UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Careelika Liisi Kuik

Multimodal Route Planning Algorithm for Encouraging the Usage of Different Means of Public Transportation

Master's Thesis (30 ECTS)

Supervisor(s): Amnir Hadachi, PhD

Tartu 2019

Multimodal Route Planning Algorithm for Encouraging the Usage of Different Means of Public Transportation

Abstract:

The ongoing urbanization and the growth of the cities is leading to the increase of complexity of the route planning in urban areas. Often it is not possible or feasible to travel from one location to another using only one mode of transportation. Moreover, in case of specific preferences like taking a wheelchair, baby carriage or a bicycle in the mean of public transport, a specific type of mean of transport (e.g. wheelchair-accessible bus) is needed. However, the existing routing engines tend to heavily prefer the first public transport trip of any mean of public transport that meets the spatiotemporal conditions instead of sticking to user's selected modes.

The aim of this thesis is to propose an alternative method for multimodal route planning, using only the modes and means of public transport that are allowed by the user.

In this thesis work an alternative method for multimodal fastest pathfinding with use of public transportation is developed. It is able to propose competitive alternatives to the results of the existing routing engines at the same time using only the modes and means of public transport that are allowed by the user.

Keywords:

Multimodal routing, transportation systems, public transport

CERCS: P 170

Multimodaalne teekonnaplaneerimise algoritm erisuguste ühistranspordivahendite kasutamise soodustamiseks

Lühikokkuvõte:

Jätkuv linnastumine ja linnade kasv muudab ka linnasisese teekonna planeerimise aina keerulisemaks.. Tihti pole võimalik reisida ühest pubktist teise, kasutades ainult üht transpordiliiki. Veelgi enam, juhul, kui kasutajal on spetsiifilisi eelistusi, nagu soov võtta ühistransporti kaasa ratastool, lapsevanker või jalgratas, kindlat tüüpi ühistranspordivahendi kasutamine (näiteks ratastoolisõbralik buss) on tarvilik.

Sellest olenemata kalduvad olemasolevad teekonnaplaneerimise mootorid suurel määral eelistama esimest suvalist tüüpi ühistranspordi reisi, kui see vastab aegruumilistele nõudmistele, selle asemel, et kinni pidada kasutaja poolt valitud ühistranspordi liikidest.

Käesoleva lõputöö eesmärk on pakkuda välja alternatiivne meetod multimodaalseks teekonnaplaneerimiseks, mis kasutaks ainult neid ühistranspordiliike, mis on kasutaja poolt lubatud.

Selles lõputöösalternatiivne kiireima multimodaalse teekonna leidmise meetod, mis kasutab ühistransporti, on arendatud. See on võimelinepakkuma konkurentsivõimelisi alternative olemasolevate teekonnaleidmise otsingumootorite poolt pakutud lahendustele, samal ajal kasutades vaid neid ühistranspordiliike, mis on kasutaja poolt lubatud.

Võtmesõnad:

Multimodaalse teekonna leidmine, transpordisüsteemid, ühistransport

CERCS: P 170

Table of Contents

List of Abbreviations

ITS	Intelligent Transportation Systems
OSM	OpenStreetMap
SQL	Structured Query Language
XML	Extensible Markup Language
API	Application Programming Interface
URL	Uniform Resource Locator
CSV	Comma Separated Values
PL/SQL	Procedural Language/Structured Query Language
GTFS	General Transit Feed Specification

1. Introduction

1.1 General view

With the ever increasing urbanization and the growth of the size of the cities and their population, finding fastest and the most comfortable traveling trajectory from one location to another has become more important than ever before. The more complex is the infrastructure of the environment, the more choices for planning the trajectory there are. Often it is not possible or feasible to travel from one location to another using only one mode of transportation. Moreover, in case of specific preferences like taking a wheelchair, baby carriage or a bicycle in the mean of public transport, a specific type of mean of transport (e.g. wheelchair-accessible bus) is needed. However, routing engines tend to heavily prefer the first public transport trip of any mean of public transport that meets the spatiotemporal conditions instead of sticking to user's selected modes. For example, user can select train as preferred mean of transport while searching for a route using Google's routing, but if according to Google's routing's calculations a trip of some other mean of transport (e.g. a bus) seems to be faster, the faster trip will be displayed to the user. Having the fastest trip is the most important in most cases, however, in case people have a strong need to use a certain type of mean of transport (e.g. need for a wheelchair-accessible mean of transport), they are ready to wait for the first trip of the preferred public transport mean instead. For example, in the case of Estonia, all the trains used by Elron are wheelchair-accessible and also easily accessible for people with baby carriers and bicycles [1]. Thus trains are a secure choice for people with such needs whereas no bus, tram nor trolley line in Tallinn can guarantee the same. In this case the user who for some reason needed to travel in train, still needs to make alternative queries or plan the trip manually in order to achieve the desired result. For this reason, there is a need for a routing service, that instead of modifying user's input in the way suitable for the routing engine, would give the user the results for the exact query made. Furthermore, when using public transport, the best paths between the start location and end locations of the journey and the public transport stops also need to be found. Also here the modes preferred by the user need to be taken into account in order to provide the user with the most suitable solution for the current situation. This converts the problem into multimodal routing problem. The challenge of finding the fastest path with the closest public transport trip between the start and the end location at the same time using only the modes and means of public transport that allowed by the user, is the topic of this thesis.

1.2 Objectives

The main purpose of this thesis is to provide an alternative method for multimodal fastest pathfinding with use of public transportation.

To achieve the main objective of this thesis, there is a need to extract and process the source data for private and public transportation modes and create an algorithm that can compose multimodal trajectories. In order to do that, it uses several algorithms:

- An algorithm for pathfinding in private modes
- An algorithm for pathfinding in public modes
- An algorithm for combining the unimodal trajectories into multimodal trajectories

In the process of finding the best trajectory user preferences are taken into account as restrictions for the trajectory to be found.

1.3 Contributions

The contributions of this thesis can be resumed as follows:

- Providing a good practice for parsing, processing and inserting the road data from OSM files to the database
- Creation of a modified algorithm based on A Star for pathfinding in private modes
- Creation of an algorithm for finding the best public transport trip in the modes allowed by the user
- Creation of an algorithm for combining and merging the unimodal trajectories into multimodal trajectories

As a result, the method finds the best multimodal trajectory for the given source and end location, taking into account the data provided by the user.

1.4 Road Map

This paper begins with an introductory chapter 1. Firstly, its purpose is to introduce and give a general view of the topic. Secondly, the objectives and limitations of the current work are listed. Next, the contribution made by this thesis to the general topic is described. The chapter concludes with a road map listing all the parts of this paper.

The second chapter of this paper aims to describe the state-of-art of the current topic and the related works that have been performed until the current moment. Additionally, the list of the

definitions used in this paper is given. The sub-chapters of the second paragraph are introduction, key terms-definitions, related work and conclusion.

The third chapter consists of the description of system architecture and contribution. More precisely this chapter firstly states the particular problem covered in this paper, describes the system design and architecture and describes the methodology in details.

The fourth chapter presents the results of using the proposed methodology and provides an analyses to the results.

The final, fifth chapter makes a conclusion of the work and states the perspectives for possible future work.

Additionally, abstract, table of contents, acknowledgement, bibliography, appendix and licence are added to the paper.

2. Background and Related Work

2.1 Introduction

This chapter covers state of the art related to the thesis topic. Therefore, the chapter is divided into two sections. Firstly, the key terms and definitions used in the context of this work are provided. Secondly, the related work concerning multimodal pathfinding is described. As there are many more algorithms for pathfinding in general than described in this section, only the ones that have similar traits to the method proposed in this thesis, are introduced.

2.2 Key Terms – Definitions

Next, key terms and definitions are listed:

a) **Node**: is defined as a pair of x and y coordinates (longitude and latitude) on planar graph and is used to provide localization on the map.

b) **Way**: is defined as an edge that connects two nodes (source and target) on planar graph and is used to depict a road segment in form of polyline on the map.

c) **Trajectory** (also path): is defined as a collection of one or more ways in a sequential order. It starts with the source node and ends with the target node on planar graph. It is used to depict directed sequence of ways on a map.

d) **Multitrajectory**: is defined as a collection of one or more trajectories in a sequential order. It starts with the source node and ends with the target node on planar graph. It is used to depict sequence of trajectories on a map.

2.3 Related Work

2.3.1 Multimodal pathfinding

Finding shortest path in graph where each edge has got a distance and a cost is considered a NP-hard problem. [2] This also applies to finding the fastest path in graph. Tsolkas et al. [3] used short-time history and estimators in their multimodal dynamic routing algorithm. [4] and [3] use historical statistics data for estimating car mode travel time. However, this kind of information is not freely available and can result costly as it needs to be kept updated in order to stay timely. Besides, collecting and storing travelling history can be complicated for various reasons. For this reason, in our proposed method, no historical data is used. [3] uses Bayesian learning techniques to analyze the data from users. However, users have to fill in a profile before starting to use the service. Additionally, this approach only performs significantly better in cold start. [3] There are some approaches described in [4], [6] and [3] that require filling in

a profile before usage. [6] takes into account user preferences like the number of modal transfers, preferred modes, expected travel time, delays at mode and arc switching points, viability of the sequence of the used modes are taken into account in this multimodal shortest path algorithm. Using small amount of user preferences like preferred modes helps to decrease the overall computing time and simplifies finding the most suitable trajectories for the user. For this reason, user inserted allowed modes are used in our proposed method as well. In this aforementioned approach a network consisting of subnetworks in different modes, a multimodal graph, needs to be constructed before executing the algorithm. The approach of multimodal graph is also used in [7] and [8]. However, constructing a multimodal graph decreases the speed of algorithm and demands more storage space. Moreover, performing large-scale construction of multimodal graph justifies itself only if the effort of finding neighbouring nodes and ways and computing their costs only on demand is greater than pre-constructing all the graph. However, more often using user preferences and decreasing the bounds of the network to be processed instead of extracting the data for e.g. of all the city leads to decrease of the computing time. For these reason, combining different network layers into one multimodal network is not used in our proposed method. Secondly, in [6] no pre-processing for the data is done. This leads to a bottle neck in data reading and compiling that increases drastically with addition of each additional mode. To avoid this downside, pre/processing of the data is done in our proposed method. There are many approaches like [7], [8], [9], [10], [11] and [12], where Dijkstra's algorithm or a modified version of it is used. However, while ensuring the answer with the best cost, using Dijkstra's algorithm can lead to unreasonably high computing time due to the number of processed nodes. To avoid this downside, Dijkstra's algorithm is not used in our proposed method. There are also some approaches like [13] that use A Star algorithm or its modification. A Star takes into account not only the cost of the path from source node to the current node, but also the remaining estimated cost from the current node to the target node. By doing so, using A Star with competent heuristics leads to decreasing the amount of nodes to be processed while still finding the answer that has the smallest cost. For this reason, a modification of A Star algorithm is used in our proposed method. [14] uses dynamic programming-based algorithm with nested itinerary planning. Nested public transport trips are also used by [12]. While this kind of approach is rather fast and compact for finding intermediate trips, this approach depends greatly on the timeliness of the public transport. For e.g. if bus on one trip arrives 5 minutes late, it is probable that the user misses the bus of the subsequent trip, etc. For example, in Tallinn the departure times in the reality are officially allowed to differ up to 5 minutes from the scheduled departure times. As a consequence,

ensuring the necessary buffer between the subsequent trips immensely decreases the accuracy of the estimated travel time making it pointless providing the expected travel time in case of shorter distances. For this reason, in our proposed method, instead of using the approach of nested subsequent public transport trips, only one public transport trip is allowed per trajectory. There are also approaches that take into account vehicles' driving events, train models with the gathered data and later use the modes to make estimations more precise. This kind of approach is used by [15]. However, such models cover only the most frequent cases and the results for corner cases tend to be inaccurate. Furthermore, as discussed earlier, the problem of the availability of the data arises. For these reasons, using trainable models are not used in our proposed method.

2.3.2 Conclusion

There is a number of works related to the current topic, but all of them have downsides that can be surmounted. The method described in the current paper aims to propose a solution that uses both public network data from peatus.ee and private network data from OSM to find the fastest feasible multimodal path between source and target locations. Furthermore, for this method an alternative algorithm for finding the closest public transport trip is developed. In addition, multimodal combinations that are not offered in the related works, like using precisely only the means of public transport that are allowed by the user, without using any hierarchy, replacement or nesting of subsequent trips, are provided.

3. System Architecture

3.1 Introduction

In the following chapter the system design and architecture of our proposed method is described. The chapter is divided into subchapters, so that each logical part is described in a separate subsection. In each subchapter the purpose and structure of the logical part is described. Furthermore, the architectural choices, challenges and constraints are described.

3.2 The General Overview of the Architecture

To begin with, the general schema of the system architecture is shown on figure 1.

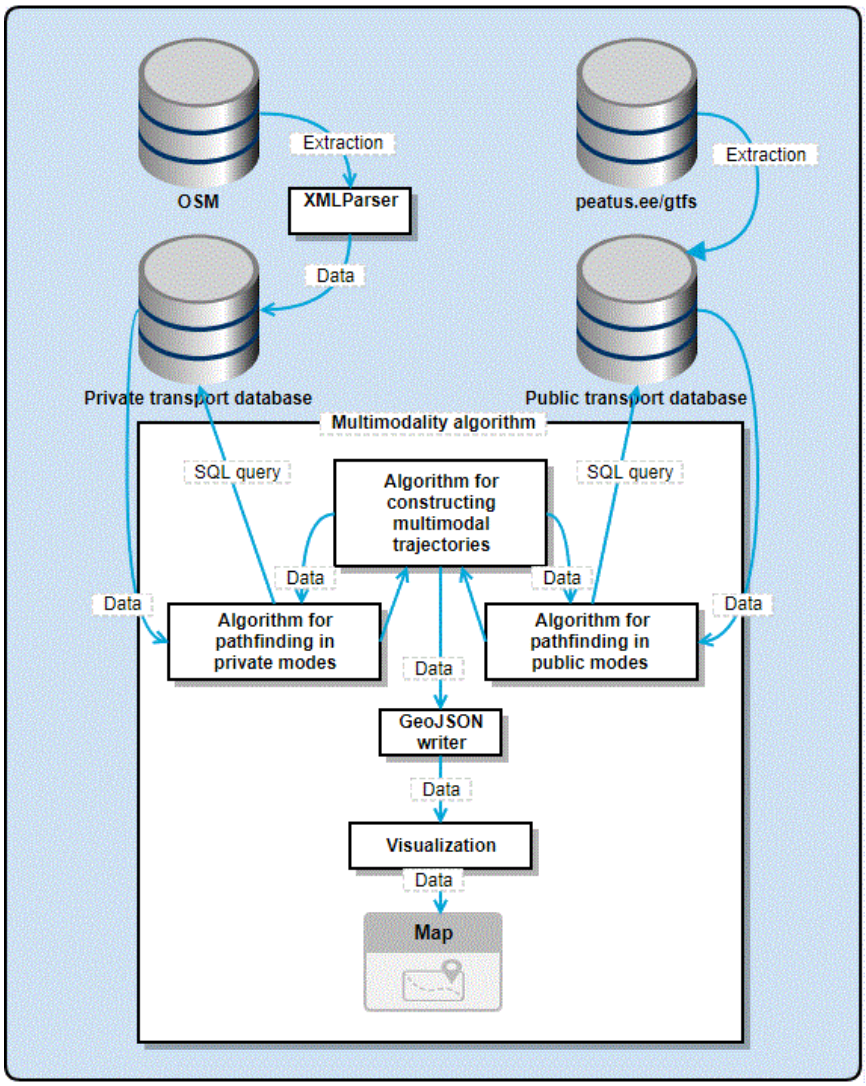

Figure 1. The system architecture.

The architecture of the system consists of data sources, an algorithm for extracting and converting the necessary data from OSM, Additionally, an algorithm for pathfinding in

private mode and an algorithm for pathfinding in public mode are included. Multimodal trajectories are combined and constructed using a separate algorithm. Finally, there is an algorithm for writing results to a GeoJSON file and an algorithm for visualization on a web map.

3.3 Pathfinding in Private Mode

3.3.1 Pre-Processing OSM Data

For pathfinding in private mode, data from OSM is used. OSM provides free and open data that is freely accessible for anybody for downloading, modification and use. [16] The data can be downloaded from OSM database in OSM XML format. OSM XML format is a data format derived from XML that has objects like nodes, ways and relations to effectively store the road data. [17] The structure of a sample OSM file can be seen on figure 2.

There are several tools for extracting and pre-processing OSM files freely available. JOSM is an open source extensible editor for OSM files that supports loading and editing OSM data. [18] However, JOSM is too slow and heavy weight if only the extraction of data is needed. Osmosis is a command line Java application that processes OSM data. However, using Osmosis is uncomfortable as for installing Osmosis in Windows operating system a manual fix needs to be added. [19] Additionally, for processing the data the configuration should be modified in order to avoid having unnecessary data included or excluded during the process. When using PostgreSQL database with PostGIS and pgrouting a few more options are available. Osm2postgresql, Osm2pgrouting and Osm2pgsql all enable converting OSM data into a database that is compatible with PostgreSQL. [20][21][22] However, all of them are open source projects that lack necessary updated documentation for configuring the configuration

file and contain unfixed bugs that result in inaccurate data, data loss and mistakes.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.6.1 (13750 thorn-03.openstreetmap.org)" copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyright"
license="http://opendatacommons.org/licenses/odbl/1-0/">
 <bounds minlat="59.4334000" minlon="24.7589800" maxlat="59.4334400" maxlon="24.7590700"/>
 <node id="401316643" visible="true" version="2" changeset="3663582" timestamp="2010-01-19T23:44:26Z" user="k__" uid="156900" lat="59.4334690" lon="24.7589795"/>
 <node id="401320878" visible="true" version="2" changeset="3688085" timestamp="2010-01-22T23:12:02Z" user="k__" uid="156900" lat="59.4326792" lon="24.7596313"/>
 <node id="401320879" visible="true" version="3" changeset="13481891" timestamp="2012-10-13T17:25:58Z" user="nomad08" uid="88466" lat="59.4325143" lon="24.7597486"/>
 <node id="401320880" visible="true" version="2" changeset="3688085" timestamp="2010-01-22T23:12:02Z" user="k__" uid="156900" lat="59.4321265" lon="24.7597131"/>
 <node id="618421702" visible="true" version="2" changeset="13481891" timestamp="2012-10-13T17:25:58Z" user="nomad08" uid="88466" lat="59.4323714" lon="24.7597384"/>
 <node id="634445962" visible="true" version="1" changeset="3834549" timestamp="2010-02-09T18:11:12Z" user="k__" uid="156900" lat="59.4333749" lon="24.7590408"/>
 <node id="880460688" visible="true" version="3" changeset="43392329" timestamp="2016-11-03T22:56:01Z" user="Volker Fröhlich" uid="2248331" lat="59.4335463" lon="24.7595669">
  <tag k="amenity" v="parking_entrance"/>
  <tag k="maxheight" v="2.2"/>
  <tag k="parking" v="underground"/>
 </node>
 <node id="1575989196" visible="true" version="1" changeset="10284871" timestamp="2012-01-03T19:40:16Z" user="plush" uid="83472" lat="59.4334130" lon="24.7590160"/>
 <node id="1823920171" visible="true" version="1" changeset="12204061" timestamp="2012-07-13T08:35:27Z" user="ilmarkerm" uid="726427" lat="59.4326611" lon="24.7596415"/>
 <way id="34570234" visible="true" version="8" changeset="38915196" timestamp="2016-04-27T09:54:51Z" user="Chaman78" uid="2093458">
  <nd ref="401316643"/>
  <nd ref="1575989196"/>
  <nd ref="634445962"/>
  <nd ref="401320878"/>
  <nd ref="1823920171"/>
  <nd ref="401320879"/>
  <nd ref="618421702"/>
  <nd ref="401320880"/>
  <tag k="highway" v="residential"/>
  <tag k="name" v="Maakri"/>
  <tag k="oneway" v="yes"/>
 </way>
 <way id="144018556" visible="true" version="1" changeset="10284871" timestamp="2012-01-03T19:40:16Z" user="plush" uid="83472">
  <nd ref="1575989196"/>
  <nd ref="880460688"/>
  <tag k="highway" v="service"/>
 </way>
 <relation id="8855425" visible="true" version="1" changeset="63925543" timestamp="2018-10-27T11:29:37Z" user="Pikse" uid="1748728">
  <member type="way" ref="34570234" role=""/>
  <member type="way" ref="88403642" role=""/>
  <member type="way" ref="638498969" role=""/>
  <tag k="name" v="Maakri tänav"/>
  <tag k="network" v="ee:local"/>
  <tag k="ref" v="7840745"/>
  <tag k="route" v="road"/>
  <tag k="type" v="route"/>
  <tag k="wikidata" v="Q12369357"/>
 </relation>
</osm>
```

Figure 2. An example of an OSM file.

Additionally, osm2pgsql, osm2pgrouting and osm2postgresql are only meant for usage with Linux operating system. For these reasons, a separate algorithm named XMLParser was created for extracting OSM data and inserting it to the database for our proposed method.

XMLParser takes an OSM file, database name and credentials as an input. It iterates over the XML object tree using dom4j framework. Dom4j is an open source XML framework for Java that enables reading, writing and parsing XML documents. [23] Next, it detects the types of the XML elements and creates the corresponding suitable objects for insertion in the database. Finally, it inserts the objects into the database.

3.3.2 Storing the Data for Private Modes

The data for private modes is stored in a set of PostgreSQL database tables. XMLParser does not distinguish modes and inserts the data for all modes in common tables. In order to optimize

the workload and size of the database tables and simplify the queries made to the database tables, there are separate tables for each mode. The scripts used for creating the database tables and distributing the data for each mode can be found in appendix 1.

3.3.3 The Algorithm for Pathfinding in Private Mode

The algorithm for finding the fastest path in private mode takes start location and destination location, time of departure and mode as an input. The input location data are two pairs formatted as latitude, longitude. If no time of departure is given, the current date and time is used.

Firstly, the modes inserted by the user are detected. For each private mode a query to extract the road data for the mode is executed. The inserted start and end latitude and longitude pairs are taken into account as the bounds of the queried region. The minimum and maximum latitude and longitude bounds are widened by 0.02 to each direction in order to include the most probable roads for the shortest path. The data extracted from the database is located in arrays and maps as nodes, ways and restrictions.

Secondly, Haversine distance between the source and target location is calculated and analyzed. Haversine formula is used when calculating the distance between two points on the Earth's surface specified in longitude and latitude [24]. The Haversine formula can be seen on figure 3.

$$d = 2r\sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right)+\cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\psi_2-\psi_1}{2}\right)}\right)$$

Figure 3. The Haversine formula. D = distance between two points (Ψ, φ), r = Earth's radius [24]

For each of the private modes there are constraints about the minimal allowed length of the trajectory travelled using this mode. The shortest path for car mode is only computed if Haversine distance between the two points is at least 1000 meters. The shortest path in bicycle mode is computed if Haversine distance between the two points is at least 500 meters. If Haversine distance between the two points is shorter than 100 meters the direct shortcut using dotted line is given instead of using the algorithm for searching the shortest path. The reason for this action is that as the road data in the database is a sparse data and does not include every possible path, the probability of finding a suitable feasible path for a distance shorter than 100 meters is quite low. Instead, in most cases it is possible to traverse this path directly without using any marked road. The direct shortcut with dotted line is used between the source and

target location inserted by the user and correspondingly the closest start and end points of the trajectory computed by the shortest path algorithm as well to give the user as accurate estimated trajectory and travel time as possible. In all cases the path marked with dotted line is considered to be of freestyle mode that is a type of pedestrian mode without using roads. The speed for traversing segments in this mode is estimated to be 3 km/h.

If Haversine distance between the source and target locations is feasible, the shortest path between the two locations is found using a special algorithm. In order to find the fastest path between the start and end locations, a modified version of A Star algorithm is used with the extracted data. The pseudo code for the original A Star algorithm can be seen in figure 4.

```
A* (start, goal)
Closed set = the empty set
Open set = includes start node
G[start] = 0, H[start] = H_calc[start, goal]
F[start] = H[start]
While Open set ≠ ∅
   do CurNode ← EXTRACT-MIN- F(Open set)
   if ( CurNode  == goal ), then return BestPath
   For each Neighbor Node N of CurNode
      If ( N is in Closed set ), then Nothing
      else if ( N is in Open set ),
         calculate N's G, H, F
         If ( G[N on the Open set] > calculated G[N] )
            RELAX(N, Neighbor in Open set, w)
            N's parent=CurNode & add N to Open set
      else, then calculate N's G, H, F
            N's parent = CurNode & add N to Open
```

Figure 4. The pseudo code for A Star algorithm. [25]

The pseudo code for the algorithm can be seen in figure 5.

```
findBestPath(s, t)
    add s openList
    calculate F score for s,t and put it in F Score Map
    put F score and s to priorities map
    initialize G score map with s, 0
    while openList not empty
            get node c from path candidate with best priority
            remove c from openList and add it to closedList
            if c == t:
                    return either composing path for car or other mode
            else:
                    get allowed ways for c
                    for each way:
                            find target and add it to closedList if not yet present
                                    get G score for c
                                    calculate F score for c,t
                                    tentative G score = G score + F score
                                    get G score for t
                                    if t not in openList or tentative G score < G score for t:
                                            add c as previous step for t
                                            save way as prevoius way for t
                                            put G score to G score map for t
                                            calculate F score for t
                                            target F score = tentative G score + F score for t
                                            put target F score to F score map for t
                                            if t not in openList:
                                                    add t to openList
                                                    add t to priorities list with priority target F score
    return null
```

Figure 5. The pseudo code for finding best path in private transport mode.

A Star algorithm uses two different costs in finding the path with the smallest cost. As the first cost it uses the cost from the source location of the path to the current node under the consideration. For each processed node it keeps in a separate map the cost from the source location. As the second cost it uses an estimated cost from the current node under the consideration to the target location of the path. At every iteration the corresponding costs are updated.

In the modified version of A Star used in our proposed method, the search starts with extracting all road segments that have the start node as their starting point. Next, the segments that are not allowed to be traversed due to some kind of restriction are filtered out. However, restrictions are not applied in pedestrian mode as they are intended for marking traffic regulations for vehicles. For each of the remaining segments in this result set the cost from start to the target node of the segment is calculated. In case car mode is used, the cost is defined as following: travel time of the segment is extracted from the list of travel times calculated taking into account the speed limits of the roads. Next, the estimated travel time to the target node of the path to be found is calculated. These two costs summed into one mark the priority of the candidate path. In other cases, the distance between the two nodes is used in calculating the costs to boost the performance. This way the algorithm uses the main core of A Star algorithm

described earlier. In all cases it is clear that the smaller the cost, the shorter the estimated travel time.

At each iteration the algorithm extends the path that minimizes the cost of the path between the source and the current node and the estimated cost between the current node and the target node. The cost in this case is the estimated travel time for this path. If the algorithm reaches dead end and there are no candidate paths left in the priority queue, it returns null value and the next pair of the nearest source and target nodes are fed to the algorithm. If the algorithm reaches to the target node, it returns the path and in case of car mode also the list of the way segments. This is important for the visualization process. As one pair of source and target node can define multiple different road segments, the correct way needs to be specified. Moreover, in case the road segment consists of several smaller segments, the correct way needs to be specified for extracting the right set of intermediate segments in order to visualize the road segment in its correct form, e.g. as a curve instead of a straight line.

3.4 Pathfinding in Public Mode

3.4.1 GTFS Data and Its Structure

Although, some amount of public transport data (stops, routes, service IDs) could be extracted from OSM server, it is not enough for time-dependent routing. For this reason, GTFS (General Transit Feed Specification) [26] data is used for finding trajectories in public mode in our proposed method. GTFS is an open standard format for exchanging public transportation schedule, geographic and fare information. [26] GTFS standard was developed by Google. [27] GTFS standard has become a generally approved format for publishing public transport data. It can be published by public transport agencies as a feed that can be used in web applications such as web maps. GTFS format consists of two formats: GTFS static that has data about scheduled service and GTFS Realtime that can be used for arrival predictions, vehicle positions and service advisories. [26] In this thesis paper GTFS static is used as GTFS static data for Estonia is provided for free by peatus.ee. [28]

The GTFS provided by Public Transport Information System on website peatus.ee is described in a document called “The Specification of the Open Data of the Registry of the Public Transport” (Ühistranspordiregistri avaandmete spetsifikatsioon) published by Republic of Estonia Road Administration (Maanteeamet).

The open data of the public transport register is downloadable in form of ZIP-file from the web page peatus.ee/gtfs. The data on the webpage is updated every day 6 AM (GMT +2) time. It is open and free to use for anybody. However, it is demanded that whenever using the data on any web application, web page, etc, the data shall not be older than 7 days from the moment of the data being published on the previously mentioned web page. [27] After unpacking this ZIP-file one can find 11 TXT-format files in it. The description of the files can be seen in table 1.

Table 1. The description of the files included in GTFS ZIP-file. [27]

agency.txt	The data of the conveyor.
calendar.txt	The data about the service dates of the trips.
calendar_dates.txt	The exceptions for the service dates of the trips.
feed_info.txt	The information about the provider of the GTFS data, Republic of Estonia Road Administration.
routes.txt	The data of the public transport routes.
stop_times.txt	The departure times of the trips from the stops.
stops.txt	The data of the public transport stops.
trips.txt	The data of the public transport trips.
fare_rules.txt	The rules for connecting the fares from the table of fare_attributes to the trips.
fare_attributes.txt	The fares used in the system.
shapes.txt	Describes the geometrical shapes of the trips.

Importing Data from GTFS to Database

For storing the GTFS data in tables, PostgreSQL database is used. The data is exported to multiple tables following the structure of the allocation of the data in the GTFS TXT-format files. The data is not exported from the files agency.txt, feed_info.txt, fare_rules.txt and fare_attributes.txt as the data in these files is not necessary for the algorithm. Additionally, to boost the performance of the database, separate tables are created for different public transport modes: train, bus, tram and trolley. The SQL commands used for creating the corresponding tables are shown in appendix 2. The entity relationship diagram of the database can be seen in figure 6.

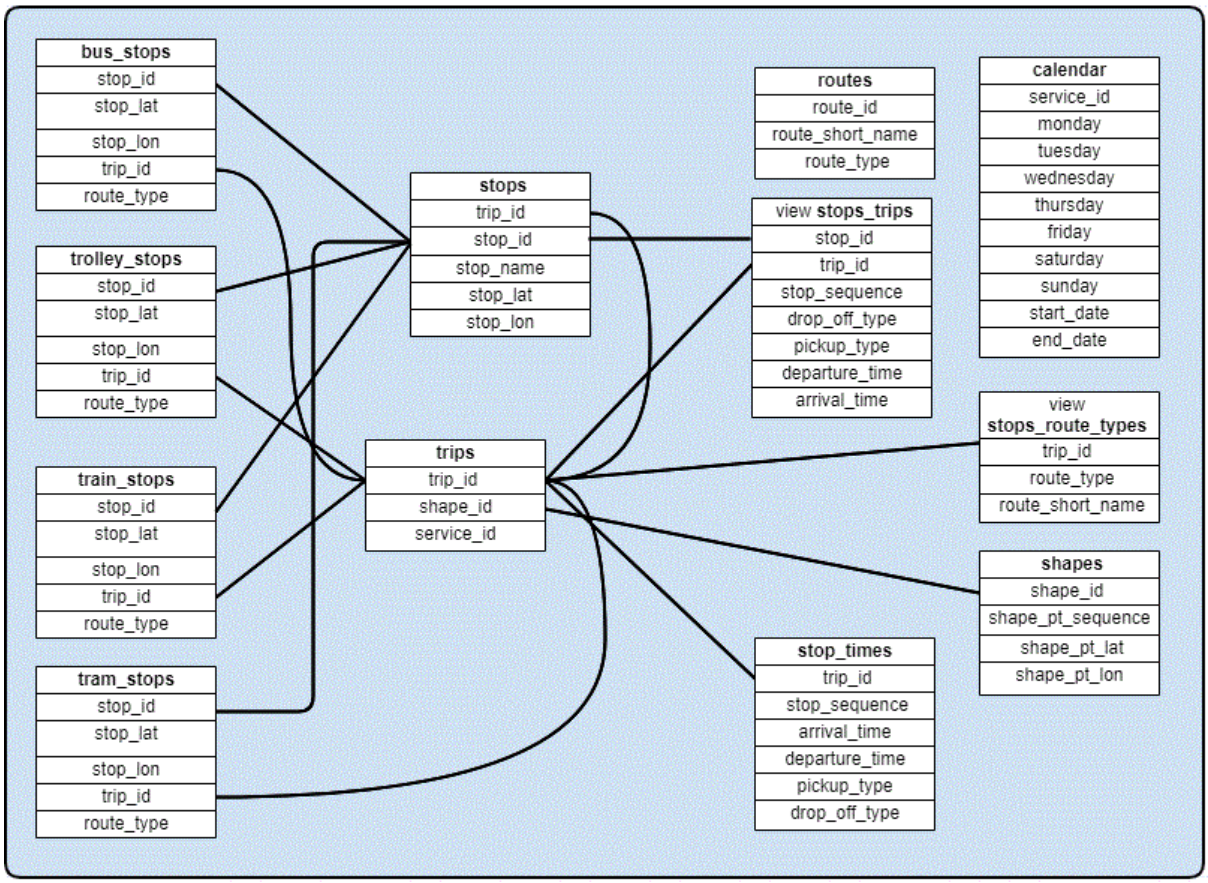

Figure 6. The entity relationship diagram of GTFS database tables and views.

3.4.2 The Algorithm for Pathfinding in Public Mode

The algorithm for pathfinding in public mode takes time, date and day of the departure, start and end location and the preferred means of public transport as parameters. In case time, date and day are not given, the current timestamp is used. In case the preferred public transport means are not specified, the algorithm includes all of the means of transport currently available in the database. The algorithm distinguishes 4 modes of public transportation: bus, trolleybus,

train and tram. Both intercity buses and transit buses are included in the bus mode. In case bus is specified as the preferred mean of public transport, only the data related to the bus mode is extracted from the database. If no suitable result is found among the available bus trips, no other mean of public transport will be considered. In case several means of public transport are given as preferred modes of public transport, the data of all these modes of public transport is extracted from the database. Taking into account the departure time, the source and target location and the preferred modes of public transport, the algorithm makes a queries to extract the necessary data for processing. Firstly, all the stops of the selected modes in the specified region are retrieved from the database. From this set the closest stop to the source location is found. Secondly, all the trips connected to this stop and currently in service are retrieved. The trips that do not belong to the suitable timeframe are filtered out. In this filtering process it is take into account that the departure times in the public transport timetable can differ up to 5 minutes from the reality. Next, from the remaining trips the trip reaching closest to the target location is found. No matter how long is the distance remaining from the end point to the target location of the trajectory, the algorithm always returns one public transport trip as maximum. No search for the subsequent trips is performed. This architectural choice has again been done due to the possible difference between the departure times in the timetable and the departure times in the reality. It would be highly likely that any of the trips would depart or arrive on a different time than in the timetable. If the suitable trip is found, the additional details about the trip e.g. route shape are extracted from the database. If the distance between the source and the target distance is less than 300 meters, no search for the public transport trip will be performed as such short distance would be unfeasible to travel using public transport. Thus, a route in pedestrian mode is returned instead The pseudo code for finding the best path in public transport mode can be seen on figure 7.

```
findPublicTransportRoute(x1, y1, x2, y2, transportTypes, time):
        extract data from database for transportTypes
        find earliest possible start time
        if trip start time > earliest possible start time:
                add to suitable trips
        if path exists:
                get candidate arrival stops for the suitable trips
                find best arrival stop candidate
                find fastest trip
                find all related data for the trip
                update trajectory data
                update duration
                return trajectory data
        return null
```

Figure 7. The pseudo code for finding the best path in public transport mode.

3.4.3 Constructing Multimodal Trajectories

Firstly, the allowed modes inserted by the user are examined. The private modes are added to a separate list for later use. Next, public transport routes are extracted from the database, the best public transport trip is found and the public mode trajectory is compound in the way described earlier. After finding the public mode trajectory the source and the target point of the trajectory are analyzed and compared to the source and the target location inserted by the user. If the source location of the trajectory differs from the location of the source point of the public transport trip adding a subtrajectory in private mode is considered. If the remaining route is long enough to feasibly change the mode, a subtrajectory in private mode is added. The missing intermediate path is found in the way described earlier under pathfinding in private mode. For each private mode that is allowed by the user the shortest path between these two locations is found. The same applies for the target location of the trajectory and the target point of the public transport trip as well. Next, the start and end points of all the found trajectories are analyzed and combined. If in the process of combining multimodal trajectories two segments of freestyle mode happen to be placed subsequently, the two segments are united into one segment of freestyle mode. Finally, from all the generated combinations, the trajectory with the shortest expected travel time is returned as the result. The pseudo code for finding best multimodal trajectories can be seen on figure 8.

```
findBestTrajectories(x1, y1, x2, y2, transportTypes, timestamp)
        add private transport modes to list
        findPublicTransportRoute(x1, y1, x2, y2, transportTypes, time)
        for each private transport mode:
                check feasibility for connecting with public transport
                if is feasible adding private mode transportation before public mode transportation:
                        findRoutesForMode(s, public mode s)
                else use default path
                if is feasible adding private mode transportation after public mode transportation:
                        findRoutesForMode(public mode t, t)
                else use default path
                merge trajectory data
                find fastest combination
        return fastest (multi)trajectories for selected transportTypes
```

Figure 8. The pseudo code for finding best multimodal trajectories.

3.4.4 Visualization Process

The data is stored in GeoJSON format. GeoJSON is a format that enables encoding a variety of geographic data structures. GeoJSON supports among others geometry types like Point, LineString, MultiLineString and Feature that are suitable for displaying trajectories on the map. [29] The results are visualized using a web map and OSM map tiles. The web map uses

Leaflet.js and jQuery libraries to display the data of the trajectories. Trajectories are displayed as MultiLineStrings.

4. Testing, Results and Analyses

4.1 Introduction

In this chapter the testing, results and analyses of our proposed method is described. The chapter is divided into subsections according to the type of testing done. Testing includes computation times comparison, the comparison of the results with three other routing engines (Google, OSRM and GraphHopper) and multimodal results comparison for different mode sets.

For testing the public transport and road network data for the city of Tallinn is used.

The pedestrian network includes 483604 ways, the bicycle network includes 345952 ways and the car network includes 121249 ways. In terms of the public transport network, there are 1756 stops (1477 bus stops, 75 tram stops, 72 trolley stops and 44 train stops), 33131 trips and 1982 routes in the database.

The details of the test environment are as follows: CPU Intel Core i5-6200U, RAM: 16.0GB, operating system: 64-bit Windows 10 Enterprise, database: PostgreSQL 10, IDE: IntelliJ IDEA 2017.3, programming language: Java.

4.2 Test Strategy

For testing three types of tests were executed. Firstly, the computation times for 14 different mode sets were measured. Each time the source and target coordinates of the trajectory were randomly generated. Secondly, 15 pairs of source and target coordinates were chosen randomly and for each of them trajectories were computed using our proposed method, Google, GraphHopper, OSRM. The similarity of the results was compared and analysed. Next, in order to test our proposed algorithm, 10 multimodal trajectories were selected. For each of them the corresponding multimodal trajectories were found using Google's routing and our proposed method. Lastly, some examples of multimodal trajectories using different mode sets are presented.

4.3 Test Results

4.3.1 The Average Calculation Time

The computation times for different modes and mode combinations can be seen in table 2. "Average calculation time" refers to complete computation time including reading in the

necessary data from databases, running the algorithm(s), putting together the information for the presentation to the user, converting the result into GeoJSON format and writing the result into the result file.

As can be seen from the table, the computation times increases with the number of modes. The average computation time is smaller when instead of selecting one mean of public transportation, like bus, all means of public transportation are allowed. This is probably due to the fact that computing times are by great extent smaller when any other mean of public transportation but bus is chosen. This occurs due to the fact that the amount of the data in the database tables related to the bus is at least about 20 times bigger than in the tables related to other means of public transport and thus the queries take more time to run and the processing of the data is slower as well.

Table 2. The computation times for different modes and mode combinations.

Mode	Average calculation time (s)
Pedestrian	52.06
Bicycle	35.33
Car	42.47
Pedestrian + bicycle	93.42
Pedestrian + car	65.47
Pedestrian + bus	132.73
Pedestrian + all public transport	89.75
Pedestrian + bicycle + car	105.48
Pedestrian + bicycle + bus	191.15
Pedestrian + bicycle + all public transport	183.41
Pedestrian + car + bus	191.52
Pedestrian + car + all public transport	155.66
Pedestrian + bicycle + car + bus	275.01
Pedestrian + bicycle + car + all public transport	258.14

4.3.2 Comparison with Google, OSRM and GraphHopper

As the next testing method, 15 pairs of source and target coordinates were chosen randomly and for each of them trajectories were computed using our proposed method, Google, OSRM and GraphHopper.

Google's routing algorithm allows to find trajectories in pedestrian mode, car mode and in public transport mode. Besides having pedestrian mode as the connecting mode of different trajectory segments, if needed, multimodality is not used. It is not possible to select suitable means of public transport for public transport mode. Bicycle mode is currently not available for Tallinn. In car mode both historical statistics and current situation of the road is taken into

account in path calculation. If source and target locations are inserted as pairs of latitude and longitude coordinates, Google replaces them with the nearest point it has in its database. This factor also needs to be taken into account while analysing the results as in many cases it can visibly change the result. One of the examples of such behaviour can be seen on figure 9.

GraphHopper allows to find trajectories in pedestrian mode, bicycle mode and car mode. Here it is used with road network data from OSM. [30]

OSRM is a routing server that provides route instructions for pedestrian, bicycle and car mode. [31] Both GraphHopper and OSRM use additional weights like U-turn penalty, traffic light penalty, etc in their calculations. [32][33]

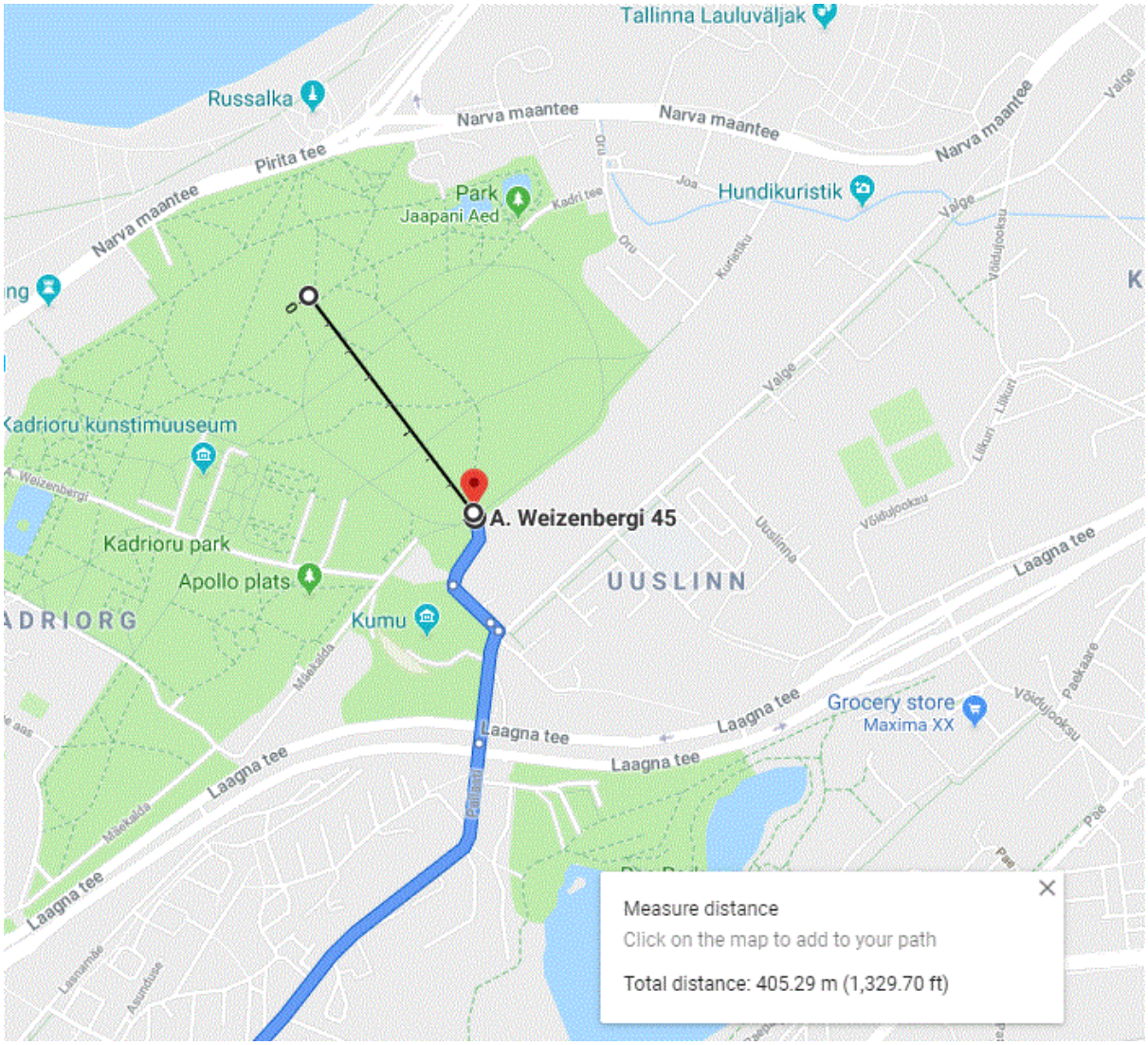

Figure 9. When inserting 59.440894, 24.793906 (upper white dot on the figure) as start point location in Google, it was automatically changed to 59.438057, 24.798133 (A. Weisenbergi street 45) which is more than 400 meters from the inserted start point.

Table 3. The percentages of the average similarity of the result trajectories between the results of the proposed algorithm and Google, GraphHopper and OSRM.

	Google	GraphHopper	OSRM
Pedestrian	72%	70%	67%
Bicycle	N/A	33%	37%
Car	48%	31%	40%
Public transport	42%	N/A	N/A

In table 3 the similarity percent of the result trajectories of the algorithm proposed in this thesis and the corresponding trajectories computed by Google, GraphHopper and OSRM are presented. The similarity percent has been found by comparing 15 result trajectories with the corresponding result trajectories computed by Google, GraphHopper and OSRM. The more

similar segments were found, the bigger is the similarity percent. As can be seen, the most similar are the results in pedestrian mode. This is due to the fact that in other modes, where traffic is taken into account, the penalties for U-turns, traffic lights, etc have more impact on the results.

Table 4. The percentages of the average difference of the distances of the result trajectories between the results of our proposed method and Google, GraphHopper and OSRM.

	Google	GraphHopper	OSRM
Pedestrian	1%	-3%	-1%
Bicycle	N/A	3%	-3%
Car	-1%	-2%	-2%
Public transport	N/A	N/A	N/A

In table 4 the average difference of the distances of the result trajectories are compared. As can be seen, the difference is minimal, only up to +/-3%. In addition, the difference is not similar in all cases. If the distances of trajectories in bicycle mode are on average 3% longer in GraphHopper than in the corresponding results of the proposed algorithm, then for OSRM it is exactly the opposite – the distances of the result trajectories in bicycle mode are on average 3% shorter than in the corresponding results of our proposed method. Thus, it appears that there is no certain similarity between the results of different comparison routers. In case of car mode, however, in all comparisons the average distance is slightly greater in the result trajectories of our proposed method.

Table 5. The percentages of the average difference of the durations of the result trajectories between the results of our proposed method and Google, GraphHopper and OSRM.

	Google	GraphHopper	OSRM
Pedestrian	2%	-5%	6%
Bicycle	N/A	-4%	5%
Car	5%	-16%	-29%
Public transport	20%	N/A	N/A

In table 5 the average difference of the durations of the result trajectories are compared. As can be seen the difference is the biggest for results in car mode. This occurs due to the fact that all of the three – Google, GraphHopper and OSRM use additional costs for trajectories besides the average speed. Google greatly relies on the historical data and the current state of the roads. Thus, as expected the durations computed differ on average up to 29 % from the durations computed by our proposed method. In case of public mode, the durations differ mostly due to the differences of the algorithms. Mostly the durations differ in public mode when instead of

using one trip, multiple trips are included in the result trajectory. Thus, the first trip is mostly the same, but the second part of the trajectory is different: in case of our proposed method the trajectory continues in one of the public modes and in case of Google another public transport trip is added in combination with the pedestrian mode.

4.3.3 Testing in the real environment

In order to test our proposed algorithm, 10 multimodal trajectories were selected. For each of them the corresponding multimodal trajectories were found using Google's routing and our proposed algorithm. As multimodal routing is not available in OSRM and in GraphHopper, they were not included in the comparison. Next, the travel time of all 10 trajectories were measured in real life. The results have been compared in the table 6.

Table 6. The comparison of the estimated travel time calculated by our proposed method, Google's routing and the travel time in reality.

Trajectory	Our proposed method (s)	Google (s)	Reality (s)	Error in percentage (our proposed method)	Error in percentage (Google)
1	1795	1620	1829	-2%	-11%
2	780	840	806	-3%	4%
3	1199	1140	1240	-3%	-8%
4	1825	1680	2397	-24%	-30%
5	2041	2880	2106	-3%	37%
6	1336	1380	1771	-25%	-22%
7	2832	2700	2448	16%	10%
8	2632	2160	2786	-6%	-22%
9	2098	2040	2004	5%	2%
10	1492	1380	1845	-19%	-25%

As can be seen from the table, the real travel time differs from both the estimated travel time of Google's routing and from the estimated travel time of our proposed method. In some extreme cases the difference between the estimated travel time and the real travel time was up to 37%. These kind of differences occurred mostly in longer pedestrian mode distances. In most cases the estimated travel times are shorter than the travel time in reality. This could be mostly due to the difference of the speed of the traveller in the estimation and in the reality. Our proposed method assumes that the average speed in pedestrian mode is 5 km/h on the roads and 3 km/h on undefined path. The average speed that Google's routing uses in its computation in pedestrian mode depends on the route. For example, for Google's routing the speed for

walking uphill differs from the speed for walking downhill [1][34]. The average pedestrian speed for these 10 trajectories computed by Google's routing was 4-6 km/h. As a result, the real travel time of 3 of 10 trajectories was more similar to the estimated travel time computed by Google's routing and for the remaining 7 cases the estimated travel time by our proposed method proved to be closer to the reality.

4.3.4 Multimodality Comparison

Our proposed method enables selecting different mode combinations as user input. The results are computed taking into account the selected modes and the result trajectories are either unimodal or multimodal trajectories that include only the selected modes. Figure 10 and figure 11 show different mode combinations in different trajectories. Our proposed method finds the results among the allowed modes inserted by the user. If user does not select the mode, it does not appear in the results. The only exception is pedestrian mode, that is always present. By doing this, user either receives the trajectory composed by the selected modes or, if there is no answer available among the selected modes, no answer is given. This behaviour can be seen on figure 10 and figure 11, For figure 10 the allowed modes are pedestrian, bicycle and train and for the result on the figure 11 the allowed modes are pedestrian, car and train. In these queries using train as the only mean of public transport is guaranteed. Even if there would be a faster trip in some other public transport mode, train would still be chosen as user's input is more important than the expected travel time.

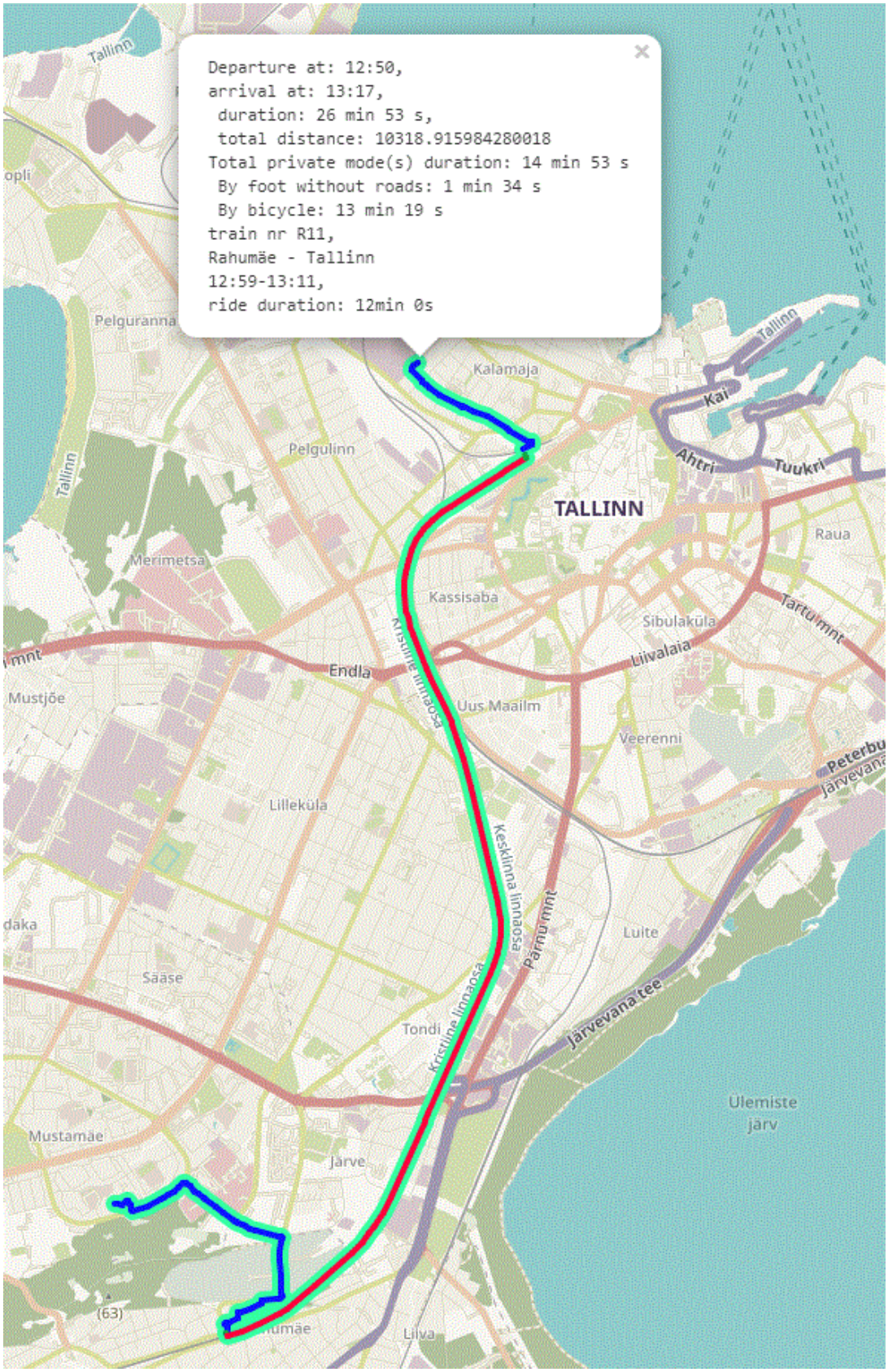

Figure 10. A multimodal trajectory combining public and bicycle mode.

Figure 11. A multimodal trajectory combining public and car modes.

5. Conclusion and Future Work

The ongoing growth of urbanization and the growth of the cities is leading to the increase of complexity of the route planning in urban areas. At the same time people's expectancies to the speed, comfort, flexibility and variety of different choices keeps increasing as well. People are conscious about the possibility to travel in public transport in a wheelchair, with a baby carriage or with a bicycle. However, finding a trip that would be the fastest possible at the same time allowing user to set constraints for the used modes, is rather time consuming. Hence, there is a need for flexible route planning algorithms that would easily allow user to find suitable trajectories taking into account user's preferences about the transport modes used in the journey to get the most of the possibilities of using public transport.

The aim of this thesis was to propose an alternative method for multimodal fastest pathfinding with use of public transportation. The method uses three algorithms in a subsequent way to compute the subtrajectories for the fastest path and compose a multimodal trajectory of them. In finding the best path, the method takes into account the data inserted by the user as well, to provide the best possible solution in accordance with the needs of the user.

In this thesis work an alternative method for multimodal fastest pathfinding with use of public transportation was developed. Firstly, the real data of the city of Tallinn both for private and public modes was extracted from the sources, processed and inserted into the databases. After that, the data from the databases was used for multimodal pathfinding with the developed method containing the algorithm for pathfinding in private modes, the algorithm for pathfinding in public modes and an algorithm for combining and constructing multimodal trajectories. All in all, the aim of this thesis was met and our proposed method was able to propose competitive alternatives to the results of the existing routing engines at the same time taking into account user's preferences about the transport modes used in the journey.

However, in future work several improvements could be done. Firstly, instead of using Java to implement the trajectory reconstruction, the algorithm could be written in Python. The database tables could be rearranged and additional views and indexes can be created. This would help to make the code less memory and time consuming. Another improvement would be to try to use a combination of graph database and relational database instead of regular relational database. The combination of the two could help to speed up the computation and minimize the complexity of the algorithm.

References

[1] Elron webpage https://elron.ee/teenused/teenused-rongides/mugavus/ (16.05.2019)

[2] Safer, Hershel & Orlin, James & B, James. “Fast Approximation Schemes for Multi-Criteria Combinatorial Optimization” (1995)

[3] Tsolkas, D., Passas, N., Xenakis, C., Papataxiarhis, V., Tsetsos, V. “Busfinder: a personalized multimodal transportation guide with dynamic routing” 2012 16th Panhellenic Conference on Informatics: 25-30, (2012)

[4] Liu, L. “Data Model and Algorithms for Multimodal Route Planning with Transportation Networks” Lehrstuhl für Kartographie, Technische Universität München (2010)

[5] Campigotto, P., Rudloff, C., Leodolter, M., Bauer, D. “Personalized and Situation-Aware Multimodal Route Recommendations: The FAVOUR Algorithm”, IEEE Transactions on Intelligent Transportation Systems: 92-102 (2017)

[6] Mouncif, H., Rida, M., Boulmakoul, A. “An Efficient Multimodal Path Computation Integrated Within Location based Service for Transportation Networks System (Multimodal Path Computation within LBS)”, Journal of Applied Sciences 11 (1): 1-15 (2011)

[7] Zhang, J., Liao, F., Arentze, T., Timmermans, H. “A Multimodal Transport Network Model of Advanced Traveler Information Systems”, Procedia Social and Behavioral Sciences 20: 313-322 (2011)

[8] Chondrogiannis, T., Cavaliere, R., Gamper, J., Ohnewein, P. “MoTrIS: A Framework for Route Planning on Multimodal Transportation Networks” (2016)

[9] Guo, C., Li, D., Zhang, G., Cui, Z. "Data delivery delay estimation based on left-turn in vehicular ad hoc networks," *2016 International Conference on Progress in Informatics and Computing (PIC)*, Shanghai, 2016, pp. 550-554.

[10] Kirchler, Dominik et al. “Efficient Computation of Shortest Paths in Time-Dependent Multi-Modal Networks.” ACM Journal of Experimental Algorithmics 19 (2014)

[11] Pajor, T. “Multi-Modal Route Planning” Institut für Teoretische Informatik, Universität Karlsruhe (2009)

[12] Dibbelt, J., Pajor, T., Wagner, D. 2015. User-Constrained Multimodal Route Planning. J. Exp. Algorithmics 19, Article 3.2 (2015)

[13] Ma, T.-Y. “On-demand Dynamic Bi-/multi-modal Ride-sharing using Optimal Passenger-vehicle Assignments”, 2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe): 1-5 (2017)

[14] K. G. Zografos and K. N. Androutsopoulos, "Algorithms for Itinerary Planning in Multimodal Transportation Networks," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 175-184, (2008)

[15] Elbery, Ahmed & Dvorak, Filip & Du, Jianhe & Rakha, Hesham & Klenk, Matthew Large-scale Agent-based Multi-modal Modeling of Transportation Networks - System Model and Preliminary Results. 103-112 (2018)

[16] OpenStreetMap webpage osm.org (02.05.2019)

[17] OSM XML webpage https://wiki.openstreetmap.org/wiki/OSM_XML (02.05.2019)

[18] JOSM webpage https://josm.openstreetmap.de/ (02.05.2019)

[19] Osmosis webpage https://wiki.openstreetmap.org/wiki/Osmosis (02.05.2019)

[20] osm2postgresql webpage https://sourceforge.net/projects/osm2postgresql/ (02.05.2019)

[21] osm2pgrouting webpage http://pgrouting.org/docs/tools/osm2pgrouting.html (02.05.2019)

[22] osm2pgsql webpage https://wiki.openstreetmap.org/wiki/Osm2pgsql (02.05.2019)

[23] dom4j webpage https://github.com/dom4j/dom4j/wiki/FAQ (02.05.2019)

[24] Chopde, Nitin R., and M. Nichat. "Landmark based shortest path detection by using A* and Haversine formula." International Journal of Innovative Research in Computer and Communication Engineering 1.2 (2013): 298-302.

[25] W. Seo, S. Ok, J. Ahn, S. Kang and B. Moon, "An Efficient Hardware Architecture of the A-star Algorithm for the Shortest Path Search Engine," 2009 Fifth International Joint Conference on INC, IMS and IDC, Seoul, 2009, pp. 1499-1502.

[26] GTFS webpage http://gtfs.org/ (02.05.2019)

[27] Public Transport Information System

https://www.mnt.ee/eng/public-transportation/public-transport-information-system

(08.02.2018)

[28] peatus.ee webpage http://peatus.ee/ (02.05.2019)

[29] GeoJson webpage https://geojson.org/ (07.05.2019)

[30] GraphHopper webpage https://www.graphhopper.com/open-source/ (20.04.2019)

[31] OpenStreetMap webpage https://routing.openstreetmap.de/about.html (20.04.2019)

[32] GraphHopper webpage https://github.com/graphhopper (20.04.2019)

[33] FossGIS routing server webpage https://github.com/fossgis-routing-server (20.04.2019)

[34] Google Maps Help https://support.google.com/maps/forum/AAAAQuUrST8PHwS75mwUiY/?hl=en&gpf=%23!topic%2Fmaps%2FPHwS75mwUiY (27.04.2019)

Appendix 1

The SQL commands for creating database tables for private routing data:

```
create database privatemodesdb;

create table osmways (temp_id serial, osm_id bigint,
length_m double precision, source_osm bigint, target_osm bigint, source bigint, target bigint,
one_way integer, type character varying,
parent_way bigint,parent_temp_id bigint, sequence integer, x1 double precision, y1 double precision, x2 double precision, y2 double precision);

create table osmnodes (temp_id serial, osm_id bigint, lat double precision, lon double precision);
create table restrictions (source_way bigint, target_way bigint, node bigint, restriction_type character varying, is_forbidden_restriction boolean);
create table osmparents (temp_id serial, osm_id bigint);

UPDATE osmways
SET parent_temp_id = osmparents.temp_id
FROM osmparents
WHERE osmparents.osm_id = osmways.parent_way;

UPDATE restrictions
SET source_way = osmparents.temp_id
FROM osmparents
WHERE osmparents.osm_id = restrictions.source_way;

UPDATE restrictions
SET target_way = osmparents.temp_id
FROM osmparents
WHERE osmparents.osm_id = restrictions.target_way;

UPDATE osmways
SET source = osmnodes.temp_id
FROM osmnodes
WHERE osmnodes.osm_id = osmways.source_osm;

UPDATE osmways
SET target = osmnodes.temp_id
FROM osmnodes
WHERE osmnodes.osm_id = osmways.target_osm;

CREATE TABLE pedestrian (temp_id serial, osm_id bigint,
length_m double precision, source_osm bigint, target_osm bigint, source bigint, target bigint,
one_way integer, type character varying,
parent_way bigint,parent_temp_id bigint, sequence integer, x1 double precision, y1 double precision, x2 double precision, y2 double precision);

INSERT INTO pedestrian(length_m, source_osm, target_osm, source, target, one_way, type,
sequence,parent_way, parent_temp_id, x1, y1, x2, y2) SELECT
length_m, source_osm, target_osm, source, target, one_way, type,
sequence,parent_way,parent_temp_id, x1, y1, x2, y2 FROM osmways;

CREATE TABLE bicycle(temp_id serial, osm_id bigint, length_m double precision, source_osm bigint, target_osm bigint, source bigint, target bigint, one_way integer,
type character varying, parent_way bigint,parent_temp_id bigint, sequence integer, x1 double precision, y1 double precision, x2 double precision, y2 double precision);

INSERT INTO bicycle(length_m, source_osm, target_osm, source, target, one_way, type,
sequence,parent_way, parent_temp_id, x1, y1, x2, y2) SELECT
length_m, source_osm, target_osm, source, target, one_way, type,
sequence,parent_way,parent_temp_id, x1, y1, x2, y2 FROM pedestrian;

DELETE FROM bicycle where type = 'steps' OR type = 'footway' OR type = 'pedestrian' OR type = 'null' OR type = 'unclassified' OR type = 'construction';

CREATE TABLE car(id serial, length_m double precision, speed integer, source bigint, target bigint, type character varying, parent_way bigint, sequence integer,
x1 double precision, y1 double precision, x2 double precision, y2 double precision);

INSERT INTO car(length_m, speed, source, target, type, parent_way ,sequence, x1, y1, x2, y2) SELECT
length_m, speed, source, target, type, parent_way, sequence, x1, y1, x2, y2 FROM bicycle;

DELETE FROM car where type = 'cycleway' OR type = 'proposed' OR type = 'path';

INSERT INTO pedestrian(length_m, source_osm, target_osm, source, target, one_way, type,
sequence, parent_way,parent_temp_id, x1, y1, x2, y2) SELECT
length_m, target_osm, source_osm, target, source, one_way, type,
sequence, parent_way,parent_temp_id, x2, y2, x1, y1 FROM osmways;
```

Appendix 2

The SQL commands for creating database tables for public transport data:

```
create table shapes (shape_id integer, shape_pt_lat double precision,
shape_pt_lon double precision, shape_pt_sequence integer);

create table stops (stop_id integer, stop_code character varying, stop_name character varying,
stop_lat double precision, stop_lon double precision,  zone_id integer, alias character varying,
stop_area character varying, stop_desc character varying, lest_x double precision,
lest_y double precision, zone_name character varying);

create table routes (route_id character varying, agency_id integer,
route_short_name character varying, route_long_name character varying, route_type integer,
route_color character varying, competent_authority character varying);

create table trips (route_id character varying, service_id integer, trip_id integer,
trip_headsign character varying, trip_long_name character varying,
direction_code character varying, shape_id integer, wheelchair_accessible integer);

create table stop_times (trip_id integer, arrival_time character varying,
departure_time character varying, stop_id integer, stop_sequence integer, pickup_type integer,
drop_off_type integer);

create table calendar_dates (service_id integer, date integer, exception integer);

create table calendar(service_id integer, monday boolean,tuesday boolean,wednesday boolean,
thursday boolean,friday boolean ,saturday boolean,sunday boolean, start_date integer,end_date integer);
```

```
CREATE VIEW stops_trips AS
SELECT stops.stop_id,
   stops.stop_lon,
   stops.stop_lat,
   stop_times.trip_id,
   stop_times.stop_sequence,
   stop_times.arrival_time,
   stop_times.departure_time,
   stop_times.pickup_type,
   stop_times.drop_off_type
  FROM stops
    JOIN stop_times ON stops.stop_id = stop_times.stop_id;
```

```
CREATE view stops_route_types AS
 SELECT stops_trips.stop_id,
   stops.stop_name,
   trips.trip_id,
   trips.shape_id,
   routes.route_id,
   routes.route_type,
   routes.route_short_name
   FROM stops,
   stops_trips,
   trips,
   routes
  WHERE stops_trips.trip_id = trips.trip_id AND trips.route_id::text = routes.route_id::text;
```

```
DELETE FROM stops WHERE NOT (stop_lon >=24.4698 AND stop_lon<= 25.0745 AND stop_lat>=59.3245 AND stop_lat<= 59.6127);

CREATE TABLE train_stops (stop_id integer, stop_lat double precision, stop_lon double precision, trip_id integer, route_type  integer);

CREATE TABLE tram_stops (stop_id integer, stop_lat double precision, stop_lon double precision, trip_id integer, route_type  integer);

CREATE TABLE bus_stops (stop_id integer, stop_lat double precision, stop_lon double precision, trip_id integer, route_type integer);

CREATE TABLE trolley_stops (stop_id integer, stop_lat double precision, stop_lon double precision, trip_id integer, route_type  integer);

INSERT INTO train_stops (stop_id, trip_id, route_type) SELECT stops_trips.stop_id, trips.trip_id, route_type
FROM stops_trips, trips, routes
WHERE stops_trips.trip_id = trips.trip_id and trips.route_id = routes.route_id and route_type=2;

UPDATE train_stops
SET stop_lat = stops.stop_lat,
 stop_lon = stops.stop_lon
FROM stops
WHERE stops.stop_id = train_stops.stop_id;

INSERT INTO tram_stops (stop_id, trip_id, route_type) SELECT stops_trips.stop_id, trips.trip_id, route_type
FROM stops_trips, trips, routes
WHERE stops_trips.trip_id = trips.trip_id and trips.route_id = routes.route_id and route_type=0;

UPDATE tram_stops
SET stop_lat = stops.stop_lat,
 stop_lon = stops.stop_lon
FROM stops
WHERE stops.stop_id = tram_stops.stop_id;

INSERT INTO bus_stops (stop_id, trip_id, route_type) SELECT stops_trips.stop_id, trips.trip_id, route_type
FROM stops_trips, trips, routes
WHERE stops_trips.trip_id = trips.trip_id and trips.route_id = routes.route_id and route_type=3;

UPDATE bus_stops
SET stop_lat = stops.stop_lat,
 stop_lon = stops.stop_lon
FROM stops
WHERE stops.stop_id = bus_stops.stop_id;

INSERT INTO trolley_stops (stop_id, trip_id, route_type) SELECT stops_trips.stop_id, trips.trip_id, route_type
FROM stops_trips, trips, routes
WHERE stops_trips.trip_id = trips.trip_id and trips.route_id = routes.route_id and route_type=800;

UPDATE trolley_stops
SET stop_lat = stops.stop_lat,
 stop_lon = stops.stop_lon
FROM stops
WHERE stops.stop_id = trolley_stops.stop_id;
```

The ‘delete’ query is run in order to only keep the stops that are in selected (Tallinn) area.

I. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Careelika Liisi Kuik,**

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Multimodal Route Planning Algorithm for Encouraging the Usage of Different Means of Public Transportation,

(*title of thesis*)

supervised by Amnir Hadachi

(*supervisor's name*)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Careelika Liisi Kuik

17/05/2019