edgeCost().addCost(EdgeCost::TRAVEL_TIME, travel_time);
109     rEdge.setSpeed(speed);
110 }
111
112 // static
113 Speed
114 CostQueries::getDefaultSpeedForEdge(
115     const Edge&             rEdge,
116     const Configuration&    rConfig)
117 {
118     OsmHighway::HighwayType type = rEdge.roadData().roadType;
119     const CostConfig& costConfig = rConfig.getCostConfig();
120     Speed speed=
121         costConfig.defaultSpeed.getDefaultSpeed(type, CostConfig::DefaultSpeed::LOW);
122     return speed;
123 }
124
125 // static
126 void
127 CostQueries::getOtherCosts(
128     pqxx::transaction_base& rTrans,
129     pqxx::result&           rResult,
130     const std::string&      rOsmPointTable,
131     const std::string&      rTopoEdgeTable,
132     const std::string&      rOsmEdgeTable,
133     const std::string&      rSchemaName)
134 {
135     rResult = rTrans.exec(
136         "SELECT p.osm_id, "
137         "       p.highway, "
138         "       p.railway, "
139         "       p.public_transport, "
140         "       p.traffic_calming, "
141         "       t.edge_id "
142         "FROM  " + rOsmPointTable + " p, "
143         "      " + rTopoEdgeTable + " t, "
144         "      " + rOsmEdgeTable + " o, "
145         "      " + rSchemaName + ".relation r "
146         "WHERE  r.topogeo_id = (topo_geom).id "
147         "AND    r.element_id = t.edge_id "
148         "AND   (p.highway = 'bus_stop' OR "
149         "       p.highway = 'crossing' OR "
150         "       p.highway = 'give_way' OR"
151         "       p.highway = 'mini_roundabout' OR"
152         "       p.highway = 'stop' OR"
153         "       p.highway = 'traffic_signals' OR"
154         "       p.railway = 'level_crossing' OR"
155         "       p.public_transport = 'stop_position' OR"
156         "       p.traffic_calming = 'yes' OR"
157         "       p.traffic_calming = 'bump' OR"
158         "       p.traffic_calming = 'hump' OR"
```

```
159         "       p.traffic_calming = 'table' OR"
160         "       p.traffic_calming = 'cushion' OR"
161         "       p.traffic_calming = 'rumble_strip' OR"
162         "       p.traffic_calming = 'chicane' OR"
163         "       p.traffic_calming = 'choker' OR"
164         "       p.traffic_calming = 'island' "
165         ")"
166         "AND    ST_Intersects(p.way, t.geom) "
167         "AND    o.highway IN " + OsmHighway::typesAsCommaSeparatedString()
168     );
169 }
170
171 // static
172 void
173 CostQueries::addOtherCosts(
174     const pqxx::result&     rResult,
175     Topology&               rTopology,
176     const Configuration&    rConfig)
177 {
178     try
179     {
180         for(const pqxx::tuple& row : rResult)
181         {
182             // throw exception if no edgeId
183             EdgeIdType edgeId =
184                 row[OtherCostResult::EDGE_ID].as<EdgeIdType>();
185
186             Edge& edge = rTopology.getEdge(edgeId);
187
188             std::string type_string = "highway=" +
189                 row[OtherCostResult::HIGHWAY].as<std::string>("");
190             addOtherCostToEdge(edge, type_string, rConfig);
191
192             type_string = "railway=" +
193                 row[OtherCostResult::RAILWAY].as<std::string>("");
194             addOtherCostToEdge(edge, type_string, rConfig);
195
196             type_string = "public_transport=" +
197                 row[OtherCostResult::PUBLIC_TRANSPORT].as<std::string>("");
198             addOtherCostToEdge(edge, type_string, rConfig);
199
200             type_string = "traffic_calming=" +
201                 row[OtherCostResult::TRAFFIC_CALMING].as<std::string>("");
202             addOtherCostToEdge(edge, type_string, rConfig);
203         }
204     }
205     catch (std::exception& e)
206     {
207         throw MapProviderException(
208             std::string("CostQueries:addOtherCosts..: ") + e.what());
209     }
210 }
211
212 // static
213 void
214 CostQueries::addOtherCostToEdge(
215     Edge&                   rEdge,
```

```
216     const std::string&      key,
217     const Configuration&    rConfig)
218 {
219     size_t eq_char = key.find('=');
220     if((eq_char == std::string::npos) || (eq_char == key.length() - 1))
221     {
222         return;
223     }
224 
225     Cost cost = rConfig.getCostConfig().otherEdgeCosts.getOtherCost(key);
226     rEdge.edgeCost().addCost(EdgeCost::OTHER, cost);
227 }
228 
229 // static
230 void
231 CostQueries::addBarrierCostToEdge(
232     Edge&                   rEdge,
233     OsmBarrier::BarrierType type,
234     const Configuration&    rConfig)
235 {
236     if(rConfig.getBarrierCostsRule().costsToPass(type))
237     {
238         Cost cost = rConfig.getBarrierCostsRule().getCost(type);
239         rEdge.edgeCost().addCost(EdgeCost::BARRIER, cost);
240     }
241 }
242 
243 //============================== ACESS      ==================================
244 //============================== INQUIRY    ==================================
245 ////////////////////////////////// PROTECTED  ////////////////////////////////////
246 ////////////////////////////////// PRIVATE    ////////////////////////////////////
247 //static
248 std::string
249 CostQueries::startOfQuery(const std::string& rTopoEdgeTable)
250 {
251     return (
252         "SELECT     edge_id, "
253         //-- osm data about original edge
254         "           osm.* "
255         "FROM      " + rTopoEdgeTable +
256         " JOIN ( "
257         "   SELECT  element_id "
258     );
259 }
260 
261 //static
262 std::string
263 CostQueries::queryColumns(const std::vector<std::string>& rCols)
264 {
265     std::ostringstream oss;
266     for(const std::string& col : rCols)
267     {
268         oss << ", " << col;
269     }
270     return oss.str();
271 }
272 
```

```
273 //static
274 std::string
275 CostQueries::midOfQuery(
276     const std::string& rSchemaName,
277     const std::string& rOsmEdgeTable)
278 {
279     return (
280         "   FROM    " + rSchemaName + ".relation "
281         "   JOIN    " + rOsmEdgeTable +
282         "   ON      topogeo_id = (topo_geom).id "
283         "   WHERE   highway in " + OsmHighway::typesAsCommaSeparatedString()
284     );
285 }
286
287 //static
288 std::string
289 CostQueries::notNullColumns(const std::vector<std::string>& rCols)
290 {
291     std::ostringstream oss;
292     oss << " AND (";
293     size_t i = 0;
294     for(const std::string& col : rCols)
295     {
296         oss << col << " IS NOT NULL ";
297         if(i < (rCols.size() - 1))
298         {
299             oss << " OR ";
300         }
301         ++i;
302     }
303     oss << ") ";
304     return oss.str();
305 }
306
307 //static
308 std::string
309 CostQueries::endOfQuery()
310 {
311     return (
312         ") AS osm "
313         "ON edge_id = element_id "
314         "ORDER BY edge_id ASC;"
315     );
316 }
```

D.11.6 LineGraphSaveQueries.h

```
1  /** Queries for saving the LineGraph to database
2   *
3   * #include "LineGraphSaveQueries.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef MAPPROVIDER_POSTGIS_LINEGRAPHSAVEQUERIES_H_
9  #define MAPPROVIDER_POSTGIS_LINEGRAPHSAVEQUERIES_H_
10
```

```
11  // SYSTEM INCLUDES
12  //
13  #include <string>
14  #include <sstream>
15  #include <vector>
16
17  // PROJECT INCLUDES
18  //
19  #include <boost/algorithm/string.hpp>
20  #include <pqxx/pqxx>
21
22  // LOCAL INCLUDES
23  //
24  #include "../../graph/GraphBuilder.h"
25  #include "../../graph/Topology.h"
26
27  /** Class for holding static queries for saving the line graph to database
28   */
29  class LineGraphSaveQueries
30  {
31  public:
32  // TYPES
33
34  // LIFECYCLE
35      LineGraphSaveQueries() = delete;
36      LineGraphSaveQueries(const LineGraphSaveQueries& from) = delete;
37      ~LineGraphSaveQueries() = default;
38
39  // OPERATORS
40  // OPERATIONS
41
42      /** Create a new schema, dropping any existing with the same name first.
43       * @param   rTrans      Transaction to perform the query in.
44       * @param   rSchemaName Name of the schema to create.
45       * @throw   pqxx::pqxx_exception
46       */
47      static void dropCreateSchema(
48          pqxx::transaction_base& rTrans,
49          const std::string&      rSchemaName);
50
51      /** Create a new table for lines, dropping any existing with the same name.
52       * @param   rTrans          Transaction to perform the query in.
53       * @param   rTableName      Name of the table to create.
54       * @throw   pqxx::pqxx_exception
55       */
56      static void dropCreateLineTable(
57          pqxx::transaction_base& rTrans,
58          const std::string&      rTableName);
59
60      /** Create a new table for nodes, dropping any existing with the same name.
61       * @param   rTrans          Transaction to perform the query in.
62       * @param   rTableName      Name of the table to create.
63       * @throw   pqxx::pqxx_exception
64       */
65      static void dropCreateNodeTable(
66          pqxx::transaction_base& rTrans,
67          const std::string&      rTableName);
```

```
68
69      /** Insert a node into the database
70       * @param   rTrans      Transaction to perform the query in.
71       * @param   id          The id of the node's corresponding edge in the topology.
72       * @param   rGeomString WKT (well-known text) representation of the node
73       * @throw   pqxx::pqxx_exception
74       */
75      static void insertNode(
76          pqxx::transaction_base& rTrans,
77          const std::string&      rTableName,
78          EdgeIdType              id,
79          const std::string&      rGeomString);
80
81      /** Insert a line into the database
82       * @param   rTrans      Transaction to perform the query in.
83       * @param   cost        The cost of traveling the edge
84       * @param   rGeomString WKT (well-known text) representation of the line
85       * @throw   pqxx::pqxx_exception
86       */
87      static void insertLine(
88          pqxx::transaction_base& rTrans,
89          const std::string&      rTableName,
90          Cost                    cost,
91          const std::string&      rGeomString);
92
93  // ACCESS
94  // INQUIRY
95
96  protected:
97  private:
98  };
99
100 #endif /* MAPPROVIDER_POSTGIS_LINEGRAPHSAVEQUERIES_H_ */
```

D.11.7 LineGraphSaveQueries.cc

```
1   /*
2    * LineGraphSaveQueries.cc
3    *
4    * @author  Jonas Bergman
5    */
6
7   #include "LineGraphSaveQueries.h"
8
9   ////////////////////////////// PUBLIC //////////////////////////////////////
10
11  //============================= LIFECYCLE ====================================
12  //============================= OPERATORS ====================================
13  //============================= OPERATIONS ===================================
14  //static
15  void
16  LineGraphSaveQueries::dropCreateSchema(
17      pqxx::transaction_base& rTrans,
18      const std::string&      rSchemaName)
19  {
20      rTrans.exec(
21              "DROP SCHEMA IF EXISTS " + rSchemaName + " CASCADE; "
```

```
22              "CREATE SCHEMA " + rSchemaName
23      );
24  }
25
26  //static
27  void
28  LineGraphSaveQueries::dropCreateLineTable(
29      pqxx::transaction_base& rTrans,
30      const std::string&      rTableName)
31  {
32      rTrans.exec(
33              "DROP TABLE IF EXISTS " + rTableName + " CASCADE; "
34              "CREATE TABLE " + rTableName + " ( "
35              "    cost     double precision, "
36              "    geom     geometry(LineString, 900913) "
37              "); "
38      );
39  }
40
41  //static
42  void
43  LineGraphSaveQueries::dropCreateNodeTable(
44      pqxx::transaction_base& rTrans,
45      const std::string&      rTableName)
46  {
47      rTrans.exec(
48              "DROP TABLE IF EXISTS " + rTableName + " CASCADE; "
49              "CREATE TABLE " + rTableName + " ( "
50              "    topo_id  bigint unique, "
51              "    geom     geometry(Point, 900913) "
52              "); "
53      );
54  }
55
56  //static
57  void
58  LineGraphSaveQueries::insertNode(
59      pqxx::transaction_base& rTrans,
60      const std::string&      rTableName,
61      EdgeIdType              id,
62      const std::string&      rGeomString)
63  {
64
65      rTrans.exec(
66          "INSERT INTO " + rTableName +" (topo_id, geom) "
67          "SELECT " + std::to_string(id) +
68          ", ST_GeomFromText('" + rGeomString + "', 900913) "
69          "WHERE NOT EXISTS ("
70          "    SELECT topo_id FROM " + rTableName +
71          "    WHERE topo_id = " + std::to_string(id) + " );"
72      );
73  }
74
75  //static
76  void
77  LineGraphSaveQueries::insertLine(
78      pqxx::transaction_base& rTrans,
```

```
79      const std::string&      rTableName,
80      Cost                    cost,
81      const std::string&      rGeomString)
82  {
83      rTrans.exec(
84          "INSERT INTO " + rTableName +" (cost, geom) "
85          "VALUES (" + std::to_string(cost) +
86          ", ST_GeomFromText('" + rGeomString + "', 900913)); "
87      );
88  }
89  //============================== ACESS      ====================================
90  //============================== INQUIRY    ====================================
91  ////////////////////////////////// PROTECTED  ////////////////////////////////////
92  ////////////////////////////////// PRIVATE    ////////////////////////////////////
```

D.11.8 RestrictionQueries.h

```
1   /**  Queries for PostGisProvider to find restrictions.
2    *
3    * #include "RestrictionQueries.h"
4    *
5    * @author  Jonas Bergman
6    */
7
8   #ifndef MAPPROVIDER_POSTGIS_RESTRICTIONQUERIES_H_
9   #define MAPPROVIDER_POSTGIS_RESTRICTIONQUERIES_H_
10
11  // SYSTEM INCLUDES
12  //
13  #include <string>
14  #include <sstream>
15  #include <vector>
16
17  // PROJECT INCLUDES
18  //
19  #include <boost/algorithm/string.hpp>
20  #include <pqxx/pqxx>
21
22  // LOCAL INCLUDES
23  //
24  #include "CostQueries.h"
25  #include "../../osm/OsmHighway.h"
26  #include "../../osm/OsmTurningRestriction.h"
27  #include "../../graph/Edge.h"
28  #include "../../graph/EdgeRestriction.h"
29  #include "../../graph/Topology.h"
30  #include "../../graph/Vertex.h"
31  #include "../MapProviderException.h"
32
33  /** Class for holding static queries about restrictions,
34   * needed by the PostGisProvider.
35   */
36  class RestrictionQueries
37  {
38  public:
39  // TYPES
40
```

```
41      /** Columns used in query for Vehicle Properties restrictions. */
42      struct VehiclePropertiesRestrictions
43      {
44          enum Columns
45          {
46              EDGE_ID,
47              ELEMENT_ID,
48              MAXHEIGHT,
49              MAXLENGTH,
50              MAXWEIGHT,
51              MAXWIDTH,
52              MAXSPEED,
53              MINSPEED,
54          };
55      };
56
57      /** Columns used in query for Access restrictions. */
58      struct AccessRestrictions
59      {
60          enum Columns
61          {
62              EDGE_ID,
63              ELEMENT_ID,
64              ACCESS,
65              BARRIER,
66              DISUSED,
67              NOEXIT,
68              MOTORCAR,
69              GOODS,
70              HGV,
71              PSV,
72              LHV,
73              MOTOR_VEHICLE,
74              VEHICLE,
75          };
76      };
77
78      /** Columns used in query for Turning restrictions. */
79      struct TurningRestrictions
80      {
81          enum Columns
82          {
83              FROM_OSM_ID,
84              TO_OSM_ID,
85              VIA_OSM,
86              EDGE_IDS,
87              RESTRICTION_TYPE
88          };
89
90          /** Results from queries are handled by these functions. */
91          struct Results
92          {
93              /** Parse a row in the results from turning restrictions.
94               * @param   rRow        The row with results.
95               * @param   rTopology   The topology that needs to be queried.
96               * @return  A turning restriction object.
97               * @throw   pqxx::pqxx_exception
```

```
98              * @throw   TopologyException
99              * @throw   MapProviderException
100             */
101            static OsmTurningRestriction* parseTurningRestrictionResultRow(
102                const pqxx::tuple&    rRow,
103                Topology&             rTopology);
104
105
106            /** Try to parse the column 'edge_ids' from the 'turning_restrictions'.
107             * Split the string of edge ids and convert them to a vector of EdgeIds.
108             * @param   rEdgeIds    A string like "{123, 456}". Gets trimmed of {}.
109             * @return  A vector of the EdgeIds separately.
110             * @throw   std::invalid_argument
111             * @throw   std::out_of_range
112             */
113            static std::vector<EdgeIdType> parseEdgeIdsString(
114                std::string& rEdgeIds);
115
116            /** Find the Edge that matches the OsmId in turning restriction.
117             * @param   osmId       The original edge osm id.
118             * @param   rEdgeIds    The candidate edges that are near restriction.
119             * @param   rTopology   The Topology to ask for edges.
120             */
121            static Edge&        findEdgeMatchingOsmId(
122                OsmIdType                       osmId,
123                const std::vector<EdgeIdType>&  rEdgeIds,
124                Topology&                       rTopology);
125        };
126    };
127
128    /** Columns used in query for EdgePoint restrictions. */
129    struct EdgePointRestrictions
130    {
131        enum Columns
132        {
133            POINT_OSM_ID,
134            BARRIER,
135            ACCESS,
136            GOODS,
137            HGV,
138            LHV,
139            MOTORCAR,
140            MOTOR_VEHICLE,
141            PSV,
142            VEHICLE,
143            EDGE_ID,
144        };
145    };
146
147
148// LIFECYCLE
149    RestrictionQueries() = delete;
150    RestrictionQueries(const RestrictionQueries& from) = delete;
151    ~RestrictionQueries() = default;
152
153// OPERATORS
154// OPERATIONS
```

```
155     /** Query for restrictions based on Vehicle Properties.
156      * @param   rTrans          Transaction to perform query in.
157      * @param   rResult         Store the result of query here.
158      * @param   rTopoEdgeTable  Name of table with topology edges.
159      * @param   rOsmEdgeTable   Name of table with OSM edges.
160      * @param   rSchemaName     Name of the topology schema.
161      * @throw   pqxx::pqxx_exception
162      */
163     static void         getVehiclePropertyEdgeRestrictions(
164                             pqxx::transaction_base& rTrans,
165                             pqxx::result&           rResult,
166                             const std::string&      rTopoEdgeTable,
167                             const std::string&      rOsmEdgeTable,
168                             const std::string&      rSchemaName);
169
170     /** Add the result of the query for vehicle properties to Edge's restrictions.
171      * @param   rResult         The results of the query
172      * @param   rTopology       Update affected edges in the topology.
173      * @throw   MapProviderException
174      */
175     static void         addVehiclePropertyRestrictionsToEdge(
176                             const pqxx::result&    rResult,
177                             Topology&              rTopology);
178
179     /** Query for general access restrictions.
180      * @param   rTrans          Transaction to perform query in.
181      * @param   rResult         Store the result of query here.
182      * @param   rTopoEdgeTable  Name of table with topology edges.
183      * @param   rOsmEdgeTable   Name of table with OSM edges.
184      * @param   rSchemaName     Name of the topology schema.
185      * @throw   pqxx::pqxx_exception
186      */
187     static void         getAccessRestrictions(
188                             pqxx::transaction_base& rTrans,
189                             pqxx::result&           rResult,
190                             const std::string&      rTopoEdgeTable,
191                             const std::string&      rOsmEdgeTable,
192                             const std::string&      rSchemaName);
193
194     /** Add the result of the query for Access to restrictions.
195      * @param   rResult         The results of the query
196      * @param   rTopology       Update affected edges in the topology.
197      * @param   rConfig         Configuration
198      * @throw   MapProviderException
199      */
200     static void         addAccessRestrictionsToEdge(
201                             const pqxx::result&    rResult,
202                             Topology&              rTopology,
203                             const Configuration&   rConfig);
204
205     /** Drop and create the table 'turning_restrictions'.
206      * @param   rTrans          Transaction to perform query in.
207      * @throw   pqxx::pqxx_exception
208      */
209     static void         dropCreateTurningRestrictionsTable(
210                             pqxx::transaction_base& rTrans);
211
```

```
212     /** Populate the table 'turning_restrictions'.
213      * @param   rTrans          The transaction to execute within.
214      * @param   rOsmEdgeTable   The name of the table with original osm edges.
215      * @param   rTopoEdgeTable  The name of the table with topology edges.
216      * @throw   pqxx::pqxx_exception
217      */
218     static void         identifyTurningRestrictions(
219                             pqxx::transaction_base& rTrans,
220                             const std::string&      rOsmEdgeTable,
221                             const std::string&      rTopoEdgeTable);
222
223     /** Get the restrictions from the 'turning_restrictions' table.
224      * @param   rTrans          Transaction to perform query in.
225      * @param   rResult         Store the result of query here.
226      * @throw   pqxx::pqxx_exception
227      */
228     static void         getTurningRestrictions(
229                             pqxx::transaction_base& rTrans,
230                             pqxx::result&           rResult);
231
232     /** Add the result of the query for Turning restrictions.
233      * @param   rResult         The results of the query
234      * @param   rTopology       Update affected edges in the topology.
235      * @throw   MapProviderException
236      */
237     static void         addTurningRestrictionsToEdge(
238                             const pqxx::result&    rResult,
239                             Topology&              rTopology);
240
241     /** Get the restrictions from the 'planet_osm_point' that relates to edges.
242      * @param   rTrans          Transaction to perform query in.
243      * @param   rResult         Store the result of query here.
244      * @param   rOsmPointTabl   The name of the table with original osm points.
245      * @param   rTopoEdgeTable  The name of the table with topology edges.
246      * @param   rOsmEdgeTable   The name of the table with OSM edges.
247      * @param   rSchemaName     The name of the schema with topology info.
248      * @throw   pqxx::pqxx_exception
249      */
250     static void         getEdgePointRestrictions(
251                             pqxx::transaction_base& rTrans,
252                             pqxx::result&           rResult,
253                             const std::string&      rOsmPointTable,
254                             const std::string&      rTopoEdgeTable,
255                             const std::string&      rOsmEdgeTable,
256                             const std::string&      rSchemaName);
257
258     /** Add the result of the query for Point restrictions on Edges .
259      * @param   rResult         The results of the query
260      * @param   rTopology       Update affected edges in the topology.
261      * @param   rConfig         Configuration
262      * @throw   MapProviderException
263      */
264     static void         addPointRestrictionsToEdge(
265                             const pqxx::result&    rResult,
266                             Topology&              rTopology,
267                             const Configuration&   rConfig);
268 // ACCESS
```

```
269 // INQUIRY
270
271 protected:
272 private:
273     /** SELECT FROM JOIN */
274     static std::string  startOfQuery(const std::string& rTopoEdgeTable);
275
276     /** Which columns to pick */
277     static std::string  queryColumns(const std::vector<std::string>& rCols);
278
279     /** FROM JOIN ON WHERE */
280     static std::string  midOfQuery(
281         const std::string& rSchemaName,
282         const std::string& rOsmEdgeTable);
283
284     /** Make sure only to pick rows with content in some column. */
285     static std::string  notNullColumns(const std::vector<std::string>& rCols);
286
287     /** AS ON ORDER BY */
288     static std::string  endOfQuery();
289 };
290
291 #endif /* MAPPROVIDER_POSTGIS_RESTRICTIONQUERIES_H_ */
```

D.11.9 RestrictionQueries.cc

```
1  /*
2   * RestrictionQueries.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "RestrictionQueries.h"  // class implemented
8
9  // Result ------------------------------------------------------------------
10 //static
11 OsmTurningRestriction*
12 RestrictionQueries::TurningRestrictions::
13     Results::parseTurningRestrictionResultRow(
14     const pqxx::tuple&    rRow,
15     Topology&             rTopology)
16 {
17     OsmIdType fromOsmId =
18         rRow[RestrictionQueries::TurningRestrictions::FROM_OSM_ID].as<OsmIdType>();
19     OsmIdType toOsmId =
20         rRow[RestrictionQueries::TurningRestrictions::TO_OSM_ID].as<OsmIdType>();
21     std::string typeString =
22         rRow[RestrictionQueries::TurningRestrictions::RESTRICTION_TYPE].as<std::string>();
23     OsmTurningRestriction::TurningRestrictionType type =
24         OsmTurningRestriction::parseString(typeString);
25     std::string edgeIdsString =
26         rRow[RestrictionQueries::TurningRestrictions::EDGE_IDS].as<std::string>();
27     std::string viaOsmIdsString =
28         rRow[RestrictionQueries::TurningRestrictions::VIA_OSM].as<std::string>("");
29
30     std::vector<EdgeIdType> edgeIds = parseEdgeIdsString(edgeIdsString);
31
```

```
32      Edge& fromEdge = findEdgeMatchingOsmId(fromOsmId, edgeIds, rTopology);
33      Edge& toEdge   = findEdgeMatchingOsmId(toOsmId, edgeIds, rTopology);
34
35      OsmTurningRestriction* p_restriction {nullptr};
36
37      // VIA WAY
38      if(fromEdge.targetId() != toEdge.sourceId())
39      {
40          p_restriction = new OsmTurningRestriction(
41              type,
42              fromEdge.id(),
43              viaOsmIdsString,
44              toEdge.id());
45      }
46      // VIA NODE
47      else
48      {
49          VertexIdType vertexId = fromEdge.targetId();
50          p_restriction = new OsmTurningRestriction(
51              type,
52              fromEdge.id(),
53              vertexId,
54              toEdge.id());
55      }
56      return p_restriction;
57  }
58
59  //static
60  std::vector<EdgeIdType>
61  RestrictionQueries::TurningRestrictions::Results::parseEdgeIdsString(
62      std::string& rEdgeIds)
63  {
64      boost::trim_if(rEdgeIds, boost::is_any_of("{}"));
65      std::vector<std::string> idStrings;
66      boost::split(idStrings, rEdgeIds, boost::is_any_of(","));
67
68      std::vector<EdgeIdType> edgeIds;
69      for(const std::string& idStr : idStrings)
70      {
71          edgeIds.push_back(Edge::parse(idStr));
72      }
73      return edgeIds;
74  }
75
76  //static
77  Edge&
78  RestrictionQueries::TurningRestrictions::Results::findEdgeMatchingOsmId(
79      OsmIdType                       osmId,
80      const std::vector<EdgeIdType>&  rEdgeIds,
81      Topology&                       rTopology)
82  {
83      for(EdgeIdType id : rEdgeIds)
84      {
85          Edge& edge = rTopology.getEdge(id);
86          if(edge.osmId() == osmId)
87          {
88              return edge;
```

```
89          }
90      }
91      throw MapProviderException(
92          "PostGisRestrictionQueries:Result:findEdgeMatchingOsmId: "
93          "No edges matching osm_id: " + std::to_string(osmId));
94  }
95
96  ////////////////////////////// PUBLIC ////////////////////////////////////
97
98  //============================= LIFECYCLE ====================================
99  //============================= OPERATORS ====================================
100 //============================= OPERATIONS ===================================
101 //static
102 void
103 RestrictionQueries::getVehiclePropertyEdgeRestrictions(
104     pqxx::transaction_base& rTrans,
105     pqxx::result&           rResult,
106     const std::string&      rTopoEdgeTable,
107     const std::string&      rOsmEdgeTable,
108     const std::string&      rSchemaName)
109 {
110     std::vector<std::string> columns {
111         "maxheight",
112         "maxlength",
113         "maxweight",
114         "maxwidth",
115         "maxspeed",
116         "minspeed"
117     };
118
119     rResult = rTrans.exec(
120         startOfQuery(rTopoEdgeTable) +
121         queryColumns(columns) +
122         midOfQuery(rSchemaName, rOsmEdgeTable) +
123         notNullColumns(columns) +
124         endOfQuery()
125     );
126 }
127
128 // static
129 void
130 RestrictionQueries::addVehiclePropertyRestrictionsToEdge(
131     const pqxx::result& rResult,
132     Topology&           rTopology)
133 {
134     try
135     {
136         for(const pqxx::tuple& row : rResult)
137         {
138             // throw exception if no edgeId
139             EdgeIdType edgeId =
140                 row[VehiclePropertiesRestrictions::EDGE_ID].as<EdgeIdType>();
141
142             Edge& edge = rTopology.getEdge(edgeId);
143             EdgeRestriction& r_restrictions = edge.restrictions();
144
145             EdgeRestriction::VehicleProperties* p_vp =
```

```
146                 new EdgeRestriction::VehicleProperties();
147
148             p_vp->maxHeight =
149                 row[VehiclePropertiesRestrictions::MAXHEIGHT].as<double>
150                 (EdgeRestriction::VehicleProperties::DEFAULT_DIMENSION_MAX);
151             p_vp->maxLength =
152                 row[VehiclePropertiesRestrictions::MAXLENGTH].as<double>
153                 (EdgeRestriction::VehicleProperties::DEFAULT_DIMENSION_MAX);
154             p_vp->maxWeight =
155                 row[VehiclePropertiesRestrictions::MAXWEIGHT].as<double>
156                 (EdgeRestriction::VehicleProperties::DEFAULT_DIMENSION_MAX);
157             p_vp->maxWidth =
158                 row[VehiclePropertiesRestrictions::MAXWIDTH].as<double>
159                 (EdgeRestriction::VehicleProperties::DEFAULT_DIMENSION_MAX);
160             p_vp->maxSpeed =
161                 row[VehiclePropertiesRestrictions::MAXSPEED].as<unsigned>
162                 (EdgeRestriction::VehicleProperties::DEFAULT_SPEED_MAX);
163             p_vp->minSpeed =
164                 row[VehiclePropertiesRestrictions::MINSPEED].as<unsigned>
165                 (EdgeRestriction::VehicleProperties::DEFAULT_SPEED_MIN);
166
167             r_restrictions.setVehiclePropertyRestriction(p_vp);
168         }
169     }
170     catch (std::exception& e)
171     {
172         throw MapProviderException(
173             std::string("RestrictionQueries:addVehicleProp..ToEdge..: ")
174                         + e.what());
175     }
176 }
177
178 //static
179 void
180 RestrictionQueries::getAccessRestrictions(
181     pqxx::transaction_base& rTrans,
182     pqxx::result&           rResult,
183     const std::string&      rTopoEdgeTable,
184     const std::string&      rOsmEdgeTable,
185     const std::string&      rSchemaName)
186 {
187     std::vector<std::string> columns {
188         "access",
189         "barrier",
190         "disused",
191         "noexit",
192         "motorcar",
193         "goods",
194         "hgv",
195         "psv",
196         "lhv",
197         "motor_vehicle",
198         "vehicle"
199     };
200
201     rResult = rTrans.exec(
202         startOfQuery(rTopoEdgeTable) +
```

```
203         queryColumns(columns) +
204         midOfQuery(rSchemaName, rOsmEdgeTable) +
205         notNullColumns(columns) +
206         endOfQuery()
207     );
208 }
209
210 // static
211 void
212 RestrictionQueries::addAccessRestrictionsToEdge(
213     const pqxx::result&     rResult,
214     Topology&               rTopology,
215     const Configuration&    rConfig)
216 {
217     try
218     {
219         for(const pqxx::tuple& row : rResult)
220         {
221             // throw exception if no edgeId
222             EdgeIdType edgeId =
223                 row[AccessRestrictions::EDGE_ID].as<EdgeIdType>();
224
225             Edge& edge = rTopology.getEdge(edgeId);
226             EdgeRestriction& r_restrictions = edge.restrictions();
227
228             std::string colString;
229             colString = row[AccessRestrictions::ACCESS].as<std::string>("");
230             if(colString != "")
231             {
232                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
233                 r_restrictions.setGeneralAccessRestriction(type);
234             }
235
236             colString = row[AccessRestrictions::MOTORCAR].as<std::string>("");
237             if(colString != "")
238             {
239                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
240                 r_restrictions.addVehicleTypeAccessRestriction(
241                     OsmVehicle::MOTORCAR,
242                     type
243                 );
244             }
245
246             colString = row[AccessRestrictions::GOODS].as<std::string>("");
247             if(colString != "")
248             {
249                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
250                 r_restrictions.addVehicleTypeAccessRestriction(
251                     OsmVehicle::GOODS,
252                     type
253                 );
254             }
255
256             colString = row[AccessRestrictions::HGV].as<std::string>("");
257             if(colString != "")
258             {
259                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
```

```
260                 r_restrictions.addVehicleTypeAccessRestriction(
261                     OsmVehicle::HGV,
262                     type
263                 );
264             }
265 
266             colString = row[AccessRestrictions::PSV].as<std::string>("");
267             if(colString != "")
268             {
269                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
270                 r_restrictions.addVehicleTypeAccessRestriction(
271                     OsmVehicle::PSV,
272                     type
273                 );
274             }
275 
276             colString = row[AccessRestrictions::LHV].as<std::string>("");
277             if(colString != "")
278             {
279                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
280                 r_restrictions.addVehicleTypeAccessRestriction(
281                     OsmVehicle::LHV,
282                     type
283                 );
284             }
285 
286             colString = row[AccessRestrictions::MOTOR_VEHICLE].as<std::string>("");
287             if(colString != "")
288             {
289                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
290                 r_restrictions.addVehicleTypeAccessRestriction(
291                     OsmVehicle::MOTOR_VEHICLE,
292                     type
293                 );
294             }
295 
296             colString = row[AccessRestrictions::VEHICLE].as<std::string>("");
297             if(colString != "")
298             {
299                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
300                 r_restrictions.addVehicleTypeAccessRestriction(
301                     OsmVehicle::VEHICLE,
302                     type
303                 );
304             }
305 
306             colString = row[AccessRestrictions::BARRIER].as<std::string>("");
307             if(colString != "")
308             {
309                 OsmBarrier::BarrierType type = OsmBarrier::parseString(colString);
310                 r_restrictions.setBarrierRestriction(type);
311                 CostQueries::addBarrierCostToEdge(edge, type, rConfig);
312             }
313 
314             colString = row[AccessRestrictions::DISUSED].as<std::string>("");
315             if(colString == "yes")
316             {
```

```
317                 r_restrictions.setDisusedRestriction();
318             }
319 
320             colString = row[AccessRestrictions::NOEXIT].as<std::string>("");
321             if(colString == "yes")
322             {
323                 r_restrictions.setNoExitRestriction();
324             }
325         }
326     }
327     catch (std::exception& e)
328     {
329         throw MapProviderException(
330             std::string("RestrictionQueries:addAccessResultToEdge..: ") + e.what());
331     }
332 }
333 
334 //static
335 void
336 RestrictionQueries::dropCreateTurningRestrictionsTable(
337     pqxx::transaction_base& rTrans)
338 {
339     rTrans.exec(
340         "DROP TABLE IF EXISTS turning_restrictions; "
341         "CREATE TABLE turning_restrictions( "
342         "   from_osm_id       bigint, "
343         "   to_osm_id         bigint, "
344         "   via_osm           varchar, "
345         "   edge_ids          integer[], "
346         "   restriction_type  varchar)"
347     );
348 }
349 
350 //static
351 void
352 RestrictionQueries::identifyTurningRestrictions(
353     pqxx::transaction_base& rTrans,
354     const std::string&      rOsmEdgeTable,
355     const std::string&      rTopoEdgeTable)
356 {
357     rTrans.exec(
358         "SELECT * FROM find_osm_turning_restrictions('"
359         + rOsmEdgeTable + "', '" + rTopoEdgeTable + "')"
360     );
361 }
362 
363 //static
364 void
365 RestrictionQueries::getTurningRestrictions(
366     pqxx::transaction_base& rTrans,
367     pqxx::result&           rResult)
368 {
369     rResult = rTrans.exec(
370         "SELECT * FROM turning_restrictions"
371     );
372 }
373 
```

```
374 // static
375 void
376 RestrictionQueries::addTurningRestrictionsToEdge(
377     const pqxx::result&     rResult,
378     Topology&               rTopology)
379 {
380     try
381     {
382         for(const pqxx::tuple& row : rResult)
383         {
384             OsmTurningRestriction* p_turn =
385                 TurningRestrictions::Results::
386                     parseTurningRestrictionResultRow(row, rTopology);
387 
388             // mark edge as having a restriction
389             Edge& edge = rTopology.getEdge(p_turn->fromEdgeId());
390             EdgeRestriction& r_restrictions = edge.restrictions();
391             r_restrictions.addTurningRestriction(p_turn);
392 
393             // explicit mark "VIA WAY"
394             if(p_turn->viaType() == OsmTurningRestriction::VIA_WAY)
395             {
396                 r_restrictions.setViaWayRestriction();
397             }
398         }
399     }
400     catch (std::exception& e)
401     {
402         throw MapProviderException(
403             std::string("RestrictionQueries:addTurningResultToEdge..: ") + e.what());
404     }
405 }
406 
407 //static
408 void
409 RestrictionQueries::getEdgePointRestrictions(
410     pqxx::transaction_base& rTrans,
411     pqxx::result&           rResult,
412     const std::string&      rOsmPointTable,
413     const std::string&      rTopoEdgeTable,
414     const std::string&      rOsmEdgeTable,
415     const std::string&      rSchemaName)
416 {
417     rResult = rTrans.exec(
418         "SELECT p.osm_id, "
419         "       p.barrier, "
420         "       p.access, "
421         "       p.goods, "
422         "       p.hgv, "
423         "       p.lhv, "
424         "       p.motorcar, "
425         "       p.motor_vehicle, "
426         "       p.psv, "
427         "       p.vehicle, "
428         "       t.edge_id "
429         "FROM  " + rOsmPointTable + " p, "
430         "      " + rTopoEdgeTable + " t, "
```

```
431         "      " + rOsmEdgeTable + " o, "
432         "      " + rSchemaName + ".relation r "
433         "WHERE  r.topogeo_id = (topo_geom).id "
434         "AND    r.element_id = t.edge_id "
435         "AND    p.barrier IS NOT NULL "
436         "AND    ST_Intersects(p.way, t.geom) "
437         "AND    o.highway IN " + OsmHighway::typesAsCommaSeparatedString()
438     );
439 }
440 // static
441 void
442 RestrictionQueries::addPointRestrictionsToEdge(
443     const pqxx::result&     rResult,
444     Topology&               rTopology,
445     const Configuration&    rConfig)
446 {
447     try
448     {
449         for(const pqxx::tuple& row : rResult)
450         {
451             // throw exception if no edgeId
452             EdgeIdType edgeId =
453                 row[EdgePointRestrictions::EDGE_ID].as<EdgeIdType>();
454 
455             Edge& edge = rTopology.getEdge(edgeId);
456             EdgeRestriction& r_restrictions = edge.restrictions();
457 
458             std::string barrierTypeString =
459                 row[EdgePointRestrictions::BARRIER].as<std::string>();
460             OsmBarrier::BarrierType barrierType =
461                 OsmBarrier::parseString(barrierTypeString);
462             r_restrictions.setBarrierRestriction(barrierType);
463             CostQueries::addBarrierCostToEdge(edge, barrierType, rConfig);
464 
465             std::string colString;
466             colString = row[EdgePointRestrictions::ACCESS].as<std::string>("");
467             if(colString != "")
468             {
469                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
470                 r_restrictions.setGeneralAccessRestriction(type);
471             }
472 
473             colString = row[EdgePointRestrictions::GOODS].as<std::string>("");
474             if(colString != "")
475             {
476                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
477                 r_restrictions.addVehicleTypeAccessRestriction(
478                     OsmVehicle::GOODS,
479                     type);
480             }
481 
482             colString = row[EdgePointRestrictions::HGV].as<std::string>("");
483             if(colString != "")
484             {
485                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
486                 r_restrictions.addVehicleTypeAccessRestriction(
487                     OsmVehicle::HGV,
```

```
488                     type);
489             }
490
491             colString = row[EdgePointRestrictions::LHV].as<std::string>("");
492             if(colString != "")
493             {
494                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
495                 r_restrictions.addVehicleTypeAccessRestriction(
496                     OsmVehicle::LHV,
497                     type);
498             }
499
500             colString = row[EdgePointRestrictions::MOTORCAR].as<std::string>("");
501             if(colString != "")
502             {
503                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
504                 r_restrictions.addVehicleTypeAccessRestriction(
505                     OsmVehicle::MOTORCAR,
506                     type);
507             }
508
509             colString = row[EdgePointRestrictions::MOTOR_VEHICLE].as<std::string>("");
510             if(colString != "")
511             {
512                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
513                 r_restrictions.addVehicleTypeAccessRestriction(
514                     OsmVehicle::MOTOR_VEHICLE,
515                     type);
516             }
517
518             colString = row[EdgePointRestrictions::PSV].as<std::string>("");
519             if(colString != "")
520             {
521                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
522                 r_restrictions.addVehicleTypeAccessRestriction(
523                     OsmVehicle::PSV,
524                     type);
525             }
526
527             colString = row[EdgePointRestrictions::VEHICLE].as<std::string>("");
528             if(colString != "")
529             {
530                 OsmAccess::AccessType type = OsmAccess::parseString(colString);
531                 r_restrictions.addVehicleTypeAccessRestriction(
532                     OsmVehicle::VEHICLE,
533                     type);
534             }
535         }
536     }
537     catch (std::exception& e)
538     {
539         throw MapProviderException(
540             std::string("RestrictionQueries:addPointResultToEdge..: ") + e.what());
541     }
542 }
543 //============================== ACESS      ==================================
544 //============================== INQUIRY    ==================================
```

```
545 ////////////////////////////// PROTECTED  ///////////////////////////////////
546 /////////////////////////////// PRIVATE   ///////////////////////////////////
547 //static
548 std::string
549 RestrictionQueries::startOfQuery(const std::string& rTopoEdgeTable)
550 {
551     return (
552         "SELECT     edge_id, "
553         //-- osm data about original edge
554         "           osm.* "
555         "FROM      " + rTopoEdgeTable +
556         " JOIN ( "
557         "   SELECT  element_id "
558     );
559 }
560 
561 //static
562 std::string
563 RestrictionQueries::queryColumns(const std::vector<std::string>& rCols)
564 {
565     std::ostringstream oss;
566     for(const std::string& col : rCols)
567     {
568         oss << ", " << col;
569     }
570     return oss.str();
571 }
572 
573 //static
574 std::string
575 RestrictionQueries::midOfQuery(
576     const std::string& rSchemaName,
577     const std::string& rOsmEdgeTable)
578 {
579     return (
580         "   FROM    " + rSchemaName + ".relation "
581         "   JOIN    " + rOsmEdgeTable +
582         "   ON      topogeo_id = (topo_geom).id "
583         "   WHERE   highway in " + OsmHighway::typesAsCommaSeparatedString()
584     );
585 }
586 
587 //static
588 std::string
589 RestrictionQueries::notNullColumns(const std::vector<std::string>& rCols)
590 {
591     std::ostringstream oss;
592     oss << " AND (";
593     size_t i = 0;
594     for(const std::string& col : rCols)
595     {
596         oss << col << " IS NOT NULL ";
597         if(i < (rCols.size() - 1))
598         {
599             oss << " OR ";
600         }
601         ++i;
```

```
602     }
603     oss << ") ";
604     return oss.str();
605 }
606 
607 //static
608 std::string
609 RestrictionQueries::endOfQuery()
610 {
611     return (
612         ") AS osm "
613         "ON edge_id = element_id "
614         "ORDER BY edge_id ASC;"
615     );
616 }
```

D.11.10 TopologyQueries.h

```
1  /**  A class for holding static queries about the topology
2   * for the PostGisProvider.
3   *
4   * #include "TopologyQueries.h"
5   *
6   * @author  Jonas Bergman
7   */
8 
9  #ifndef MAPPROVIDER_POSTGIS_TOPOLOGYQUERIES_H_
10 #define MAPPROVIDER_POSTGIS_TOPOLOGYQUERIES_H_
11 
12 // SYSTEM INCLUDES
13 //
14 #include <string>
15 
16 // PROJECT INCLUDES
17 //
18 #include <pqxx/pqxx>
19 
20 // LOCAL INCLUDES
21 //
22 #include "../../graph/Topology.h"
23 #include "../../graph/TopologyException.h"
24 #include "../MapProviderException.h"
25 
26 // FORWARD REFERENCES
27 //
28 
29 /** This class holds static queries about the topology to be used by the
30  * PostGisProvider. It also holds types for handling the results.
31  */
32 class TopologyQueries
33 {
34 public:
35 // TYPES
36 
37     /** Columns used in queries for Vertices. */
38     struct VertexResult
39     {
```

```
40          enum Columns
41          {
42              NODE_ID,
43              X,
44              Y
45          };
46      };
47
48      /** Columns used in queries for Edges. */
49      struct EdgeResult
50      {
51          enum Columns
52          {
53              EDGE_ID,
54              START_NODE,
55              END_NODE,
56              EDGE_LENGTH,
57              CENTER_X,
58              CENTER_Y,
59              SOURCE_BEARING,
60              TARGET_BEARING,
61              OSM_ID,
62              ELEMENT_ID,     // NOT USED: same as EDGE_ID_COL
63
64              // road data
65              HIGHWAY,
66              JUNCTION,
67              LANES,
68              ONEWAY,
69
70              // access
71              ACCESS,
72              MOTORCAR,
73              GOODS,
74              HGV,
75              PSV,
76              LHV,
77              MOTOR_VEHICLE,
78              VEHICLE,
79          };
80      };
81
82
83
84  // LIFECYCLE
85      /** Constructor. */
86      TopologyQueries() = delete;
87      /** Copy constructor. */
88      TopologyQueries(const TopologyQueries& from) = delete;
89
90  // OPERATORS
91  // OPERATIONS
92      /** Fetch the vertices for the topology.
93       * @param   rTrans          Transaction to perform query in.
94       * @param   rResult         Store the result of query here.
95       * @param   rVertexTable    Name of table to fetch topology vertices from.
96       */
```

```
97      static void     getTopologyVertices(
98                          pqxx::transaction_base& rTrans,
99                          pqxx::result&           rResult,
100                         const std::string&      rVertexTable);
101
102     /** Add vertices to topology.
103      * @throws  TopologyException
104      */
105     static void     addVertexResultToTopology(
106                         const pqxx::result&     rResult,
107                         Topology&               rTopology);
108
109     /** Fetch the edges for the topology.
110      * @param   rTrans          Transaction to perform query in.
111      * @param   rResult         Store the result of query here.
112      * @param   rTopoEdgeTable  Name of table to fetch topology edges from.
113      * @param   rSchemaName     Name of topology schema.
114      * @param   rOsmEdgeTable   Name of table with original OSM edge data.
115      */
116     static void     getTopologyEdges(
117                         pqxx::transaction_base& rTrans,
118                         pqxx::result&           rResult,
119                         const std::string&      rTopoEdgeTable,
120                         const std::string&      rSchemaName,
121                         const std::string&      rOsmEdgeTable);
122
123     /** Add edges to topology.
124      * @param   rEdgeResult    Result of db query for edges.
125      * @param   rTopology      Topology to fill with edges.
126      * @throws  TopologyException
127      */
128     static void     addEdgeResultToTopology(
129                         const pqxx::result&     rResult,
130                         Topology&               rTopology);
131
132     /** Helper to add basic data from db to Edge.
133      * @param   rRow        Row with data for an Edge.
134      * @param   rTopology   Topology to add edge to.
135      * @return  Reference to the newly added Edge.
136      * @throws  TopologyException
137      */
138     static Edge&    addBasicResultToEdge(
139                         const pqxx::tuple&  rRow,
140                         Topology&           rTopology);
141
142     /** Add geometric result from query to an Edge.
143      * @param   rEdge   Reference to Edge to set Geom data on.
144      * @param   rRow    Reference to Row with Geom data in it.
145      */
146     static void     addGeomDataResultToEdge(
147                         Edge&               rEdge,
148                         const pqxx::tuple&  rRow);
149
150     /** Add road related result from query to an Edge.
151      * @param   rEdge   Reference to Edge to set road data on.
152      * @param   rRow    Reference to Row with road data in it.
153      */
```

```
154     static void     addRoadDataResultToEdge(
155                         Edge&               rEdge,
156                         const pqxx::tuple&  rRow);
157 
158     /** Extract highway type from database result and store in RoadData.
159      * @param   rRoadData   The RoadData to store in.
160      * @param   rRow    Reference to Row with road data in it.
161      * @throw   MapProviderException
162      */
163     static void     addHighwayTypeToEdgeRoadData(
164                         Edge::RoadData&     rRoadData,
165                         const pqxx::tuple&  rRow);
166 
167     /** Make sure the 'postgis_topology' extension is installed.
168      * @param   rTrans          Transaction to perform query in.
169      */
170     static void     installPostgisTopology(pqxx::transaction_base& rTrans);
171 
172     /** Set schema search path for queries.
173      * @param   rTrans          Transaction to perform query in.
174      */
175     static void     setSearchPath(pqxx::transaction_base& rTrans);
176 
177     /** Create the temporary table for topologies.
178      * @param   rTrans          Transaction to perform query in.
179      * @param   rTableName      Name of the temporary topology table.
180      */
181     static void     createTemporaryTable(
182                         pqxx::transaction_base& rTrans,
183                         const std::string&      rTableName);
184 
185     /** Create a schema for the temporary postgis topology.
186      * @param   rTrans          Transaction to perform query in.
187      * @param   rSchemaName     Name of the temporary schema.
188      * @param   srid            The projection to use.
189      */
190     static void     createTemporarySchema(
191                         pqxx::transaction_base& rTrans,
192                         const std::string&      rSchemaName,
193                         int                     srid);
194 
195     /** Add a column for geometry in the table with Osm Edges.
196      * @param   rTrans          Transaction to perform query in.
197      * @param   rSchemaName     Name of the temporary schema.
198      * @param   rOsmeEdgeTable  Name of the table with OSM edges.
199      */
200     static void     addTopoGeometryColumn(
201                         pqxx::transaction_base& rTrans,
202                         const std::string&      rSchemaName,
203                         const std::string&      rOsmEdgeTable);
204 
205     /** Fill geometry in the table with Osm Edges, using a tolerance for
206      * merging nodes near one another.
207      * @param   rTrans          Transaction to perform query in.
208      * @param   rSchemaName     Name of the temporary schema.
209      * @param   rOsmeEdgeTable  Name of the table with OSM edges.
210      * @param   tolerance       Tolerance in unit of projection.
```

```
211         */
212        static void     fillTopoGeometryColumn(
213                            pqxx::transaction_base& rTrans,
214                            const std::string&      rSchemaName,
215                            const std::string&      rOsmEdgeTable,
216                            double                  tolerance);
217
218        /** Drop the temporary table for topologies.
219         * @param   rTrans          Transaction to perform query in.
220         * @param   rTableName      Name of the temporary topology table.
221         */
222        static void     dropTemporaryTable(
223                            pqxx::transaction_base& rTrans,
224                            const std::string&      rTableName);
225
226        /** Drop the temporary schema for topologies.
227         * @param   rTrans          Transaction to perform query in.
228         * @param   rSchemaName     Name of the temporary topology scheam.
229         */
230        static void     dropTemporarySchema(
231                            pqxx::transaction_base& rTrans,
232                            const std::string&      rTableName);
233
234        /** Clean up in records for postgis topologies.
235         * @param   rTrans          Transaction to perform query in.
236         * @param   rTableName      Name of the temporary topology table.
237         */
238        static void     deleteTemporaryLayerRecord(
239                            pqxx::transaction_base& rTrans,
240                            const std::string&      rTableName);
241
242        /** Clean up in records for postgis topologies.
243         * @param   rTrans          Transaction to perform query in.
244         * @param   rTableName      Name of the temporary topology table.
245         */
246        static void     deleteTemporaryTopoRecord(
247                            pqxx::transaction_base& rTrans,
248                            const std::string&      rSchemaName);
249
250    // ACCESS
251    // INQUIRY
252
253    protected:
254    private:
255    };
256
257    // INLINE METHODS
258    //
259
260    // EXTERNAL REFERENCES
261    //
262
263    #endif /* MAPPROVIDER_POSTGIS_TOPOLOGYQUERIES_H_ */
```

D.11.11 TopologyQueries.cc

```
/*
 * TopologyQueries.cc
 *
 * @author  Jonas Bergman
 */

#include "TopologyQueries.h"  // class implemented

#include "../../osm/OsmHighway.h"


////////////////////////////// PUBLIC /////////////////////////////////////

//============================= LIFECYCLE ====================================
//============================= OPERATORS ====================================
//============================= OPERATIONS ===================================
//static
void
TopologyQueries::getTopologyVertices(
    pqxx::transaction_base& rTrans,
    pqxx::result&           rResult,
    const std::string&      rVertexTable)
{
        rResult = rTrans.exec(
                "SELECT node_id, ST_X(geom) AS x, ST_Y(geom) AS y "
                " FROM " + rVertexTable +
                " ORDER BY node_id ASC;"
        );
}

// static
void
TopologyQueries::addVertexResultToTopology(
    const pqxx::result& rResult,
    Topology&           rTopology)
{
    for(size_t row = 0; row < rResult.size(); ++row)
    {
        VertexIdType id(rResult[row][VertexResult::NODE_ID].as<int>());
        Point point(rResult[row][VertexResult::X].as<double>(),
                    rResult[row][VertexResult::Y].as<double>());
        rTopology.addVertex(id, point);
    }
}

//static
void
TopologyQueries::getTopologyEdges(
    pqxx::transaction_base& rTrans,
    pqxx::result&           rResult,
    const std::string&      rTopoEdgeTable,
    const std::string&      rSchemaName,
    const std::string&      rOsmEdgeTable)
{
    std::string sql(
        "SELECT     edge_id, "
```

```
57          "           start_node, "
58          "           end_node, "
59          //-- geom data about edge
60          "           ST_Length(geom) AS edge_length, "
61          "           ST_X(ST_LineInterpolatePoint(geom, 0.5)) AS center_x, "
62          "           ST_Y(ST_LineInterpolatePoint(geom, 0.5)) AS center_y, "
63          "           (ST_Azimuth("
64          "               ST_PointN(geom,1), "
65          "               ST_PointN(geom,2))/(2*pi())*360)::int "
66          "               AS source_bearing, "
67          "           (ST_Azimuth("
68          "               ST_PointN(geom,ST_NPoints(geom)-1), "
69          "               ST_PointN(geom,ST_NPoints(geom)))/(2*pi())*360)::int "
70          "               AS target_bearing, "
71          //-- osm data about original edge
72          "           osm.* "
73          "FROM      " + rTopoEdgeTable +
74          " JOIN ( "
75          "   SELECT  osm_id, element_id "
76          //-- road data
77          "           , highway "
78          "           , junction "
79          "           , lanes "
80          "           , oneway "
81          "   FROM    " + rSchemaName + ".relation "
82          "   JOIN    " + rOsmEdgeTable+
83          "   ON      topogeo_id = (topo_geom).id "
84          "   WHERE   highway in " + OsmHighway::typesAsCommaSeparatedString() +
85          ") AS osm "
86          "ON edge_id = element_id "
87          "ORDER BY edge_id ASC;"
88      );
89      rResult = rTrans.exec(sql);
90  }
91
92  // static
93  void
94  TopologyQueries::addEdgeResultToTopology(
95      const pqxx::result& rResult,
96      Topology&           rTopology)
97  {
98      for(const pqxx::tuple& row : rResult)
99      {
100         Edge& edge = addBasicResultToEdge(row, rTopology);
101         addGeomDataResultToEdge(edge, row);
102         addRoadDataResultToEdge(edge, row);
103     }
104 }
105
106 // static
107 Edge&
108 TopologyQueries::addBasicResultToEdge(
109     const pqxx::tuple&  rRow,
110     Topology&           rTopology)
111 {
112     EdgeIdType
113         edge_id(rRow[EdgeResult::EDGE_ID].as<EdgeIdType>(Edge::MAX_ID));
```

```
114     OsmIdType
115         osm_id(rRow[EdgeResult::OSM_ID].as<OsmIdType>(Osm::MAX_ID));
116     VertexIdType
117         source_id(rRow[EdgeResult::START_NODE].as<int>(Vertex::MAX_ID));
118     VertexIdType
119         target_id(rRow[EdgeResult::END_NODE].as<int>(Vertex::MAX_ID));
120 
121     Edge& edge = rTopology.addEdge(edge_id, osm_id, source_id, target_id);
122 
123     return edge;
124 }
125 
126 // static
127 void
128 TopologyQueries::addGeomDataResultToEdge(Edge& rEdge, const pqxx::tuple& rRow)
129 {
130     Edge::GeomData gd(
131            rRow[EdgeResult::EDGE_LENGTH].as<double>(0),
132            Point(rRow[EdgeResult::CENTER_X].as<double>(0),
133                  rRow[EdgeResult::CENTER_Y].as<double>(0)),
134            rRow[EdgeResult::SOURCE_BEARING].as<int>(0),
135            rRow[EdgeResult::TARGET_BEARING].as<int>(0));
136     rEdge.setGeomData(gd);
137 }
138 
139 // static
140 void
141 TopologyQueries::addRoadDataResultToEdge(Edge& rEdge, const pqxx::tuple& rRow)
142 {
143     Edge::RoadData rd;
144     std::string
145         onewayStr(rRow[EdgeResult::ONEWAY].as<std::string>("no"));
146 
147     if(rRow[EdgeResult::JUNCTION].as<std::string>("") ==
148         OsmHighway::JUNCTION_ROUNDABOUT)
149     {
150         onewayStr = "yes";
151     }
152     if(onewayStr == "yes")
153     {
154         rd.direction = Edge::DirectionType::FROM_TO;
155     }
156     else if(onewayStr == "-1")
157     {
158         rd.direction = Edge::DirectionType::TO_FROM;
159     }
160 
161     rd.nrLanes = rRow[EdgeResult::LANES].as<size_t>(1);
162 
163     addHighwayTypeToEdgeRoadData(rd, rRow);
164 
165     rEdge.setRoadData(rd);
166 }
167 
168 // static
169 void
170 TopologyQueries::addHighwayTypeToEdgeRoadData(Edge::RoadData& rRoadData,
```

```
171                                                 const pqxx::tuple& rRow)
172 {
173     std::string roadTypeStr( rRow[EdgeResult::HIGHWAY].as<std::string>("road"));
174     try
175     {
176         rRoadData.roadType = OsmHighway::parseString(roadTypeStr);
177     }
178     catch (OsmException& oe)
179     {
180         throw MapProviderException(
181             std::string("TopologyQueries:addHighwayTypeToEdgeRoadData:")
182             + oe.what());
183     }
184 }
185
186 //static
187 void
188 TopologyQueries::installPostgisTopology(pqxx::transaction_base& rTrans)
189 {
190     rTrans.exec(
191             "CREATE EXTENSION IF NOT EXISTS postgis_topology"
192     );
193 }
194
195 //static
196 void
197 TopologyQueries::setSearchPath(pqxx::transaction_base& rTrans)
198 {
199     rTrans.exec(
200             "SET search_path = topology, public"
201     );
202 }
203
204 //static
205 void
206 TopologyQueries::createTemporaryTable(pqxx::transaction_base& rTrans,
207                                       const std::string& rTableName)
208 {
209     rTrans.exec(
210             "CREATE TABLE public." + rTableName + " " +
211             "AS SELECT * "
212             "FROM planet_osm_line "
213             "WHERE highway IS NOT NULL"
214     );
215 }
216
217 //static
218 void
219 TopologyQueries::createTemporarySchema(pqxx::transaction_base& rTrans,
220                                       const std::string& rSchemaName, int srid)
221 {
222     rTrans.exec(
223             "SELECT topology.CreateTopology('" +
224             rSchemaName + "'," +
225             rTrans.quote(srid) + ")"
226     );
227 }
```

```
228
229 //static
230 void
231 TopologyQueries::addTopoGeometryColumn(pqxx::transaction_base& rTrans,
232                                        const std::string& rSchemaName,
233                                        const std::string& rOsmEdgeTable)
234 {
235     rTrans.exec(
236             "SELECT topology.AddTopoGeometryColumn('" +
237             rSchemaName + "', " +
238             "'public', '" +
239             rOsmEdgeTable + "', " +
240             "'topo_geom', 'LINESTRING')"
241     );
242 }
243
244 //static
245 void
246 TopologyQueries::fillTopoGeometryColumn(pqxx::transaction_base& rTrans,
247                                         const std::string& rSchemaName,
248                                         const std::string& rOsmEdgeTable,
249                                         double tolerance)
250 {
251     rTrans.exec(
252             "UPDATE public." +
253             rOsmEdgeTable + " " +
254             "SET topo_geom = topology.toTopoGeom(way, '" +
255             rSchemaName +
256             "', 1, " +
257             rTrans.quote(tolerance) + ")"
258     );
259 }
260
261 //static
262 void
263 TopologyQueries::dropTemporaryTable(pqxx::transaction_base& rTrans,
264                                     const std::string& rTableName)
265 {
266     rTrans.exec(
267             "DROP TABLE IF EXISTS public." + rTableName
268     );
269 }
270
271 //static
272 void
273 TopologyQueries::dropTemporarySchema(pqxx::transaction_base& rTrans,
274                                      const std::string& rSchemaName)
275 {
276     rTrans.exec(
277             "DROP SCHEMA IF EXISTS " + rSchemaName + " CASCADE"
278     );
279 }
280
281 //static
282 void
283 TopologyQueries::deleteTemporaryLayerRecord(pqxx::transaction_base& rTrans,
284                                             const std::string& rTableName)
```

```
285 {
286     rTrans.exec(
287             "DELETE FROM topology.layer "
288             "WHERE table_name = " + rTrans.quote(rTableName)
289     );
290 }
291
292 //static
293 void
294 TopologyQueries::deleteTemporaryTopoRecord(pqxx::transaction_base& rTrans,
295                                            const std::string& rSchemaName)
296 {
297     rTrans.exec(
298             "DELETE FROM topology.topology "
299             "WHERE name = " + rTrans.quote(rSchemaName)
300     );
301 }
302 //============================== ACESS      ====================================
303 //============================== INQUIRY    ====================================
304 ////////////////////////////////// PROTECTED  ////////////////////////////////////
305
306 ////////////////////////////////// PRIVATE    ////////////////////////////////////
```

D.11.12 PostGisProvider_test.cc

```
1  /*
2   * DatabaseHandler_test.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "../../postgis/PostGisProvider.h"
8  #include "../../postgis/RestrictionQueries.h"
9
10 #include <iostream>
11 #include <string>
12 #include <sstream>
13 #include <vector>
14
15 #include "../../../catchtest/catch.hpp"
16 #include "../../../config/ConfigurationReader.h"
17 #include "../../../config/DatabaseConfig.h"
18 #include "../../../util/TimeToStringMaker.h"
19 #include "../../../graph/Edge.h"
20 #include "../../../graph/Vertex.h"
21 #include "../../../graph/GraphBuilder.h"
22
23 SCENARIO ("PostGis topology handling", "[postgis][topology]")
24 {
25     try
26     {
27         // ================================================================
28         GIVEN ("a configuration file with NO topology name")
29         {
30             WHEN ("we try to read in topology")
31             {
32                 std::string config_file(
```

```
33                      "catchtest/testsettings/missing-topo-testsettings.json");
34                  ConfigurationReader config_reader(config_file);
35                  Configuration config;
36                  config_reader.fillConfiguration(config);
37
38                  THEN ("we should get an exception")
39                  {
40                      REQUIRE_THROWS_AS (PostGisProvider pgp(config),
41                          MapProviderException&);
42                  }
43              }
44          }
45      }
46      catch (ConfigurationException& e) {
47          INFO(e.what());
48          REQUIRE (false);    // force output of error and failure
49      }
50  }
51
52
53  SCENARIO ("PostGis queries", "[postgis][query]")
54  {
55      try
56      {
57
58          // ============================================================================
59          GIVEN ("a valid database configuration structure and "
60                  "name to existing topology")
61          {
62              std::string config_file(
63                  "catchtest/testsettings/mikh0522-testsettings.json");
64              ConfigurationReader config_reader(config_file);
65              Configuration config;
66              config_reader.fillConfiguration(config);
67
68              PostGisProvider db_handler(config);
69
70              // ....................................................................
71              WHEN ("we try to fetch a topology")
72              {
73                  Topology topology;
74
75                  THEN ("we should not get an exception")
76                  {
77                      REQUIRE_NOTHROW (db_handler.getTopology(topology););
78                  }
79              }
80
81              // ....................................................................
82              WHEN ("we try to fetch topology ")
83              {
84                  Topology topology;
85                  db_handler.getTopology(topology);
86
87                  THEN ("we should receive a vertices and edges")
88                  {
89                      REQUIRE (topology.nrVertices() > 0);
```

```
90                      REQUIRE (topology.nrEdges() > 0);
91                  }
92              }
93
94              // ..........................................................
95              WHEN ("we try to build a graph ")
96              {
97                  Topology topology;
98                  db_handler.getTopology(topology);
99                  Configuration config;
100                 GraphBuilder graph(topology, config);
101                 std::ostringstream oss;
102
103                 THEN ("we should be able to print some information")
104                 {
105                     graph.printGraphInformation(oss);
106                     INFO(oss.str());
107                     REQUIRE (true);
108                 }
109             }
110
111             // ..........................................................
112             WHEN ("fetching an edge from topology")
113             {
114                 Topology topology;
115                 db_handler.getTopology(topology);
116                 const Edge& edge = topology.getEdge(1);
117
118                 THEN ("we should be able to print it out")
119                 {
120                     INFO (edge);
121                     REQUIRE (true);
122
123                     /* Information matches this query:
124 $ psql -U jonas -d mikh_0522 -c
125 "SELECT edge_id, osm_id, start_node, end_node, lanes, highway
126  FROM   topo_test.edge_data
127  JOIN (
128  SELECT osm_id, element_id, highway, lanes
129  FROM topo_test.relation
130  JOIN highways_test
131  ON topogeo_id = (topo_geom).id )
132  AS osm
133  ON edge_id = element_id
134  WHERE edge_id = 1;"
135
136  edge_id |  osm_id   | start_node | end_node | lanes |   highway
137 ---------+-----------+------------+----------+-------+-------------
138       1 | 124227193 |          1 |       54 |       | residential
139 (1 row)
140
141                      */
142                 }
143             }
144
145         }
146     }
```

```
147     catch (ConfigurationException& e)
148     {
149         INFO(e.what());
150         REQUIRE (false);    // force output of error and failure
151     }
152     catch (MapProviderException& dbe)
153     {
154         INFO(dbe.what());
155         REQUIRE (false);    // force output of error and failure
156 
157     }
158 }
159 
160 SCENARIO ("Set costs on Edges", "[postgis][cost]")
161 {
162     try
163     {
164         // ============================================================
165         GIVEN ("a valid database configuration structure and "
166                "name to existing topology")
167         {
168             std::string config_file(
169                 "catchtest/testsettings/mikh_restr_0617-testsettings.json");
170             ConfigurationReader config_reader(config_file);
171             Configuration config;
172             config_reader.fillConfiguration(config);
173 
174             PostGisProvider pgp(config);
175 
176             Topology topology;
177             pgp.getTopology(topology);
178 
179             // ........................................................
180             WHEN ("we try to set restrictions and costs on topology")
181             {
182                 pgp.setRestrictionsAndCosts(topology);
183 
184                 THEN ("we should be able to read travel time cost on edges")
185                 {
186                     EdgeIdType id = 1;
187                     const Edge& edge = topology.getEdge(id);
188                     INFO ("edge " << id
189                         << ", length: " << edge.geomData().length
190                         << ", travel time: "
191                         << edge.edgeCost().getCost(EdgeCost::TRAVEL_TIME)
192                         << ", total cost: " << edge.cost());
193                     INFO ("edge " << edge);
194                     REQUIRE (edge.cost() > 0);
195                 }
196 
197                 THEN ("we should be able to find cost for barriers")
198                 {
199                     EdgeIdType id = 869;
200                     const Edge& edge = topology.getEdge(id);
201                     INFO ("edge " << id
202                         << ", length: " << edge.geomData().length
203                         << ", travel time: "
```

```
204                         << edge.edgeCost().getCost(EdgeCost::TRAVEL_TIME)
205                         << ", barrier cost: "
206                         << edge.edgeCost().getCost(EdgeCost::BARRIER)
207                         << ", total cost: " << edge.cost());
208                     REQUIRE (edge.cost() > 0);
209                 }
210 
211                 THEN ("we should be able to find cost for other hindrances")
212                 {
213                     EdgeIdType id = 869;
214                     const Edge& edge = topology.getEdge(id);
215                     INFO ("edge " << id
216                         << ", length: " << edge.geomData().length
217                         << ", travel time: "
218                         << edge.edgeCost().getCost(EdgeCost::TRAVEL_TIME)
219                         << ", barrier cost: "
220                         << edge.edgeCost().getCost(EdgeCost::BARRIER)
221                         << ", other cost: "
222                         << edge.edgeCost().getCost(EdgeCost::OTHER)
223                         << ", total cost: " << edge.cost());
224                     REQUIRE (edge.cost() > 0);
225                 }
226             }
227         }
228     }
229     catch (ConfigurationException& e)
230     {
231         INFO(e.what());
232         REQUIRE (false);    // force output of error and failure
233     }
234     catch (MapProviderException& dbe)
235     {
236         INFO(dbe.what());
237         REQUIRE (false);    // force output of error and failure
238     }
239 
240 }
```

D.12 osm

D.12.1 README.md

```
OSM
===

OpenStreetMap related classes and constants are placed in this package.


Relations
---------

There is no really easy way to get to relations if data has been imported with
 ↪ osm2pgsql. Best chance is to import in "slim mode" (with flag `-s`) and look
 ↪ through table `planet_osm_rel` and search the column `tags` for
 ↪ `restriction`. Then parse the `members` column for members of the relation
 ↪ and their roles.
```

```
The TurnRestriction class could be smarter with handling turn either via `nodes`
↪  or `ways` but it is not implemented yet.
```

D.12.2 OsmAccess.h

```
/**  Access to `Access` data from the OSM file.
 *
 * #include "OsmAccess.h"
 *
 * @author  Jonas Bergman
 */

#ifndef OSM_OSMACCESS_H_
#define OSM_OSMACCESS_H_

// SYSTEM INCLUDES
//
#include <algorithm>
#include <initializer_list>
#include <string>
#include <vector>

// PROJECT INCLUDES
//

// LOCAL INCLUDES
//
#include "OsmException.h"

// FORWARD REFERENCES
//

/**
 * Class to represent OSM key `access`.
 */
class OsmAccess
{
public:
// TYPES
    enum AccessType
    {
        YES,
        PRIVATE,
        NO,
        PERMISSIVE,
        AGRICULTURAL,
        USE_SIDEPATH,
        DELIVERY,
        DESIGNATED,
        DISMOUNT,
        DISCOURAGED,
        FORESTRY,
        DESTINATION,
        CUSTOMERS,

        NR_ACCESS_TYPES
    };
```

```
53
54
55      /** Allow access to the types in the 'allowAccessTypes', deny all other.
56       */
57      struct AccessRule
58      {
59          AccessRule() = default;
60          AccessRule(const AccessRule& from) = default;
61          AccessRule(std::initializer_list<AccessType> allowedTypes);
62
63          bool    hasAccess(AccessType type) const;
64
65          std::vector<AccessType> allowAccessToTypes;
66      };
67
68  // LIFECYCLE
69      OsmAccess() = delete;
70      OsmAccess(AccessType type);
71      OsmAccess(const OsmAccess& from) = default;
72      ~OsmAccess() = default;
73
74  // OPERATORS
75  // OPERATIONS
76      /** Attempt to parse a string to a AccessType
77       * @param   rTypeString     String which could contain a Access type
78       * @return  A valid AccessType
79       * @throw   OsmException if invalid string.
80       */
81      static AccessType   parseString(const std::string& rTypeString);
82
83      /** Convert a Access Type to a string representation.
84       * @param   accessType     The type to convert.
85       * @return  string representation of the type.
86       * @throw   OsmException if unknown vehicle type (out of bounds).
87       */
88      static std::string  toString(AccessType accessType);
89
90      /** Convert this AccessType to a string.
91       * @return  string representation of this VehicleType.
92       */
93      std::string         toString() const;
94
95      /** See if this Access type permits access according to rule;
96       * @param   AccessRule
97       * @return  true if access is allowed, false if not
98       */
99      bool                allowsAccess(AccessRule rule) const;
100
101 // ACCESS
102     /**
103      * @return  The access type.
104      */
105     AccessType          accessType() const;
106 // INQUIRY
107 protected:
108 private:
109     AccessType              mType {YES};
```

```
110     static const std::string sTypeStrings[];
111 };
112 
113 // INLINE METHODS
114 //
115 
116 // EXTERNAL REFERENCES
117 //
118 
119 #endif /* OSM_OSMACCESS_H_ */
```

D.12.3 OsmAccess.cc

```
1  /*
2   * OsmAccess.cc
3   *
4   * @author  Jonas Bergman
5   */
6  
7  
8  #include "OsmAccess.h"  // class implemented
9  
10 // AccessRule ----------------------------------------------------------------
11 OsmAccess::AccessRule::AccessRule(
12     std::initializer_list<OsmAccess::AccessType> allowedTypes)
13     : allowAccessToTypes(allowedTypes)
14 {
15 }
16 
17 bool
18 OsmAccess::AccessRule::hasAccess(OsmAccess::AccessType type) const
19 {
20     auto it = std::find(allowAccessToTypes.begin(),
21         allowAccessToTypes.end(),
22         type);
23     return it != allowAccessToTypes.end();
24 }
25 
26 
27 ////////////////////////////// PUBLIC //////////////////////////////////////
28 
29 //============================= LIFECYCLE ====================================
30 OsmAccess::OsmAccess(OsmAccess::AccessType type)
31     : mType(type)
32 {}
33 
34 //============================= OPERATORS ====================================
35 
36 //static
37 OsmAccess::AccessType
38 OsmAccess::parseString(const std::string& rTypeString)
39 {
40     for(size_t i = 0; i < NR_ACCESS_TYPES; ++i)
41     {
42         if(rTypeString == OsmAccess::sTypeStrings[i])
43         {
44             return static_cast<AccessType>(i);
```

```
45          }
46      }
47      throw OsmException("OsmAccess:parseString: Unknown Access Type string.");
48  }
49
50  //static
51  std::string
52  OsmAccess::toString(OsmAccess::AccessType accessType)
53  {
54      if(accessType >= NR_ACCESS_TYPES)
55      {
56          throw OsmException("OsmAccess:toString: Unknown Access Type");
57      }
58      return OsmAccess::sTypeStrings[accessType];
59  }
60
61  std::string
62  OsmAccess::toString() const
63  {
64      return sTypeStrings[this->mType];
65  }
66
67  bool
68  OsmAccess::allowsAccess(OsmAccess::AccessRule rule) const
69  {
70      return rule.hasAccess(mType);
71  }
72
73  OsmAccess::AccessType
74  OsmAccess::accessType() const
75  {
76      return mType;
77  }
78
79  //============================== OPERATIONS ====================================
80  //============================== ACESS      ====================================
81  //============================== INQUIRY    ====================================
82  ////////////////////////////////// PROTECTED  ////////////////////////////////////
83
84  ////////////////////////////////// PRIVATE    ////////////////////////////////////
85  const std::string OsmAccess::sTypeStrings[] =
86  {
87      "yes",
88      "private",
89      "no",
90      "permissive",
91      "agricultural",
92      "use_sidepath",
93      "delivery",
94      "designated",
95      "dismount",
96      "discouraged",
97      "forestry",
98      "destination",
99      "customers"
100 };
```

D.12.4 OsmBarrier.h

```
/**  Access to `Barrier` data from the OSM file.
 *
 * #include "OsmBarrier.h"
 *
 * @author  Jonas Bergman
 */

#ifndef OSM_OSMBARRIER_H_
#define OSM_OSMBARRIER_H_

// SYSTEM INCLUDES
//
#include <algorithm>
#include <initializer_list>
#include <map>
#include <string>
#include <vector>

// PROJECT INCLUDES
//

// LOCAL INCLUDES
//
#include "OsmException.h"
#include "../graph/Cost.h"

// FORWARD REFERENCES
//


/**
 * Class to represent OSM key `barrier`.
 */
class OsmBarrier
{
public:
// TYPES
    enum BarrierType
    {
        NONE,
        BLOCK,
        BOLLARD,
        BORDER_CONTROL,
        BUMP_GATE,
        BUS_TRAP,
        CATTLE_GRID,
        CHAIN,
        CYCLE_BARRIER,
        DEBRIS,
        ENTRANCE,
        FULLHEIGHT_TURNSTILE,
        GATE,
        HAMPSHIRE_GATE,
        HEIGHT_RESTRICTOR,
        HORSE_STILE,
        JERSEY_BARRIER,
```

```
57          KENT_CARRIAGE_GAP,
58          KISSING_GATE,
59          LIFT_GATE,
60          LOG,
61          MOTORCYCLE_BARRIER,
62          ROPE,
63          SALLY_PORT,
64          SPIKES,
65          STILE,
66          SUMP_BUSTER,
67          SWING_GATE,
68          TOLL_BOOTH,
69          TURNSTILE,
70          YES,
71
72          NR_BARRIER_TYPES
73      };
74
75      /** Barriers which imposes restrictions on access.
76       */
77      struct RestrictionsRule
78      {
79          RestrictionsRule() = default;
80          RestrictionsRule(const RestrictionsRule& from) = default;
81          RestrictionsRule(std::initializer_list<BarrierType> restrictionTypes);
82
83          bool    restrictsAccess(BarrierType type) const;
84
85          std::vector<BarrierType> restrictionTypes;
86      };
87
88      /** Barriers which infer costs.
89       */
90      struct CostsRule
91      {
92          CostsRule() = default;
93          CostsRule(const CostsRule& from) = default;
94
95          bool        costsToPass(BarrierType type) const;
96          Cost        getCost(BarrierType type) const;
97          void        addCost(BarrierType type, Cost cost);
98
99          std::map<BarrierType, Cost> costs;
100     };
101
102 // LIFECYCLE
103     OsmBarrier() = delete;
104     OsmBarrier(BarrierType type);
105     OsmBarrier(const OsmBarrier& from) = default;
106     ~OsmBarrier() = default;
107
108 // OPERATORS
109 // OPERATIONS
110     /** Attempt to parse a string to a BarrierType
111      * @param   rTypeString     String which could contain a Barrier type
112      * @return  A valid BarrierType
113      * @throw   OsmException if invalid string.
```

```
114         */
115        static BarrierType  parseString(const std::string& rTypeString);
116
117        /** Convert a Barrier Type to a string representation.
118         * @param   barrierType     The type to convert.
119         * @return  string representation of the type.
120         * @throw   OsmException if unknown barrier type (out of bounds).
121         */
122        static std::string  toString(BarrierType barrierType);
123
124        /** Convert this BarrierType to a string.
125         * @return  string representation of this VehicleType.
126         */
127        std::string         toString() const;
128
129        /** See if this Barrier type permits access according to rule;
130         * @param   RestrictionRule
131         * @return  true if access is allowed, false if not
132         */
133        bool                restrictsAccess(RestrictionsRule rule) const;
134
135        /** See if this Barrier type costs to pass according to rule;
136         * @param   RestrictionRule
137         * @return  true if access is allowed, false if not
138         */
139        bool                costsToPass(CostsRule rule) const;
140
141    // ACCESS
142    // INQUIRY
143    protected:
144    private:
145        BarrierType                             mType {NONE};
146        static const std::vector<std::string> sTypeStrings;
147        static const std::vector<std::string> sDisregardedTypes;
148    };
149
150    // INLINE METHODS
151    //
152
153    // EXTERNAL REFERENCES
154    //
155
156    #endif /* OSM_OSMBARRIER_H_ */
```

D.12.5 OsmBarrier.cc

```
1   /*
2    * OsmBarrier.cc
3    *
4    * @author  Jonas Bergman
5    */
6
7
8   #include "OsmBarrier.h"  // class implemented
9
10  // RestrictionsRule ----------------------------------------------------------
11  OsmBarrier::RestrictionsRule::RestrictionsRule(
```

```
12      std::initializer_list<OsmBarrier::BarrierType> restrictionTypes)
13      : restrictionTypes(restrictionTypes)
14  { }
15
16  bool
17  OsmBarrier::RestrictionsRule::restrictsAccess(OsmBarrier::BarrierType type) const
18  {
19      auto it = std::find(restrictionTypes.begin(), restrictionTypes.end(), type);
20      return it != restrictionTypes.end();
21  }
22
23  bool
24  OsmBarrier::CostsRule::costsToPass(OsmBarrier::BarrierType type) const
25  {
26      const auto& it  = costs.find(type);
27      return it != costs.end();
28  }
29
30  Cost
31  OsmBarrier::CostsRule::getCost(OsmBarrier::BarrierType type) const
32  {
33      const auto& it  = costs.find(type);
34      if(it != costs.end())
35      {
36          return it->second;
37      }
38      else
39      {
40          return 0;
41      }
42  }
43
44  void
45  OsmBarrier::CostsRule::addCost(OsmBarrier::BarrierType type, Cost cost)
46  {
47      costs.erase(type);
48      costs.insert({type, cost});
49  }
50
51  ////////////////////////////// PUBLIC //////////////////////////////////////
52
53  //============================ LIFECYCLE ====================================
54  OsmBarrier::OsmBarrier(OsmBarrier::BarrierType type)
55      : mType(type)
56  {}
57
58  //============================ OPERATORS ====================================
59
60  //static
61  OsmBarrier::BarrierType
62  OsmBarrier::parseString(const std::string& rTypeString)
63  {
64      for(size_t i = 0; i < sTypeStrings.size(); ++i)
65      {
66          if(rTypeString == OsmBarrier::sTypeStrings[i])
67          {
68              return static_cast<BarrierType>(i);
```

```
69          }
70      }
71      // no match in types. Look if it is disregarded or unknown.
72      auto it = std::find(sDisregardedTypes.begin(),
73                          sDisregardedTypes.end(),
74                          rTypeString);
75      if(it != sDisregardedTypes.end())
76      {
77          return BarrierType::NONE;
78      }
79      throw OsmException("OsmBarrier:parseString: Unknown Barrier Type string: "
80          + rTypeString);
81  }
82
83  //static
84  std::string
85  OsmBarrier::toString(OsmBarrier::BarrierType accessType)
86  {
87      if(accessType >= sTypeStrings.size())
88      {
89          throw OsmException("OsmBarrier:toString: Unknown Barrier Type");
90      }
91      return OsmBarrier::sTypeStrings[accessType];
92  }
93
94  std::string
95  OsmBarrier::toString() const
96  {
97      return sTypeStrings[this->mType];
98  }
99
100 bool
101 OsmBarrier::restrictsAccess(OsmBarrier::RestrictionsRule rule) const
102 {
103     return rule.restrictsAccess(mType);
104 }
105
106 bool
107 OsmBarrier::costsToPass(OsmBarrier::CostsRule rule) const
108 {
109     return rule.costsToPass(mType);
110 }
111
112 //============================== OPERATIONS ====================================
113 //============================== ACESS      ====================================
114 //============================== INQUIRY    ====================================
115 ////////////////////////////// PROTECTED  ////////////////////////////////////
116
117 ////////////////////////////// PRIVATE    ////////////////////////////////////
118 const std::vector<std::string> OsmBarrier::sTypeStrings
119 {
120     "none",
121     "block",
122     "bollard",
123     "border_control",
124     "bump_gate",
125     "bus_trap",
```

```
126     "cattle_grid",
127     "chain",
128     "cycle_barrier",
129     "debris",
130     "entrance",
131     "full-height_turnstile",
132     "gate",
133     "hampshire_gate",
134     "height_restrictor",
135     "horse_stile",
136     "jersey_barrier",
137     "kent_carriage_gap",
138     "kissing_gate",
139     "lift_gate",
140     "log",
141     "motorcycle_barrier",
142     "rope",
143     "sally_port",
144     "spikes",
145     "stile",
146     "sump_buster",
147     "swing_gate",
148     "toll_booth",
149     "turnstile",
150     "yes"
151 };
152
153 const std::vector<std::string> OsmBarrier::sDisregardedTypes
154 {
155     "cable_barrier",
156     "city_wall",
157     "ditch",
158     "fence",
159     "guard_rail",
160     "handrail",
161     "hedge",
162     "kerb",
163     "retaining_wall",
164     "wall",
165 };
```

D.12.6 OsmException.h

```
1  /**  Exception thrown in the 'osm' package.
2   *
3   * #include "OsmException.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef OSM_OSMEXCEPTION_H_
9  #define OSM_OSMEXCEPTION_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <exception>
14 #include <string>
```

```
15
16  // PROJECT INCLUDES
17  //
18
19  // LOCAL INCLUDES
20  //
21
22  // FORWARD REFERENCES
23  //
24
25  /**
26   * Exception to throw in the 'osm' package.
27   * More information of the type of exception is given in the 'what()' message.
28   */
29  class OsmException : public std::exception
30  {
31  public:
32  // LIFECYCLE
33      /** Default constructor.
34       */
35      OsmException() = delete;
36
37      /** Constructor taking a message to display.
38       *
39       * @param   message     The message to prepend when 'what()' is called.
40       */
41      OsmException(const std::string& rMessage) noexcept
42          : std::exception(), mMessage(rMessage)
43      {}
44
45  // OPERATORS
46  // OPERATIONS
47  // ACCESS
48  // INQUIRY
49      const char* what() const noexcept
50      { return (mMessage.c_str()); }
51
52  protected:
53  private:
54  // ATTRIBUTES
55      std::string     mMessage;
56  };
57
58  // INLINE METHODS
59  //
60
61  // EXTERNAL REFERENCES
62  //
63
64  #endif /* OSM_OSMEXCEPTION_H_ */
```

D.12.7 OsmHighway.h

```
1  /**  Access to `Highway` data from the OSM file.
2   *
3   * #include "OsmHighway.h"
4   *
```

```
 * @author Jonas Bergman
 */

#ifndef OSM_OSMHIGHWAY_H_
#define OSM_OSMHIGHWAY_H_

// SYSTEM INCLUDES
//
#include <string>
#include <sstream>
#include <vector>

// PROJECT INCLUDES
//

// LOCAL INCLUDES
//
#include "OsmException.h"

// FORWARD REFERENCES
//


/**
 * Class for categoreis of OSM `highway` and `surface`.
 */
class OsmHighway
{
public:
// TYPES
    enum HighwayType
    {
        // roads
        MOTORWAY,
        MOTORWAY_LINK,
        TRUNK,
        TRUNK_LINK,
        PRIMARY,
        PRIMARY_LINK,
        SECONDARY,
        SECONDARY_LINK,
        TERTIARY,
        TERTIARY_LINK,
        UNCLASSIFIED,
        RESIDENTIAL,
        SERVICE,

        // special types
        LIVING_STREET,
        BUS_GUIDEWAY,
        ROAD,

        NR_HIGHWAY_TYPES
    };

    enum SurfaceType
    {
```

```
62          PAVED,
63          ASPHALT,
64          COBBLESTONE,
65          COBBLESTONE_FLATTENED,
66          SETT,
67          CONCRETE,
68          CONCRETE_LANES,
69          CONCRETE_PLATES,
70          PAVING_STONES,
71          METAL,
72          WOOD,
73
74          UNPAVED,
75          COMPACTED,
76          DIRT,
77          EARTH,
78          FINE_GRAVEL,
79          GRASS,
80          GRASS_PAVER,
81          GRAVEL,
82          GROUND,
83          ICE,
84          MUD,
85          PEBBLESTONE,
86          SALT,
87          SAND,
88          SNOW,
89          WOODCHIPS,
90
91          METAL_GRID,
92
93          NR_SURFACE_TYPES
94      };
95
96      enum JunctionType
97      {
98          ROUNDABOUT
99      };
100     static constexpr const char* JUNCTION_ROUNDABOUT = "roundabout";
101
102 // LIFECYCLE
103     OsmHighway() = delete;
104     OsmHighway(HighwayType type);
105     OsmHighway(const OsmHighway& from) = default;
106     ~OsmHighway() = default;
107
108 // OPERATORS
109 // OPERATIONS
110     /** Attempt to parse a string to a HighwayType
111      * @param   rTypeString     String which could contain a Highway type
112      * @return  A valid HighwayType
113      * @throw   OsmException if invalid string.
114      */
115     static HighwayType  parseString(const std::string& rTypeString);
116
117     /** Attempt to parse a string to a SurfaceType
118      * @param   rTypeString     String which could contain a Surface type
```

```
119         * @return  A valid SurfaceType
120         * @throw   OsmException if invalid string.
121         */
122        static SurfaceType parseSurfaceString(const std::string& rSurfaceString);
123
124        /** Convert a Highway Type to a string representation.
125         * @param   highwayType     The type to convert.
126         * @return  string representation of the type.
127         * @throw   OsmException if unknown highway type (out of bounds).
128         */
129        static std::string  toString(HighwayType highwayType);
130
131        /** Convert a SurfaceType to a string representation.
132         * @param   surfaceType     The type to convert.
133         * @return  string representation of the type.
134         * @throw   OsmException if unknown highway type (out of bounds).
135         */
136        static std::string  toSurfaceString(SurfaceType surfaceType);
137
138        /** Convert this HighwayType to a string.
139         * @return  string representation of this HighwayType.
140         */
141        std::string         toString() const;
142
143    // ACCESS
144        /**
145         * @return A vector of all types as strings.
146         */
147        static const std::vector<std::string>& typeStrings();
148
149        /**
150         * @return A vector of all surface types as strings.
151         */
152        static const std::vector<std::string>& surfaceTypeStrings();
153
154        /** Return "(motorway, trunk....)".
155         * @return A string of all types, comma separated, with parentheses round.
156         */
157        static std::string typesAsCommaSeparatedString();
158
159    // INQUIRY
160    protected:
161    private:
162        HighwayType                             mType {ROAD};
163        static const std::vector<std::string>   sTypeStrings;
164        static const std::vector<std::string>   sSurfaceTypeStrings;
165    };
166
167    // INLINE METHODS
168    //
169
170    // EXTERNAL REFERENCES
171    //
172
173    #endif /* OSM_OSMHIGHWAY_H_ */
```

D.12.8 OsmHighway.cc

```
/*
 * OsmHighway.cc
 *
 * @author  Jonas Bergman
 */

#include "OsmHighway.h"  // class implemented

////////////////////////////// PUBLIC ////////////////////////////////////

//============================ LIFECYCLE ====================================
OsmHighway::OsmHighway(OsmHighway::HighwayType type)
    : mType(type)
{}

//============================ OPERATORS ====================================
//============================ OPERATIONS ===================================
//static
OsmHighway::HighwayType
OsmHighway::parseString(const std::string& rTypeString)
{
    for(size_t i = 0; i < NR_HIGHWAY_TYPES; ++i)
    {
        if(rTypeString == OsmHighway::sTypeStrings[i])
        {
            return static_cast<HighwayType>(i);
        }
    }
    throw OsmException("OsmHighway:parseString: Unknown Highway Type string.");
}

//static
OsmHighway::SurfaceType
OsmHighway::parseSurfaceString(const std::string& rSurfaceString)
{
    for(size_t i = 0; i < NR_SURFACE_TYPES; ++i)
    {
        if(rSurfaceString == OsmHighway::sSurfaceTypeStrings[i])
        {
            return static_cast<SurfaceType>(i);
        }
    }
    throw OsmException("OsmHighway:parseSurfaceString: Unknown Surface Type string.");
}

//static
std::string
OsmHighway::toString(OsmHighway::HighwayType highwayType)
{
    if(highwayType >= NR_HIGHWAY_TYPES)
    {
        throw OsmException("OsmHighway:toString: Unknown Highway Type");
    }
    return OsmHighway::sTypeStrings[highwayType];
}

```

```
57  //static
58  std::string
59  OsmHighway::toSurfaceString(OsmHighway::SurfaceType surfaceType)
60  {
61      if(surfaceType >= NR_SURFACE_TYPES)
62      {
63          throw OsmException("OsmHighway:toSurfaceString: Unknown Surface Type");
64      }
65      return OsmHighway::sSurfaceTypeStrings[surfaceType];
66  }
67
68  std::string
69  OsmHighway::toString() const
70  {
71      return sTypeStrings[this->mType];
72  }
73
74  //============================== ACESS      ===================================
75  //static
76  const std::vector<std::string>&
77  OsmHighway::typeStrings()
78  {
79      return OsmHighway::sTypeStrings;
80  }
81
82  //static
83  const std::vector<std::string>&
84  OsmHighway::surfaceTypeStrings()
85  {
86      return OsmHighway::sSurfaceTypeStrings;
87  }
88
89  // static
90  std::string
91  OsmHighway::typesAsCommaSeparatedString()
92  {
93      std::string cols;
94      std::stringstream ss;
95      ss << "(";
96      for(size_t i = 0; i < sTypeStrings.size(); ++i)
97      {
98          ss << "'" << sTypeStrings[i] << "'";
99          if(i < sTypeStrings.size() - 1)
100         {
101             ss << ", ";
102         }
103     }
104     ss << ")";
105     return ss.str();
106 }
107 //============================== INQUIRY    ===================================
108 ////////////////////////////////// PROTECTED  ////////////////////////////////////
109
110 ////////////////////////////////// PRIVATE    ////////////////////////////////////
111 const std::vector<std::string> OsmHighway::sTypeStrings
112 {
113     "motorway",
```

```
114     "motorway_link",
115     "trunk",
116     "trunk_link",
117     "primary",
118     "primary_link",
119     "secondary",
120     "secondary_link",
121     "tertiary",
122     "tertiary_link",
123     "unclassified",
124     "residential",
125     "service",
126 
127     "living_street",
128     "bus_guideway",
129     "road"
130 };
131 
132 const std::vector<std::string> OsmHighway::sSurfaceTypeStrings
133 {
134     "paved",
135     "asphalt",
136     "cobblestone",
137     "cobblestone:flattened",
138     "sett",
139     "concrete",
140     "concrete:lanes",
141     "concrete:plates",
142     "paving_stones",
143     "metal",
144     "wood",
145 
146     "unpaved",
147     "compacted",
148     "dirt",
149     "earth",
150     "fine_gravel",
151     "grass",
152     "grass_paver",
153     "gravel",
154     "ground",
155     "ice",
156     "mud",
157     "pebblestone",
158     "salt",
159     "sand",
160     "snow",
161     "woodchips",
162 
163     "metal_grid"
164 };
```

D.12.9 OsmId.h

```
1 /*
2  * OsmId.h
3  *
```

```
 * @author  Jonas Bergman
 */

#ifndef OSM_OSMID_H_
#define OSM_OSMID_H_

#include <limits>

typedef long long OsmIdType;

struct Osm
{
    static const OsmIdType MAX_ID;
};



#endif /* OSM_OSMID_H_ */
```

D.12.10 OsmId.cc

```
/*
 * OsmId.cc
 *
 * @author  Jonas Bergman
 */

#include "OsmId.h"

const OsmIdType Osm::MAX_ID = std::numeric_limits<OsmIdType>::max();
```

D.12.11 OsmTurningRestriction.h

```
/**  Access to Turning restriction data from the OSM file.
 *
 * #include "OsmTurningRestriction.h"
 *
 * @author  Jonas Bergman
 */

#ifndef OSM_OSMTURNINGRESTRICTION_H_
#define OSM_OSMTURNINGRESTRICTION_H_

// SYSTEM INCLUDES
//
#include <algorithm>
#include <initializer_list>
#include <sstream>
#include <string>
#include <vector>

// PROJECT INCLUDES
//

// LOCAL INCLUDES
//
#include "OsmException.h"
```

```
25 #include "../graph/Edge.h"
26 #include "../graph/Vertex.h"
27
28 // FORWARD REFERENCES
29 //
30
31 /**
32  * Class for working with "turning restrictions" from OSM relations.
33  */
34 class OsmTurningRestriction
35 {
36 public:
37 // TYPES
38     enum TurningRestrictionType
39     {
40         NONE,
41         NO_LEFT_TURN,
42         NO_RIGHT_TURN,
43         NO_STRAIGHT_ON,
44         NO_U_TURN,
45         ONLY_RIGHT_TURN,
46         ONLY_LEFT_TURN,
47         ONLY_STRAIGHT_ON,
48         NO_ENTRY,
49         NO_EXIT,
50
51         NR_TURNING_RESTRICTION_TYPES
52     };
53
54     enum TurningViaType
55     {
56         VIA_NODE,
57         VIA_WAY
58     };
59
60 // LIFECYCLE
61     /** Constructor. Disabled*/
62     OsmTurningRestriction() = delete;
63
64     /** Constructor.
65      * Turning restriction via a vertex.
66      * @param   type        The type of turning restriction.
67      * @param   fromEdgeId  The Edge the turn starts at.
68      * @param   viaVertexId The Vertex the turn travels via.
69      * @param   toEdgeId    The Edge the turn ends at.
70      */
71     OsmTurningRestriction(TurningRestrictionType    type,
72                           EdgeIdType                fromEdgeId,
73                           VertexIdType              viaVertexId,
74                           EdgeIdType                toEdgeId);
75
76     /** Constructor.
77      * Turning restriction via  other Edges.
78      * @param   type        The type of turning restriction.
79      * @param   fromEdgeId  The Edge the turn starts at.
80      * @param   viaOsmIds   String with the OsmIds of roads the turn travels via.
81      * @param   toEdgeId    The Edge the turn ends at.
```

```
82          */
83         OsmTurningRestriction(TurningRestrictionType    type,
84                               EdgeIdType                fromEdgeId,
85                               std::string               viaOsmIds,
86                               EdgeIdType                toEdgeId);
87
88         /** Copy constructor. */
89         OsmTurningRestriction(const OsmTurningRestriction& from) = default;
90
91         /** Destructor. */
92         ~OsmTurningRestriction() = default;
93
94     // OPERATORS
95     // OPERATIONS
96         /** Attempt to parse a string to a TurningRestrictionType
97          * @param   rTypeString     String which could contain a Turning Restriction
98          * @return  A valid TurningRestrictionType
99          * @throw   OsmException if invalid string.
100         */
101        static TurningRestrictionType  parseString(const std::string& rTypeString);
102
103        /** Convert a Turning Restriction Type to a string representation.
104         * @param   turnRestrictionType     The type to convert.
105         * @return  string representation of the type.
106         * @throw   OsmException if unknown turn restriction type (out of bounds).
107         */
108        static std::string  toString(TurningRestrictionType turnRestrictionType);
109
110        /** Convert this TurningRestriction to a string.
111         * @return  string representation of this turning restriction.
112         */
113        std::string         toString() const;
114
115        /** Convert this TurningRestrictions type to a string.
116         * @return  string representation of this turning restriction type.
117         */
118        std::string         typeToString() const;
119
120    // ACCESS
121        /**
122         * @return  The Edge Id of the 'from' edge
123         */
124        EdgeIdType          fromEdgeId() const;
125
126        /**
127         * @return  The Via type, 'way' or 'node'.
128         */
129        TurningViaType      viaType() const;
130
131        /**
132         * @return  The Ids of the Edges in a 'via way' relation.
133         */
134        std::string         viaOsmIds() const;
135
136        /**
137         * @return  The Vertex Id of the 'via' vertex.
138         */
```

```
139     VertexIdType        viaVertexId() const;
140
141     /**
142      * @return  The Edge id of the 'to' Edge.
143      */
144     EdgeIdType          toEdgeId() const;
145
146 // INQUIRY
147     /** Check if an Edge is in this restriction.
148      * @param   Edge Id to check.
149      * @return  true if the edge is part of this restriction, false if not.
150      */
151     bool                isInRestriction(EdgeIdType edgeId) const;
152
153     /** Check if Travel from 'from' to 'to' is restricted.
154      * @param   fromEdgeId    Travel from edge.
155      * @param   toEdgeId      Travel to edge.
156      * @return  true if travle is restricted.
157      */
158     bool                isRestricted(
159                             EdgeIdType fromEdgeId,
160                             EdgeIdType toEdgeId) const;
161
162 protected:
163 private:
164     TurningRestrictionType              mType {NONE};
165     EdgeIdType                          mFromEdgeId;
166     TurningViaType                      mViaType {VIA_NODE};
167     std::string                         mViaOsmIds;
168     VertexIdType                        mViaVertexId;
169     EdgeIdType                          mToEdgeId;
170     static std::vector<std::string>     sTypeStrings;
171 };
172
173 // INLINE METHODS
174 //
175
176 // EXTERNAL REFERENCES
177 //
178
179 #endif /* OSM_OSMTURNINGRESTRICTION_H_ */
```

D.12.12 OsmTurningRestriction.cc

```
1  /*
2   * OsmTurningRestriction.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "OsmTurningRestriction.h"  // class implemented
8
9  ////////////////////////////// PUBLIC //////////////////////////////////////
10
11 //============================ LIFECYCLE ====================================
12 OsmTurningRestriction::OsmTurningRestriction(
13     OsmTurningRestriction::TurningRestrictionType   type,
```

```
14      EdgeIdType                                          fromEdgeId,
15      VertexIdType                                        viaVertexId,
16      EdgeIdType                                          toEdgeId)
17      : mType(type),
18        mFromEdgeId(fromEdgeId),
19        mViaType(VIA_NODE),
20        mViaOsmIds(),
21        mViaVertexId(viaVertexId),
22        mToEdgeId(toEdgeId)
23  {}
24
25  OsmTurningRestriction::OsmTurningRestriction(
26      OsmTurningRestriction::TurningRestrictionType   type,
27      EdgeIdType                                      fromEdgeId,
28      std::string                                     viaOsmIds,
29      EdgeIdType                                      toEdgeId)
30      : mType(type),
31        mFromEdgeId(fromEdgeId),
32        mViaType(VIA_WAY),
33        mViaOsmIds(viaOsmIds),
34        mViaVertexId(),
35        mToEdgeId(toEdgeId)
36  {}
37
38  //============================== OPERATORS ==================================
39  //============================== OPERATIONS =================================
40  //static
41  OsmTurningRestriction::TurningRestrictionType
42  OsmTurningRestriction::parseString(const std::string& rTypeString)
43  {
44      for(size_t i = 0; i < sTypeStrings.size(); ++i)
45      {
46          if(rTypeString == OsmTurningRestriction::sTypeStrings[i])
47          {
48              return static_cast<TurningRestrictionType>(i);
49          }
50      }
51      throw OsmException(
52          "OsmTurningRestriction:parseString: "
53          "Unknown TurningRestriction Type string.");
54  }
55
56  //static
57  std::string
58  OsmTurningRestriction::toString(
59      OsmTurningRestriction::TurningRestrictionType turnRestrictionType)
60  {
61      if(turnRestrictionType >= sTypeStrings.size())
62      {
63          throw OsmException(
64              "OsmTurningRestriction:toString: Unknown TurningRestriction Type");
65      }
66      return OsmTurningRestriction::sTypeStrings[turnRestrictionType];
67  }
68
69  std::string
70  OsmTurningRestriction::toString() const
```

```
71  {
72      std::ostringstream oss;
73      oss << sTypeStrings[this->mType] << ": "
74          << "from: " << mFromEdgeId;
75
76      // via vertex
77      if(mViaType == VIA_NODE)
78      {
79          oss << ", via vertex: " << mViaVertexId;
80      }
81      else // via edges
82      {
83          oss << ", via edges: [" << mViaOsmIds << "]";
84      }
85
86      oss << ", to: " << mToEdgeId;
87
88      return oss.str();
89  }
90
91  std::string
92  OsmTurningRestriction::typeToString() const
93  {
94      return sTypeStrings[this->mType];
95  }
96  //============================== ACESS      ===================================
97  EdgeIdType
98  OsmTurningRestriction::fromEdgeId() const
99  {
100     return mFromEdgeId;
101 }
102
103 OsmTurningRestriction::TurningViaType
104 OsmTurningRestriction::viaType() const
105 {
106     return mViaType;
107 }
108
109 std::string
110 OsmTurningRestriction::viaOsmIds() const
111 {
112     return mViaOsmIds;
113 }
114
115 VertexIdType
116 OsmTurningRestriction::viaVertexId() const
117 {
118     return mViaVertexId;
119 }
120
121 EdgeIdType
122 OsmTurningRestriction::toEdgeId() const
123 {
124     return mToEdgeId;
125 }
126 //============================== INQUIRY    ===================================
127 bool
```

```
128 OsmTurningRestriction::isInRestriction(EdgeIdType edgeId) const
129 {
130     if(edgeId == mFromEdgeId
131     || edgeId == mToEdgeId)
132     {
133         return true;
134     }
135     return false;
136 }
137
138 bool
139 OsmTurningRestriction::isRestricted(
140     EdgeIdType fromEdgeId,
141     EdgeIdType toEdgeId) const
142 {
143     if(mFromEdgeId == fromEdgeId && mToEdgeId == toEdgeId)
144     {
145         if(mType == NO_LEFT_TURN
146         || mType == NO_RIGHT_TURN
147         || mType == NO_STRAIGHT_ON
148         || mType == NO_U_TURN
149         || mType == NO_ENTRY
150         || mType == NO_EXIT)
151         {
152             return true;
153         }
154     }
155     return false;
156 }
157 ////////////////////////////// PROTECTED  ///////////////////////////////////
158
159 ////////////////////////////// PRIVATE    ///////////////////////////////////
160 // static
161 std::vector<std::string> OsmTurningRestriction::sTypeStrings
162 {
163     "none",
164     "no_left_turn",
165     "no_right_turn",
166     "no_straight_on",
167     "no_u_turn",
168     "only_right_turn",
169     "only_left_turn",
170     "only_straight_on",
171     "no_entry",
172     "no_exit"
173 };
```

D.12.13 OsmVehicle.h

```
1 /**  Access to Vehicle data from the OSM file.
2  *
3  * #include "OsmVehicle.h"
4  *
5  * @author  Jonas Bergman
6  */
7
8 #ifndef OSM_OSMVEHICLE_H_
```

```
9  #define OSM_OSMVEHICLE_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <string>
14
15 // PROJECT INCLUDES
16 //
17
18 // LOCAL INCLUDES
19 //
20 #include "OsmException.h"
21
22 // FORWARD REFERENCES
23 //
24
25 /**
26  * Class for working with different categories of vehicles.
27  */
28 class OsmVehicle
29 {
30 public:
31 // TYPES
32     enum VehicleType
33     {
34         MOTORCAR,
35         GOODS,
36         HGV,
37         PSV,
38         LHV,
39         MOTOR_VEHICLE,
40         VEHICLE,
41
42         NR_VEHICLE_TYPES
43     };
44
45 // LIFECYCLE
46     OsmVehicle() = delete;
47     OsmVehicle(VehicleType type);
48     OsmVehicle(const OsmVehicle& from) = default;
49     ~OsmVehicle() = default;
50
51 // OPERATORS
52 // OPERATIONS
53     /** Attempt to parse a string to a VehicleType
54      * @param   rTypeString     String which could contain a Vehicle type
55      * @return  A valid VehicleType
56      * @throw   OsmException if invalid string.
57      */
58     static VehicleType  parseString(const std::string& rTypeString);
59
60     /** Convert a Vehicle Type to a string representation.
61      * @param   vehicleType     The type to convert.
62      * @return  string representation of the type.
63      * @throw   OsmException if unknown vehicle type (out of bounds).
64      */
65     static std::string  toString(VehicleType vehicleType);
```

```
66
67      /** Convert this VehicleType to a string.
68       * @return  string representation of this VehicleType.
69       */
70      std::string         toString() const;
71
72  // ACCESS
73  // INQUIRY
74  protected:
75  private:
76      VehicleType             mType {VEHICLE};
77      static const std::string sTypeStrings[];
78  };
79
80  // INLINE METHODS
81  //
82
83  // EXTERNAL REFERENCES
84  //
85
86  #endif /* OSM_OSMVEHICLE_H_ */
```

D.12.14 OsmVehicle.cc

```
1   /*
2    * OsmVehicle.cc
3    *
4    * @author  Jonas Bergman
5    */
6
7   #include "OsmVehicle.h"  // class implemented
8
9   ////////////////////////////// PUBLIC //////////////////////////////////////
10
11  //============================= LIFECYCLE ====================================
12  OsmVehicle::OsmVehicle(OsmVehicle::VehicleType type)
13      : mType(type)
14  {}
15
16  //============================= OPERATORS ====================================
17
18  //static
19  OsmVehicle::VehicleType
20  OsmVehicle::parseString(const std::string& rTypeString)
21  {
22      for(size_t i = 0; i < NR_VEHICLE_TYPES; ++i)
23      {
24          if(rTypeString == OsmVehicle::sTypeStrings[i])
25          {
26              return static_cast<VehicleType>(i);
27          }
28      }
29      throw OsmException("OsmVehicle:parseString: Unknown Vehicle Type string.");
30  }
31
32  //static
33  std::string
```

```
34 OsmVehicle::toString(OsmVehicle::VehicleType vehicleType)
35 {
36     if(vehicleType >= NR_VEHICLE_TYPES)
37     {
38         throw OsmException("OsmVehicle:toString: Unknown Vehicle Type");
39     }
40     return OsmVehicle::sTypeStrings[vehicleType];
41 }
42
43 std::string
44 OsmVehicle::toString() const
45 {
46     return sTypeStrings[this->mType];
47 }
48
49 //============================== OPERATIONS ====================================
50 //============================== ACESS      ====================================
51 //============================== INQUIRY    ====================================
52 ////////////////////////////// PROTECTED  ////////////////////////////////////
53
54 ////////////////////////////// PRIVATE    ////////////////////////////////////
55 const std::string OsmVehicle::sTypeStrings[] =
56 {
57     "motorcar",
58     "goods",
59     "hgv",
60     "psv",
61     "lhv",
62     "motor_vehicle",
63     "vehicle"
64 };
```

D.12.15 OsmAccess_test.cc

```
1  /*
2   * OsmAccess_test.cc
3   *
4   * @author  Jonas Bergman
5   */
6
7  #include "../OsmAccess.h"
8  #include "../../catchtest/catch.hpp"
9
10 SCENARIO ("OsmAccess functionality testing", "[osm][access]")
11 {
12     try
13     {
14         // ---------------------------------------------------------------------
15         GIVEN ("a valid string of an access type")
16         {
17             std::string type_string("designated");
18
19             //.....................................................................
20             WHEN ("parsing string to an AccessType")
21             {
22                 OsmAccess::AccessType type =
23                     OsmAccess::parseString(type_string);
```

```
24                  THEN ("we should get the corresponding type")
25                  {
26                      REQUIRE (type == OsmAccess::AccessType::DESIGNATED);
27                  }
28              }
29          }
30
31          // ------------------------------------------------------------------
32          GIVEN ("an invalid string of an access type")
33          {
34              std::string type_string("foo");
35
36              //..........................................................
37              WHEN ("parsing string to a AccessType")
38              {
39                  THEN ("we should get an OsmException")
40                  {
41                      REQUIRE_THROWS_AS (OsmAccess::parseString(type_string),
42                          OsmException&);
43                  }
44              }
45          }
46
47          // ------------------------------------------------------------------
48          GIVEN ("an access type")
49          {
50              OsmAccess type(OsmAccess::DELIVERY);
51
52              //..........................................................
53              WHEN ("converting type to a string")
54              {
55                  THEN ("we should the corresponding string")
56                  {
57                      REQUIRE (type.toString() == "delivery");
58                  }
59              }
60          }
61
62          // ------------------------------------------------------------------
63          GIVEN ("an access rule")
64          {
65              OsmAccess::AccessRule rule({OsmAccess::YES, OsmAccess::PERMISSIVE});
66
67              //..........................................................
68              WHEN ("checking for access for type not in rule")
69              {
70                  OsmAccess type(OsmAccess::DELIVERY);
71
72                  THEN ("we should not be allowed access")
73                  {
74                      REQUIRE_FALSE (type.allowsAccess(rule));
75                  }
76              }
77          }
78      }
79      catch (OsmException& oe)
80      {
```

```
81          INFO(oe.what());
82          REQUIRE (false);    // force output of error and failure
83      }
84      catch (const std::exception& e)
85      {
86          INFO(e.what());
87          REQUIRE (false);    // force output of error and failure
88
89      }
90  }
```

D.12.16 OsmBarrier_test.cc

```
1   /*
2    * OsmBarrier_test.cc
3    *
4    * @author  Jonas Bergman
5    */
6
7   #include "../OsmBarrier.h"
8   #include "../../catchtest/catch.hpp"
9
10  SCENARIO ("OsmBarrier functionality testing", "[osm][barrier]")
11  {
12      try
13      {
14          // ------------------------------------------------------------------
15          GIVEN ("a valid string of an access type")
16          {
17              std::string type_string("swing_gate");
18
19              //..........................................................
20              WHEN ("parsing string to a BarrierType")
21              {
22                  OsmBarrier::BarrierType type =
23                      OsmBarrier::parseString(type_string);
24                  THEN ("we should get the corresponding type")
25                  {
26                      REQUIRE (type == OsmBarrier::BarrierType::SWING_GATE);
27                  }
28              }
29          }
30
31          // ------------------------------------------------------------------
32          GIVEN ("an invalid string of a barrier type")
33          {
34              std::string type_string("foo");
35
36              //..........................................................
37              WHEN ("parsing string to a BarrierType")
38              {
39                  THEN ("we should get an OsmException")
40                  {
41                      REQUIRE_THROWS_AS (OsmBarrier::parseString(type_string),
42                          OsmException&);
43                  }
44              }
```

```
45          }
46
47          // ------------------------------------------------------------------
48          GIVEN ("a barrier type")
49          {
50              OsmBarrier type(OsmBarrier::CATTLE_GRID);
51
52              //.....................................................................
53              WHEN ("converting type to a string")
54              {
55                  THEN ("we should the corresponding string")
56                  {
57                      REQUIRE (type.toString() == "cattle_grid");
58                  }
59              }
60          }
61
62          // ------------------------------------------------------------------
63          GIVEN ("a restriction rule")
64          {
65              OsmBarrier::RestrictionsRule rule({OsmBarrier::YES, OsmBarrier::BOLLARD});
66
67              //.....................................................................
68              WHEN ("checking if access is restricted for type not in rule")
69              {
70                  OsmBarrier type(OsmBarrier::SPIKES);
71
72                  THEN ("we should be told there is no restriction on access")
73                  {
74                      REQUIRE_FALSE (type.restrictsAccess(rule));
75                  }
76              }
77              //.....................................................................
78              WHEN ("checking if access is restricted for type in rule")
79              {
80                  OsmBarrier type(OsmBarrier::BOLLARD);
81
82                  THEN ("we should be told there restriction on access")
83                  {
84                      REQUIRE (type.restrictsAccess(rule));
85                  }
86              }
87          }
88
89          // ------------------------------------------------------------------
90          GIVEN ("a cost rule")
91          {
92  //            OsmBarrier::CostsRule rule({OsmBarrier::GATE, OsmBarrier::DEBRIS});
93              OsmBarrier::CostsRule rule;
94              rule.addCost(OsmBarrier::GATE, 10);
95              rule.addCost(OsmBarrier::DEBRIS, 10);
96
97              //.....................................................................
98              WHEN ("checking if access costs for type not in rule")
99              {
100                 OsmBarrier type(OsmBarrier::YES);
101
```

```
102                 THEN ("we should be told there is no cost on access")
103                 {
104                     REQUIRE_FALSE (type.costsToPass(rule));
105                 }
106             }
107             //.....................................................................
108             WHEN ("checking if access costs for type in rule")
109             {
110                 OsmBarrier type(OsmBarrier::DEBRIS);
111 
112                 THEN ("we should be told there is cost on access")
113                 {
114                     REQUIRE (type.costsToPass(rule));
115                 }
116             }
117         }
118     }
119     catch (OsmException& oe)
120     {
121         INFO(oe.what());
122         REQUIRE (false);    // force output of error and failure
123     }
124     catch (const std::exception& e)
125     {
126         INFO(e.what());
127         REQUIRE (false);    // force output of error and failure
128 
129     }
130 }
```

D.12.17 OsmHighway_test.cc

```
1  /*
2   * OsmHighway_test.cc
3   *
4   * @author  Jonas Bergman
5   */
6 
7  #include "../OsmHighway.h"
8  #include "../../catchtest/catch.hpp"
9 
10 SCENARIO ("OsmHighway functionality testing", "[osm][highway]")
11 {
12     try
13     {
14         // ---------------------------------------------------------------------
15         GIVEN ("a valid string of a highway type")
16         {
17             std::string type_string("primary");
18 
19             //.....................................................................
20             WHEN ("parsing string to a HighwayType")
21             {
22                 OsmHighway::HighwayType type =
23                     OsmHighway::parseString(type_string);
24                 THEN ("we should get the corresponding type")
25                 {
```

```
26                      REQUIRE (type == OsmHighway::HighwayType::PRIMARY);
27                  }
28              }
29          }
30
31          // ------------------------------------------------------------------
32          GIVEN ("an invalid string of a highway type")
33          {
34              std::string type_string("foo");
35
36              //.....................................................................
37              WHEN ("parsing string to a HighwayType")
38              {
39                  THEN ("we should get an OsmException")
40                  {
41                      REQUIRE_THROWS_AS (OsmHighway::parseString(type_string),
42                      OsmException&);
43                  }
44              }
45          }
46
47          // ------------------------------------------------------------------
48          GIVEN ("a highway type")
49          {
50              OsmHighway type(OsmHighway::PRIMARY);
51
52              //.....................................................................
53              WHEN ("converting type to a string")
54              {
55                  THEN ("we should the corresponding strng")
56                  {
57                      REQUIRE (type.toString() == "primary");
58                  }
59              }
60          }
61      }
62      catch (OsmException& oe)
63      {
64          INFO(oe.what());
65          REQUIRE (false);    // force output of error and failure
66      }
67      catch (const std::exception& e)
68      {
69          INFO(e.what());
70          REQUIRE (false);    // force output of error and failure
71
72      }
73  }
```

D.12.18 OsmTurningRestriction_test.cc

D.12.19 OsmVehicle_test.cc

```
1   /*
2    * OsmVehicle_test.cc
3    *
4    * @author  Jonas Bergman
```

```
 */

#include "../OsmVehicle.h"
#include "../../catchtest/catch.hpp"

SCENARIO ("OsmVehicle functionality testing", "[osm][vehicle]")
{
    try
    {
        // ---------------------------------------------------------------------
        GIVEN ("a valid string of a vehicle type")
        {
            std::string type_string("motorcar");

            //.....................................................................
            WHEN ("parsing string to a VehicleType")
            {
                OsmVehicle::VehicleType type =
                    OsmVehicle::parseString(type_string);
                THEN ("we should get the corresponding type")
                {
                    REQUIRE (type == OsmVehicle::VehicleType::MOTORCAR);
                }
            }
        }

        // ---------------------------------------------------------------------
        GIVEN ("an invalid string of a vehicle type")
        {
            std::string type_string("foo");

            //.....................................................................
            WHEN ("parsing string to a VehicleType")
            {
                THEN ("we should get an OsmException")
                {
                    REQUIRE_THROWS_AS (OsmVehicle::parseString(type_string),
                    OsmException&);
                }
            }
        }

        // ---------------------------------------------------------------------
        GIVEN ("a Vehicle type")
        {
            OsmVehicle type(OsmVehicle::PSV);

            //.....................................................................
            WHEN ("converting type to a string")
            {
                THEN ("we should the corresponding string")
                {
                    REQUIRE (type.toString() == "psv");
                }
            }
        }
    }
```

```
62      catch (OsmException& oe)
63      {
64          INFO(oe.what());
65          REQUIRE (false);    // force output of error and failure
66      }
67      catch (const std::exception& e)
68      {
69          INFO(e.what());
70          REQUIRE (false);    // force output of error and failure
71
72      }
73  }
```

D.13 preparation

D.13.1 README.md

```
Preparing database
==================

The preparation differs depending on how you import the map data and build the
 ↪  topology. So far in this project import has been done with `osm2pgsql` and
 ↪  topology built with `postgis_topology`.


Preparation for `osm2pgsql` and `postgis_topology`
--------------------------------------------------

To prepare the database for this software module when using `osm2pgsql` as
 ↪  importer of OpenStreetMap data into the database, we need to install
 ↪  extensions for:

- `postgis`
- `postgis_topology`
- `hstore`

and a couple of custom functions for finding turning restrictions:

- function `find_topo_edges_at_turning_restriction()`
- function `find_osm_turning_restrictions()`

The steps to follow are (assuming `mikhailovsk.osm` as source for OpenStreetMap
 ↪  data, and `tester` as a user with administrative rights in database, and
 ↪  `mikh_0530` as name of database):

### 1. Create database

    $ createdb mikh_0530 -U tester

### 2. Install extensions and functions

    $ psql -U tester -d mikh_0530 -f init_osm2pgsql_postgis_topology.sql

### 3. Import OSM data

    $ osm2pgsql -U tester -d mikh_0530 -s -k -S LGU.style mikhailovsk.osm
```

```
This uses the tool `osm2pgsql` to parse the osm-file into database tables.
The flags are

- `-s` slim, keeping extra tables.
- `-k` keeping tags in `hstore` if not in their own column.
- `-S` Style file, configuring which tags to have columns or not.

### 4. Building topology

This step is optional. It should be efficient and safe to build the topology once
 ↪  and for all after importing as differing conditions and temporary closures
 ↪  could be specified with costs and restrictions instead of via topology. But
 ↪  one can also configure the tool to build topology on each call, see the
 ↪  `configuration` package.

    $ psql -U tester -d mikh_0530 -f build_postgis_topology.sql

This step creates a table `public.highways_lgu` and adds a new schema called
 ↪  `topo_lgu` which contains tables for the topology.


Test databases
--------------
During testing different databases has been tested, they were created so:

#### `mikh_style`

    $ createdb mikh_style -U jonas
    $ psql -U jonas -d mikh_style -c "CREATE extension postgis;"
    $ psql -U jonas -d mikh_style -c "CREATE extension postgis_topology;"
    $ psql -U jonas -d mikh_style -c "CREATE extension hstore;"
    $ psql -U jonas -d mikh_style -c "SET search_path=topology,public
    $ osm2pgsql -U jonas -d mikh_style -s -k -S new.style mikhailovsk.osm
    $ psql -U jonas -d mikh_style -c "CREATE TABLE highways_test AS SELECT * FROM
     ↪  planet_osm_line WHERE highway IS NOT NULL;"
    $ psql -U jonas -d mikh_style -c "SELECT topology.CreateTopology('topo_test',
     ↪  900913);"
    $ psql -U jonas -d mikh_style -c "SELECT
     ↪  topology.AddTopoGeometryColumn('topo_test', 'public', 'highways_test',
     ↪  'topo_geom', 'LINESTRING');"
    $ psql -U jonas -d mikh_style -c "UPDATE highways_test SET topo_geom =
     ↪  topology.toTopoGeom(way, 'topo_test', 1, 1.0);"

#### `mikh_0522`

    $ createdb mikh_0522 -U jonas
    $ psql -U jonas -d mikh_0522 -c "CREATE EXTENSION postgis;"
    $ psql -U jonas -d mikh_0522 -c "CREATE EXTENSION postgis_topology;"
    $ psql -U jonas -d mikh_0522 -c "CREATE EXTENSION hstore;"
    $ osm2pgsql -U jonas -d mikh_0522 -s -k -S LGU.style mikhailovsk.osm
    $ psql -U jonas -d mikh_0522 -c "CREATE TABLE highways_test AS SELECT * FROM
     ↪  planet_osm_line WHERE highway IS NOT NULL;"
    $ psql -U jonas -d mikh_0522 -c "SELECT topology.CreateTopology('topo_test',
     ↪  900913);"
    $ psql -U jonas -d mikh_0522 -c "SELECT
     ↪  topology.AddTopoGeometryColumn('topo_test', 'public', 'highways_test',
     ↪  'topo_geom', 'LINESTRING');"
```

```
    $ psql -U jonas -d mikh_0522 -c "UPDATE highways_test SET topo_geom =
    ↪  topology.toTopoGeom(way, 'topo_test', 1, 1.0);"

#### `mikh_0530`
Described above.

#### `mikh_restr_0602`
As `mikh_0530` but using the modified osm-file `mikhailovsk-turnrestriction.osm`
↪  instead. That file has been modified with a turn restriction for testing
↪  purposes.

#### `mikh_restr_0617`
As `mikh_0602` but extra columns in `planet_osm_point` to get point restrictions.
```

D.13.2 build_postgis_topology.sql

```
CREATE TABLE highways_lgu
AS SELECT *
FROM planet_osm_line
WHERE highway IS NOT NULL;

SELECT topology.CreateTopology('topo_lgu', 900913);

SELECT topology.AddTopoGeometryColumn('topo_lgu',
                                      'public',
                                      'highways_lgu',
                                      'topo_geom',
                                      'LINESTRING');

UPDATE highways_lgu SET topo_geom = topology.toTopoGeom(way, 'topo_lgu', 1, 1.0);
```

D.13.3 init_osm2pgsql_postigs_topology.sql

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
CREATE EXTENSION hstore;

DROP TABLE IF EXISTS turning_restrictions;
CREATE TABLE turning_restrictions(
   from_osm_id       bigint,
   to_osm_id         bigint,
   via_osm           varchar,
   edge_ids          integer[],
   restriction_type  varchar);

--
-- Find the topology edge ids affected by osm turn restrictions.
--
CREATE OR REPLACE FUNCTION
   find_topo_edges_at_turning_restriction(
      osm_edges_table text,
      from_osm_id bigint,
      to_osm_id bigint,
      topo_edges_table text)
RETURNS setof RECORD
AS $$
```

```
24 BEGIN
25    RETURN QUERY EXECUTE format('
26       SELECT edge_id
27       FROM %4$s
28       WHERE ST_DWithin (
29          geom,
30          (   SELECT ST_Intersection(a.way, b.way)
31              FROM %1$I a, %1$I b
32              WHERE a.osm_id = %2$s AND b.osm_id = %3$s
33          ),
34          1.0
35       );'
36       , osm_edges_table, from_osm_id, to_osm_id, topo_edges_table);
37 END;
38 $$ LANGUAGE 'plpgsql';
39
40
41 --
42 -- Find all the restrictions and put them in table 'turning_restrictions'
43 --
44 CREATE OR REPLACE FUNCTION
45     find_osm_turning_restrictions(osm_edges_table text, topo_edges_table text)
46 RETURNS integer
47 AS $$
48 DECLARE
49    nrFindings integer := 0;
50    nrFrom integer := 0;
51    nrTo integer := 0;
52    restrictionRecord  record;
53    ix integer;
54    id bigint;
55    fromOsmId bigint;
56    toOsmId bigint;
57    viaText text := '';
58    restrictions text[] := '{
59             "no_right_turn",
60             "no_left_turn",
61             "no_u_turn",
62             "no_straight_on",
63             "only_right_turn",
64             "only_left_turn",
65             "only_straight_on",
66             "no_entry",
67             "no_exit"
68          }';
69    restrType text;
70    edgeId integer;
71    edges integer[];
72
73 BEGIN
74    FOR restrictionRecord IN
75       SELECT *
76       FROM planet_osm_rels
77       WHERE (
78          SELECT 'restriction' = ANY(tags)
79       )
80       AND ( -- check that the restriction type is given in tags
```

```
81          SELECT restrictions && tags
82       )
83    LOOP
84       -- look through 'members' in all restrictions, must have at least 6 elements
85       -- {from_id, from, via_id, via, to_id, to}
86       IF (array_upper(restrictionRecord.members, 1) >= 6) THEN
87
88          nrFrom := 0;
89          nrTo := 0;
90
91          -- look for type: from, via, to
92          FOR ix IN 1..(array_length(restrictionRecord.members, 1)-1)
93          LOOP
94             IF restrictionRecord.members[ix+1] LIKE 'from' THEN
95                fromOsmId :=
96                   trim(leading 'wn' from restrictionRecord.members[ix])::bigint;
97                nrFrom := nrFrom + 1;
98             ELSIF restrictionRecord.members[ix+1] LIKE 'to' THEN
99                toOsmId :=
100                  trim(leading 'wn' from restrictionRecord.members[ix])::bigint;
101               nrTo := nrTo + 1;
102            ELSIF restrictionRecord.members[ix+1] LIKE 'via' THEN
103               viaText := viaText || restrictionRecord.members[ix] || ',';
104            END IF;
105         END LOOP;
106
107         IF (nrFrom != 1 OR nrTo != 1) THEN
108            CONTINUE;
109         END IF;
110
111         -- look for restriction type
112         FOR ix IN 1..array_upper(restrictions, 1)
113         LOOP
114            IF (SELECT restrictions[ix] = ANY(restrictionRecord.tags)) THEN
115               restrType := restrictions[ix];
116               EXIT;
117            END IF;
118         END LOOP;
119
120        -- find topology edge ids that might be affected
121        -- each osm edge could have two topology edges (in and out at vertex)
122        -- and there is no really easy way of finding who is who?
123        FOR edgeId IN
124           SELECT *
125           FROM find_topo_edges_at_turning_restriction(
126              osm_edges_table,
127              fromOsmId,
128              toOsmId,
129              topo_edges_table)
130           AS f(id integer)
131        LOOP
132           edges := array_append(edges, edgeId);
133        END LOOP;
134
135         -- store findings
136         INSERT INTO turning_restrictions
137            VALUES (fromOsmId, toOsmId, viaText, edges, restrType);
```

```
138             nrFindings :=  nrFindings + 1;
139           END IF;
140       END LOOP;
141       RETURN nrFindings;
142   END;
143   $$ LANGUAGE 'plpgsql';
```

D.13.4 LGU.style

```
1   # This is the default osm2pgsql .style file that comes with osm2pgsql.
2   #
3   # A .style file has 4 columns that define how OSM objects end up in tables in
4   # the database and what columns are created. It interacts with the command-line
5   # hstore options.
6   #
7   # Columns
8   # =======
9   #
10  # OsmType: This is either "node", "way" or "node,way" and indicates if this tag
11  # applies to nodes, ways, or both.
12  #
13  # Tag: The tag
14  #
15  # DataType: The type of the column to be created. Normally "text"
16  #
17  # Flags: Flags that indicate what table the OSM object is moved into.
18  #
19  # There are 5 possible flags. These flags are used both to indicate if a column
20  # should be created, and if ways with the tag are assumed to be areas. The area
21  # assumptions can be overridden with an area=yes/no tag
22  #
23  # polygon - Create a column for this tag, and objects the tag with are areas
24  #
25  # linear - Create a column for this tag
26  #
27  # phstore - Don't create a column for this tag, but objects with the tag are areas
28  #
29  # delete - Drop this tag completely and don't create a column for it. This also
30  # prevents the tag from being added to hstore columns
31  #
32  # nocache - Deprecated and does nothing
33  #
34  # If an object has a tag that indicates it is an area or has area=yes/1,
35  # osm2pgsql will try to turn it into an area. If it succeeds, it places it in
36  # the polygon table. If it fails (e.g. not a closed way) it places it in the
37  # line table.
38  #
39  # Nodes are never placed into the polygon or line table and are always placed in
40  # the point table.
41  #
42  # Hstore
43  # ======
44  #
45  # The options --hstore, --hstore-match-only, and --hstore-all interact with
46  # the .style file.
47  #
48  # With --hstore any tags without a column will be added to the hstore column.
```

```
# This will also cause all objects to be kept.
#
# With --hstore-match-only the behavior for tags is the same, but objects are
# only kept if they have a non-NULL value in one of the columns.
#
# With --hstore-all all tags are added to the hstore column unless they appear
# in the style file with a delete flag, causing duplication between the normal
# columns and the hstore column.
#
# Special database columns
# ========================
#
# There are some special database columns that if present in the .style file
# will be populated by osm2pgsql.
#
# These are
#
# z_order - datatype int4
#
# way_area - datatype real. The area of the way, in the units of the projection
# (e.g. square mercator meters). Only applies to areas
#
# osm_user - datatype text
# osm_uid - datatype integer
# osm_version - datatype integer
# osm_changeset - datatype integer
# osm_timestamp - datatype timestamptz(0).
# Used with the --extra-attributes option to include metadata in the database.
# If importing with both --hstore and --extra-attributes the meta-data will
# end up in the tags hstore column regardless of the style file.

# OsmType   Tag                 DataType     Flags
##################################################
node,way    access              text         linear
node,way    barrier             text         linear
node        crossing            text         linear
node,way    disused             text         linear
node,way    emergency           text         linear
node,way    highway             text         linear
node,way    incline             text         linear
way         junction            text         linear
way         lanes               text         linear
way         maxheight           text         linear
way         maxlength           text         linear
way         maxspeed            text         linear
way         minspeed            text         linear
way         maxweight           text         linear
way         maxwidth            text         linear
node,way    noexit              text         linear
way         oneway              text         linear
node,way    public_transport    text         linear
node,way    restriction         text         linear
node,way    railway             text         linear # :level_crossing, tram, tram_stop
way         surface             text         linear
node        toll                text         linear
way         tracktype           text         linear
node,way    traffic_calming     text         linear
```

```
node,way    traffic_sign        text         linear

# Access restrictions for vehicle types
node,way    goods               text         linear
node,way    hgv                 text         linear
node,way    lhv                 text         linear
node,way    motorcar            text         linear
node,way    motor_vehicle       text         linear
node,way    psv                 text         linear
node,way    vehicle             text         linear



# Deleted tags
# These are tags that are generally regarded as useless for most rendering.
# Most of them are from imports or intended as internal information for mappers
# Some of them are automatically deleted by editors.
# If you want some of them, perhaps for a debugging layer, just delete the lines.

# These tags are used by mappers to keep track of data.
# They aren't very useful for rendering.
node,way    note                text         delete
node,way    note:*              text         delete
node,way    source              text         delete
node,way    source_ref          text         delete
node,way    source:*            text         delete
node,way    attribution         text         delete
node,way    comment             text         delete
node,way    fixme               text         delete

# Tags generally dropped by editors, not otherwise covered
node,way    created_by          text         delete
node,way    odbl                text         delete
node,way    odbl:note           text         delete
node,way    SK53_bulk:load      text         delete

# Lots of import tags
# TIGER (US)
node,way    tiger:*             text         delete

# NHD (US)
# NHD has been converted every way imaginable
node,way    NHD:*               text         delete
node,way    nhd:*               text         delete

# GNIS (US)
node,way    gnis:*              text         delete

# Geobase (CA)
node,way    geobase:*           text         delete
# NHN (CA)
node,way    accuracy:meters     text         delete
node,way    sub_sea:type        text         delete
node,way    waterway:type       text         delete

# KSJ2 (JA)
# See also note:ja and source_ref above
```

```
node,way    KSJ2:*              text        delete
# Yahoo/ALPS (JA)
node,way    yh:*                text        delete

# osak (DK)
node,way    osak:*              text        delete

# kms (DK)
node,way    kms:*               text        delete

# ngbe (ES)
# See also note:es and source:file above
node,way    ngbe:*              text        delete

# naptan (UK)
node,way    naptan:*            text        delete

# Corine (CLC) (Europe)
node,way    CLC:*               text        delete

# misc
node,way    3dshapes:ggmodelk   text        delete
node,way    AND_nosr_r          text        delete
node,way    import              text        delete
node,way    it:fvg:*            text        delete
```

D.13.5 mikhailovsk.osm

The file used during testing was supplied by the company, and is an edited version. But it is not necessary to put 2.1 MiB of *xml* data here. A new file, containing more information, can be downloaded, see listing D.1:

```
$ wget -O mikhailovsk.osm "http://overpass-api.de/api/map?bbox=41.9491,45.0918,42.1151,45.173"
```

Listing D.1: Download Mikhailovsk map data.

D.13.6 partille.osm

The file used during testing was supplied by the company, and is an edited version. But it is not necessary to put 4.4 MiB of *xml* data here. A new file, containing more information, can be downloaded, see listing D.2:

```
$ wget -O partille.osm "http://overpass-api.de/api/map?bbox=12.0873,57.7168,12.1703,57.7475"
```

Listing D.2: Download Partille map data.

D.14 util

D.14.1 Logging.h

```
/* Use Boost loggging, and handle setup in this file.
 *
 * #include "Logging.h"
 *
 * Needs a lot of linking to work:
 * -lboost_log -lboost_log_setup -lboost_thread -lboost_system -lpthread
 *
```

```
 * @author  Jonas Bergman
 *
 */

#ifndef LGU_LOGGING_H_
#define LGU_LOGGING_H_

#include <boost/log/common.hpp>
#include <boost/log/core.hpp>
#include <boost/log/expressions.hpp>
#include <boost/log/sinks/text_file_backend.hpp>
#include <boost/log/sources/severity_logger.hpp>
#include <boost/log/support/date_time.hpp>
#include <boost/log/utility/setup/file.hpp>
#include <boost/log/utility/setup/common_attributes.hpp>
#include <boost/log/trivial.hpp>

/** To simplify the set up of logging in the application: include this file
 * and call the 'initLogging()' function.
 */
struct Logging
{
    static void initLogging()
    {
        if(isInited)
        {
            return;
        }

        boost::log::add_file_log(
            boost::log::keywords::file_name = "lgu.log",
            boost::log::keywords::format = "[%TimeStamp%]: %Message%"
            );
        boost::log::core::get()->set_filter(
            boost::log::trivial::severity >= boost::log::trivial::info
        );

        isInited = true;
    }

    Logging() = delete;
    Logging(const Logging& from) = delete;

private:
    static bool isInited;
};

#endif /* LGU_LOGGING_H_ */
```

D.14.2 Logging.cc

```
/*
 * Logging.cc
 */

#include "Logging.h"

```

```
7  //static
8  bool Logging::isInited {false};
```

D.14.3 Point.h

```
1  /**  Data structure for Point.
2   *
3   * #include "Point.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef UTIL_POINT_H_
9  #define UTIL_POINT_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <ostream>
14
15 // PROJECT INCLUDES
16 //
17
18 // LOCAL INCLUDES
19 //
20
21 // FORWARD REFERENCES
22 //
23
24 struct Point
25 {
26 // ATTRIBUTES
27     double    x {0.0};
28     double    y {0.0};
29
30     Point(double x, double y) : x(x), y(y) {}
31     Point() = default;
32     Point(const Point&) = default;
33
34 // OPERATORS
35     friend std::ostream& operator<<(std::ostream& os, const Point& rPoint)
36     {
37         os << std::fixed << "Point [x: " << rPoint.x << ", y: " << rPoint.y << "]";
38         return os;
39     }
40
41     bool    operator==(const Point& rhs) const
42     {
43         return (rhs.x == x) && (rhs.y == y);
44     }
45 };
46
47 // INLINE METHODS
48 //
49
50 // EXTERNAL REFERENCES
51 //
52
```

```
53 #endif /* UTIL_POINT_H_ */
```

D.14.4 TimeToStringMaker.h

```
1  /**  Static class to provide strings based on time.
2   *
3   * #include "TimeToStringMaker.h"
4   *
5   * @author  Jonas Bergman
6   */
7
8  #ifndef UTIL_TIMETOSTRINGMAKER_H_
9  #define UTIL_TIMETOSTRINGMAKER_H_
10
11 // SYSTEM INCLUDES
12 //
13 #include <string>
14
15 // PROJECT INCLUDES
16 //
17
18 // LOCAL INCLUDES
19 //
20
21 // FORWARD REFERENCES
22 //
23
24 /**
25  * Class who provide strings from times.
26  */
27 class TimeToStringMaker
28 {
29 public:
30 // LIFECYCLE
31
32     /** Default constructor.
33      */
34     TimeToStringMaker() = delete;
35
36     /** Copy constructor */
37     TimeToStringMaker(const TimeToStringMaker& from) = delete;
38
39 // OPERATORS
40 // OPERATIONS
41
42     /** Get the current time as a string.
43      *
44      * @return    A string representation of the time.
45      */
46     static std::string        getEpochMsTimeString();
47
48 // ACCESS
49 // INQUIRY
50
51 protected:
52
53 private:
```

```
};

// INLINE METHODS
//

// EXTERNAL REFERENCES
//

#endif /* UTIL_TIMETOSTRINGMAKER_H_ */
```

D.14.5 TimeToStringMaker.cc

```
/*
 * TimeToStringMaker.cc
 *
 * @author  Jonas Bergman
 */


#include "TimeToStringMaker.h"  // class implemented

#include <chrono>

////////////////////////////// PUBLIC //////////////////////////////////////

//============================= LIFECYCLE ====================================

//============================= OPERATORS ====================================

//============================= OPERATIONS ===================================
//static
std::string
TimeToStringMaker::getEpochMsTimeString()
{
    using namespace std::chrono;
    milliseconds ms = duration_cast< milliseconds >(
        system_clock::now().time_since_epoch());
    return std::to_string(ms.count());
}

//============================= ACESS      ===================================
//============================= INQUIRY    ===================================
////////////////////////////// PROTECTED  //////////////////////////////////

////////////////////////////// PRIVATE    //////////////////////////////////
```