Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Daniel S l u n e č k o

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Using GPS Traces for Journey Duration Estimates in Transport Networks

Guidelines:

1. Survey existing methods for matching GPS traces to directed graph representation.
2. Formalise the GPS traces graph matching problem and a model for the journey duration estimates.
3. Use an existing GPS traces matching algorithm and propose a method to extract an instantiation of the model for the journey duration estimates.
4. Implement the proposed solution.
5. Evaluate the implemented solution using generated and real-world GPS traces data.

Bibliography/Sources:

[1] Helmut Alt,Alon Efrat,Günter Rote,and Carola Wenk – Matching planar maps – 2003
[2] Sotiris Brakatsoulas,Dieter Pfoser,Randall Salas,Carola Wenk - On Map-Matching Vehicle Tracking Data – 2005.
[3] Pablo Samuel Castro,Daqing Zhang,Shijian Li Urban Traffic Modelling and Prediction Using Large Scale Taxi GPS Traces – 2012.

Bachelor Project Supervisor: Mgr. Jan Hrnčíř

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
Head of Department	**Dean**

Prague, January 14, 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Daniel S l u n e č k o

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Využití GPS záznamů tras pro odhady dojezdových časů v dopravních sítích

Pokyny pro vypracování:

1. Prozkoumejte existující metody pro mapování GPS tras do reprezentace orientovaného grafu.
2. Formalizujte problém mapování GPS tras na graf a model pro odhadování dojezdových dob.
3. Použijte existující algoritmus mapování GPS tras a navrhněte metodu pro extrakci instance modelu pro odhadování dojezdových dob.
4. Implementujte navrhnuté řešení.
5. Vyhodnoťte implementované řešení na základě generovaných a reálných GPS tras.

Seznam odborné literatury:

[1] Helmut Alt,Alon Efrat,Günter Rote,and Carola Wenk – Matching planar maps – 2003
[2] Sotiris Brakatsoulas,Dieter Pfoser,Randall Salas,Carola Wenk - On Map-Matching Vehicle Tracking Data – 2005.
[3] Pablo Samuel Castro,Daqing Zhang,Shijian Li Urban Traffic Modelling and Prediction Using Large Scale Taxi GPS Traces – 2012.

Vedoucí bakalářské práce: Mgr. Jan Hrnčíř

Platnost zadání: do konce letního semestru 2015/2016

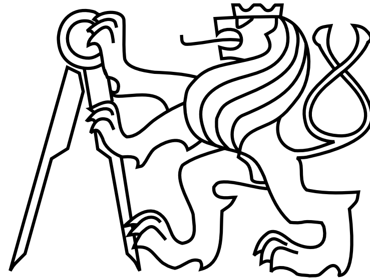
L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 14. 1. 2015

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor's Project

Using GPS Traces for Journey Duration Estimates in Transport Networks

Daniel Slunečko

Supervisor: Mgr. Jan Hrnčíř

Study Programme: Otevřená informatika - B1806/10302204

Field of Study: Informatika a počítačové vědy

May 21, 2015

Aknowledgements

My sincere thanks go to my supervisor Mgr. Jan Hrnčíř for guiding this work. I also thank my family for supporting me throughout the semester.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

Podpis autora práce

Abstract

The topic of this work is using GPS traces for journey duration estimates in transport networks. The GPS traces graph matching problem has been formalized as well as a model for the journey duration estimates. The work describes an implementation of existing GPS traces matching algorithm and proposes a method for journey duration estimates model instantiation. The proposed method for model instantiation has been described and implemented. The solution has been evaluated using generated and real-world data. We have created a speed profile for the graph representing a road network of the Prague city using mappings of 10000 randomly generated traces for each of the two time intervals it is defined for. We also created instances of speed profile for graph of Brno based on various numbers of generated traces to show how the percentage coverage changes.

Keywords

Journey duration estimates; Speed profile; Map-matching of traces

Abstrakt

Tématem této práce je využití GPS záznamů tras pro odhady dojezdových časů v dopravních sítích. Problém přiřazení odpovídající cesty v grafu k danému záznamu GPS trasy je formalizován společně s modelem pro odhady dojezdových časů. Práce popisuje implentaci známeho algoritmu pro přiřazení cesty v grafu k danému záznamu GPS trasy a navrhuje metodu pro vytvoření modelu, který reprezentuje odhady dojezdových časů. Tato navržená metoda byla popsána a implementována. Řešení bylo otestováno s využitím uměle generovaných i reálných dat. Pro dva časové úseky byl nad grafem reprezentujícím silniční síť města Prahy vytvořen rychlostní profil. Pro oba časové úseky bylo využito 10000 generovaných tras. Rovněž byly vytvořeny rychlostní profily na základě různých počtů tras pro Brno, abychom ukázali, jakým způsobem to ovlivní procentuální pokrytí grafu.

Klíčová slova

Odhady dojezdových časů; Rychlostní profil; Mapování tras

Contents

1 Introduction 1
1.1 Motivation . . . 1
1.2 Aim of the Thesis . . . 1
1.3 Structure of the Thesis . . . 2

2 Related Work 3

3 Problem Representation 5
3.1 Used Theory . . . 5
3.2 Data Representation . . . 7
3.2.1 Input . . . 7
3.2.2 Inner Representation . . . 8
3.2.3 Speed Profile . . . 8
3.2.4 GPS Traces to Speed Profile Problem . . . 9

4 Solution Approach 11
4.1 Mapping Traces Onto a Graph . . . 12
4.1.1 Preprocessing . . . 13
4.1.2 Path Search . . . 14
4.1.3 Path Reconstruction . . . 15
4.1.4 Mapping Algorithm Improvement . . . 15
4.2 Data Extraction . . . 15
4.2.1 Projection . . . 15
4.2.2 Interpolation . . . 18
4.3 Model Instantiation . . . 18

5 Implementation 21
5.1 Architecture . . . 21
5.2 Testing . . . 22

6 Evaluation 23
6.1 Mapping Precision . . . 23
6.1.1 Experiment Settings . . . 23
6.1.2 Experiment Results . . . 23
6.2 Speed Profile Coverage . . . 25
6.2.1 Experiment Settings . . . 25

6.2.2 Experiment Results . . . 25
6.3 Roads Hierarchy Coverage . . . 26
6.3.1 Experiment Settings . . . 26
6.3.2 Experiment Results . . . 27
6.4 Speed Profile for Brno . . . 28
6.4.1 Experiment Settings . . . 28
6.4.2 Experiment Results . . . 28
6.5 Speed Profile for Prague . . . 32
6.5.1 Experiment Settings . . . 32
6.5.2 Experiment Results . . . 32
6.5.3 Trace Mapping Example Using a Generated Trace . . . 36
6.6 Trace Mapping Using a Real-World Trace . . . 39

7 Conclusion **43**
7.1 Future Work . . . 43

8 User Guide **47**

9 DVD content **49**

List of Figures

3.1 Free space diagram for two curves. Source [1]. 7
3.2 Two-dimensional free space diagram. Source [1]. 7

4.1 Activity diagram for data processing. 12
4.2 An example of mapped trace with original waypoints. 17
4.3 An example of mapped trace with original and projected waypoints. 17

5.1 Workflow diagram. 22

6.1 Correctly mapped percentage of segments for distinct time intervals and degrees of noise. 24
6.2 Generated traces for percentage coverage of the graph of Brno. 29
6.3 Speed profiles for percentage coverage of the graph of Brno. 30
6.4 Usage rates for edges of the graph of Brno. 31
6.5 Details of original and mapped traces for Prague. 33
6.6 Speed profiles for the graph of Prague. 34
6.7 Detail of speed profiles for the graph of Prague. 35
6.8 Single trace for Brno. 37
6.9 Single mapped trace for the graph of Brno with linear interpolation of the waypoints. 38
6.10 Single mapped trace for the graph of Brno with projected waypoints. 38
6.11 The real-world trace. 40
6.12 Detail of the mapping of real-world trace. 41

List of Tables

6.1 Average of correctly mapped segments [%]. 25
6.2 Average time for mapping of one trace [s]. 25
6.3 Number of traces needed to cover n percents of all edges in the graph of Brno. 26
6.4 Coverage of road types in Brno with respect to length. 27
6.5 Coverage of road types in Prague with respect to length. 27

Chapter 1

Introduction

1.1 Motivation

Since the traffic level in streets of cities is high, algorithms for route planning can be very helpful not only to commuters. Using optimal routes can help companies to save money but it also can lead to reduction of emissions. Although it is possible to use a shortest path, it does not ensure that we will avoid waiting for a long time in traffic jams. Thus on many occasions we prefer to optimize the travel time instead of the distance. For these and other reasons, we want to consider the level of traffic while planning the routes.

Knowledge of average speed for roads can also be used to analyze the traffic flow in the road network and in considering changes in the infrastructure. Although real-time data might be very useful, our goal is to use data collected in longer term to predict duration estimates based on a speed profile.

Instead of building expensive infrastructure to measure the traffic volumes, we use the GPS traces from the fleets of cars owned by companies.

1.2 Aim of the Thesis

The aim of this thesis is to choose and implement existing map matching algorithm for GPS traces to propose a method for an instantiation of a model representing journey duration estimates and to evaluate the implemented solution. To achieve this, we did the following steps.

At first, we survey the existing methods of matching GPS traces onto a road network represented by directed graph and list the ones we consider relevant to our task.

Then, we describe the input and internal representation of data and formalize the model for the represantation of journey duration estimates.

The most important part of the thesis is the description of the the algorithm to complete the journey duration estimates for a city. We divide the task of using GPS traces for journey duration estimates in transport networks into two parts. First part of the problem is to determine the original routes from noisy GPS data by mapping each GPS trace onto some path in a given graph. The second part is to correctly interpret the collected data in terms

of computing journey duration estimates. Therefore, we describe implementation of an existing GPS traces matching algorithm that was used and propose a method to extract an instantiation of the model for the journey duration estimates. Once we have mapped a GPS trace onto some path in a given graph, we need to determine which points in the path correspond to the waypoints the GPS trace consists of. We use timestamps from GPS data to estimate average speeds for segments of the mapped path. Having computed average speeds for segments of each mapped trace, we create a representation of average speeds for edges of the graph. Since the level of traffic usually varies throughout the day, there are multiple values for each edge. Each value corresponds to average speed in different time interval of a day.

Finally, we evaluate the implemented solution using generated and real-world GPS traces data for graphs of road networks of Prague and Brno to show the precision of the mapping algorithm, the scalability of the algorithm, and the ability to reflect different traffic conditions.

1.3 Structure of the Thesis

In Chapter 2, we mention some of the articles concerning the mapping algorithms and other related problems. Chapter 3 specifies the input and output of the algorithm as well as theoretical concepts used to describe the algorithm. In Chapter 4, the used algorithms are described. Section 4.1 is about the mapping part of the algorithm while the Section 4.2 and Section 4.3 describe how the speed profile is created. Some further details about how the algorithm is implemented are mentioned in Chapter 5. In Chapter 6, there is a description of how the algorithm is evaluated. The work is summarized in Chapter 7 where we also mention some possible further improvements.

Chapter 2

Related Work

Many articles about mapping a trace onto a graph were published. Here we summarize the most important ones for our work.

In the article [3], two approaches for mapping GPS traces onto paths in a graph are described and compared. The article compares global map-matching algorithm to an iterative one in terms of mapping precision and the time needed to find the mapping. The global map-matching algorithm looks for a path in a graph with minimal Fréchet distance from the original trace. The iterative algorithm looks for mapping of the segments of the trace sequentially using a local look-ahead. The result of the comparison shows that the global map-matching algorithm is slower but more precise then the iterative one. Further details on the global map-matching algorithm we use concerning its description, computational complexity and some variations of the algorithm are well presented in the article [1].

The definition of Fréchet distance can be found in the article [2] together with the description of how to compute the Fréchet distance between two polygonal curves.

The article [4] presents another type of map-matching algorithm. This algorithm only maps the trace onto close edges in the graph using contextual information to increase the precision of the mapping. The results are then used to measure traffic flow and level of congestion for individual road segments. However, the algorithm does not ensure that the trace is mapped onto a path in the graph since the mapping does not have to be continuous. This fact renders the algorithm rather unuseful for the purpose of creating journey duration estimates since we do not only need to know which roads were used but we also need some estimate of the time needed to traverse it.

Using speed profiles with a combination of real-time data to determine optimal routing policies and optimal departure times under time-varying traffic flows is concerned in [5]. Combining speed profiles and real-time data is also mentioned in the article [7] as a way to obtain better cost functions assigning travel times to road segments. The reason for using this combination is that the speed profile reflects the long term traffic dynamics and the real-time data is used to take the actual traffic level into account or in some cases even to predict congestions.

Chapter 3

Problem Representation

3.1 Used Theory

The mapping part of the algorithm applied on a trace is looking for a closest path in the graph with respect to Fréchet distance. The Fréchet distance is defined in [1] as

Definition (Fréchet distance). Let $f : I = [l_I, r_I] \to \mathbb{R}^2, g : J = [l_J, r_J] \to \mathbb{R}^2$ be two planar curves and let $\|.\|$ denote the Euclidean norm. Then the *Fréchet distance* $\delta_F(f, g)$ is defined as

$$\delta_F(f,g) := \inf_{\substack{\alpha:[0,1]\to I \\ \beta:[0,1]\to J}} \max_{t\in[0,1]} \|f(\alpha(t)) - g(\beta(t))\|$$

where α and β range over continuous and non-decreasing reparametrizations with $\alpha(0) = l_I, \alpha(1) = r_I, \beta(0) = l_J, \beta(1) = r_J$.

The intuitive definition based on [1] and [8] is:

A dog and its master are going for a walk but they are moving on different trajectories. The Fréchet distance between the two trajectories (curves) is the minimum length of a leash required to connect a dog and its master if neither of them can retrace his steps but both can vary their speed or stop.

We use the definition of free space and free space diagram that is mentioned in [1].

Definition. Let $f : I \to \mathbb{R}^2, g : J \to \mathbb{R}^2$ be two curves; $I, J \subseteq \mathbb{R}$. The set $F_\epsilon(f,g) := (s,t) \in I \times J | \|f(s) - g(t)\| \leq \epsilon$ denotes the *free space* of f and g. We call the partition of $I \times J$ into regions belonging or not belonging to $F_\epsilon(f,g)$ the *free space diagram* $FD_\epsilon(f,g)$.

(See Figure 3.1.)

For the purpose of the algorithm description we shall introduce some simplified concepts similar to those used in [1].

- Let *two-dimensional free space diagram* be a free space diagram for one edge of the graph and linear interpolation of the trace. (See figure 3.2.)
- Let *one-dimensional free space diagram* be a free space diagram for one node of the graph and linear interpolation of the trace.
- Let *two-dimensional free space diagram cell* (*one-dimensional* respectively) be a free space diagram for edge (node) in the graph and a section of trace between its two consecutive waypoints.
- Let *white interval* be defined as an segment of a one-dimensional free space diagram cell that belongs to its free space. (It is obvious that every one-dimensional free space diagram is well described by an ordered list of its white intervals.)
- Let e be an edge from node n_0 to node n_1. Let w_0 (w_1) be white interval in the one-dimensional free space diagram of n_0 (n_1 respectively). Let there be a non-decreasing curve starting at w_0 and ending in w_1 such that every point of this curve belongs to the free space of the edge and trace. Then we call this curve *feasible path* in two-dimensional free space diagram of e.
- Let *free space surface* be union of two-dimensional diagrams for the entire graph, i.e., for each edge in the graph.
- Let *feasible path* in free space surface be such a path in the free space surface that its intersection with any two-dimensional free space diagram is either empty set or feasible path in that two-dimensional free space diagram.
- Let n_0, n_1 be two nodes in the graph. Let e be edge from n_0 to n_1. Let x_0 (x_1 respectively) be points in free space of n_0 (n_1) and the trace such that x_1 is reachable from x_0 by feasible path in two-dimensional free space diagram of e and the trace. For fixed x_0 the *left pointer* (*right pointer* respectively) points to the leftmost (rightmost) possible position of w_1. The *left pointer* (*right pointer*) for an interval is defined as *left pointer* (*right pointer*) where x_0 is the beginning of the interval.
- Let e be an edge from node f to node t in a given graph. Then we define *segment pointers* for given edge as a set of two ordered lists where the first list contains left pointer and the second contains right pointer for each white interval in one-dimensional free space diagram for f.
- Let e be an edge from node f to node t in a given graph. Let sp_e be segment pointers for e. Let F_f (F_t respectively) be a one-dimensional free space diagram for f (for t).Let x_0 be a point in F_f. Let R be set of all points in F_f between values of left and right pointer for given x_0. Then for given value of x_0 and given e we define *reachable intervals* of F_t as intersection of R and white intervals of F_t.
- Let $G < N, E >$ be a graph where N is set of nodes and E set of edges. Let $n \in N$ be some node. Let $N_p \subseteq N$ be a set of nodes in this graph such that for every $n_i \in N_p$ there exists some edge $e_i \in E$ from n_i to n. Let $RI_{x_0,i}$ be reachable intervals for the edge e_i and some given x_0. Then for given value of x_0 we define *consecutive chain* for n as $\bigcup_{e_i} RI_{x_0,i}$.

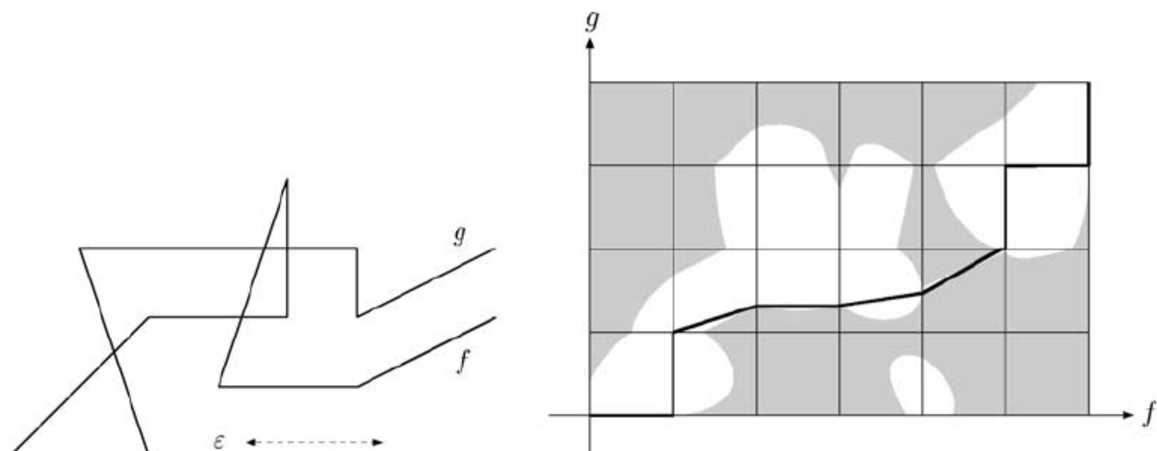


Figure 3.1: Free space diagram for two curves. Source [1].

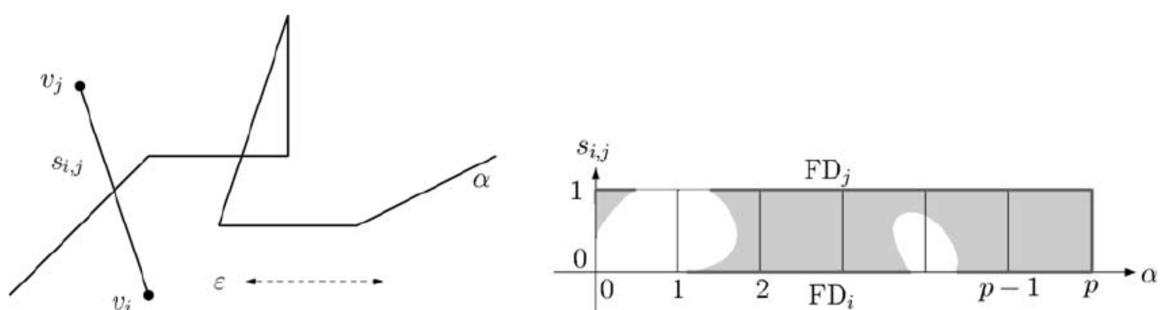


Figure 3.2: Two-dimensional free space diagram. Source [1].

3.2 Data Representation

In this section we formalize the model for the journey duration estimates, we describe how the data are internally represented throughout the mapping algorithm and what does the input of the algorithm look like.

3.2.1 Input

Input of the algorithm is a road network map represented as graph G and a set of GPS traces.

Let V be set of vertices and E set of edges. Let $w : E \to \mathbb{R}$ be function that assigns weight to each edge so that for $e \in E$ is $w(e)$ equal to length of e in meters. Let $g : V \to \mathbb{R}^2$ be function that assigns latitude and longitude to each $v \in V$. Then graph G can be represented as weighted directed graph $G = (V, E, w, g)$.

Each GPS trace is a sequence of n waypoints $W = (w_1, w_2, \ldots, w_n)$. Waypoint w_i is defined as ordered triple $w_i = (x, y, z)$ where x, y represent latitude and longitude of w_i respectively and z is a timestamp.

3.2.2 Inner Representation

In this section, we describe some of used classes that are mentioned during the description of the mapping algorithm.

- $OneDimensionalFreeSpaceDiagram$ – represents the one-dimensional free space diagram.
- $SegmentPointers$ – contains segment pointers for given two-dimensional free space diagram, i.e., this class contains left and right pointer for each cell in this free space diagram segment.
- $ConsecutiveChain$ – represents the consecutive chain respective to some node. The chain is described by its beginning and end point and can be easily reconstructed by intersection with one-dimensional free space diagram for given node.
- $QueueElement$ – represents a single element of priority queue. Contains ID of node and white interval (element of one-dimensional free space diagram respective to this node). Two instances are equal if node id is equal and natural ordering is done with respect to beginning of the white interval.
- $IntervalDescriptor$ – is used to identify white interval in one-dimensional free space diagram of some node. Contains id of the node and number of the cell to which the white interval belongs.
- $Trace$ – contains trace as ordered set of waypoints.
- $MappedTrace$ – contains $Trace$, and sequence of edges in the graph representing the estimated original path.
- $TimedMappedTrace$ – contains $MappedTrace$, list of timestamps for nodes (instances of DateTime) and list of durations for edges (differences between two consecutive timestamps).

3.2.3 Speed Profile

We define a *speed profile* in a following way:

Let k be length of one time interval of a day in minutes. Let $N = (24 * 60/k)$ be number of segments of day. Let E_0 be set of all edges in a given graph and $S = \{s_1, \ldots, s_N\}$ be set of all time intervals of a day. Then we can represent speed profile for edges as function p_0 such that

$$\forall e \in E_0, \forall s \in S, p_0 : (e, s) \to speed_{avg}$$

.

Such model can represent average speeds for any edge, but cannot reflect difference for distinctive turn lanes. That is why we use a slightly different representation of speed profile.

Let k be length of one segment of day in minutes. Let $N = (24 * 60/k)$ be number of segments of day. Let E be set of all tuples (e_1, e_2) where e_1 and e_2 are two consecutive edges in the graph and $S = \{s_1, \ldots, s_N\}$ be a set of all segments of day. Then we can represent speed profile for edges and junctions as function p such that

$$\forall (e_1, e_2) \in E, \forall s \in S, p : ((e_1, e_2), s) \rightarrow speed_{avg}$$

.

3.2.4 GPS Traces to Speed Profile Problem

Output of the entire algorithm is S_p instance of *speed profile* representing previously described function p with restricted domain of definition. The function p represented by S_p is only defined for $((e_1, e_2), s)$ if at least part of one of the traces was mapped onto the edges e_1, e_2 and the data was logged at time interval s of the day.

The *speed profile* S_p is internally implemented as $HashMap<<N1, N2, N3, s>, speed_{avg}>$ where $N1$ is id of node in which the first edge begins, $N3$ is id of the node the second edge ends in, $N2$ is id of the node that both edges have in common and s is the tima intervals of a day ($speed_{avg}$ is average speed for given consecutive edges and time interval of a day). This implementation can also be used to represent the function p_0 if we set $N3$ as constant.

Chapter 4

Solution Approach

In this Chapter, we describe the used mapping algorithm (see Section Mapping Traces Onto a Graph) and the proposed method for model instantiation (see Sections Data Extraction and Model Instantiation).

The task is divided into four parts:

1. *Mapping Traces Onto a Graph* – input: set of $Trace$; output: set of $MappedTrace$
2. *Data Extraction* – Here we add timings to all nodes in the mapped path input: set of $MappedTrace$; output: set of $TimedMappedTrace$
3. *Model Instantiation* – Here we create the representation of *speed profile* input: set of $TimedMappedTrace$; output: instance of speed profile
4. *Visualization* – used only for representation of the results

Each part is further described in its own subsection. An activity diagram showing how the data are processed can be seen in Figure 4.1

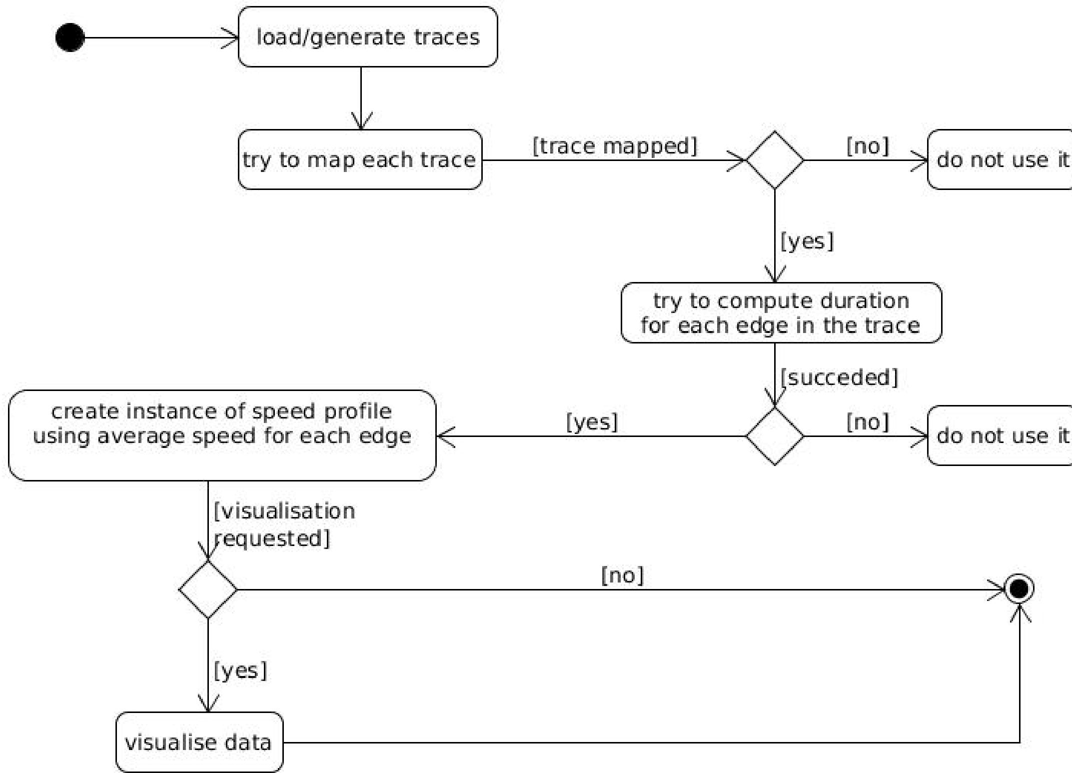


Figure 4.1: Activity diagram for data processing.

4.1 Mapping Traces Onto a Graph

This part of the algorithm estimates the original routes based on GPS data. It maps one trace at a time and is run for all given traces. Since our intention isn't to perform the mapping task in a real time, the algorithm we use is the slower but more precise one that is based on Fréchet distance and maps the trace as a whole.

The mapping has three main parts. In the first one (preprocessing) a representation of all the free space diagrams is computed and stored into variables. Thus we have the representation of the entire free space surface. The second part (dynamic programming) searches for feasible path in the computed free space surface from some point within epsilon from the first waypoint of the trace to some point within ϵ from the last waypoint of the trace. Existence of such path means there is a mapping of the trace onto the graph and all the points in the path are at most ϵ distant from linear interpolation of the trace. The third part reconstructs the path using the output of the second part. Using these three steps we run a binary search for minimal ϵ for which there exists a mapping.

The implemented algorithm is described in [1] including the mathematical theory used. The following description is rather description of the implementation than the mathematical definition of the algorithm and as such is mathematically less precise but should be more

'reader-friendly'.

4.1.1 Preprocessing

During the preprocessing, we create a representation of free space diagram for given graph and given trace.

The following definitions of symbols are valid until the end of this section.

- Let G be a graph of some road network.
- Let t be a trace we want to map onto some path in G.
- Let I be a set of IDs of all nodes in G.
- Let $i_n \in I$ be an ID of some node n in G.
- Let J be a set of IDs of all edges in G.
- Let $j_e \in J$ be an ID of some edge e in G.

For each node in the graph we count its one-dimensional free space diagram and store it in a $HashMap$. This $HashMap$ represents a function f_1 we are about to define.

Having O^1 set of one-dimensional free space diagrams for each node in G and the given trace t. Let $o_n^1 \in O^1$ be one-dimensional free space diagram for the node n and given trace t. Then function $f_1 : I \to O^1$ is defined as: $\forall i_n \in I : f_1(i_n) = o_n^1$.

At the same time we compute segment pointers for each two-dimensional free space diagram (i.e., for every edge compatible with given mode of transport) and we store the information in a different $HashMap$ representing function f_2.

Having S^{ptrs} set of segment pointers for each edge in G and the given trace t. Let $s_e^{ptr} \in S^{ptrs}$ be segment pointers for the edge e and given trace t. Then function $f_2 : J \to S^{ptrs}$ is defined as: $\forall j_e \in J : f_2(j_e) = s_e^{ptr}$.

Next thing done as a part of preprocessing is initialization of data structure for representation of consecutive chains. We store the data in $HashMap$ representing function f_3.

Having C^{chns} set of consecutive chains for each node in G and the given trace t. Let $c_n^{chn} \in C^{chns}$ be consecutive chain for the node n and given trace t. Then function $f_3 : I \to C^{chns}$ is defined as: $\forall i_n \in I : f_1(i_n) = c_n^{chn}$.

The initialization of the data structure is done as follows:

- if a node is within epsilon from first waypoint of the trace then we assign consecutive chain consisting of degenerate interval $[0, 0]$ to this node.
- else an empty consecutive chain is assigned to the node.

Finally a variable *prevInterval* of type $HashMap$ is created to represent function f_4.

Let E be set of all edges in G. Let n_0 and n_1 be two nodes in G such that there is an edge $e \in E$ from n_0 to n_1. Let id_0 (id_1) be an instance of *IntervalDescriptor* representing some white interval in one-dimensional free space diagram for node n_0 (n_1 respectively) in G and let there be a feasible path in the two-dimensional free space diagram of e from id_0 to id_1. Then $f_4(id_1) = id0$.

This $HashMap$ is used for path reconstruction.

4.1.2 Path Search

The second part of the mapping algorithm is looking for a feasible path in the free space surface with use of dynamic programming, sweeping and free space diagram representation created in preprocessing part.

The sweeping is done by line that is shifted from beginning of the trace to its end so we sweep the entire free space diagram by one sweep line and we represent it only by its position stored in variable x initialized as $x = 0$.

We create a priority queue q of *QueueElement* with natural ordering. The initial elements in q are *QueueElement*s with nodes that are within epsilon from first waypoint in the trace to be mapped.

We update the q until we find the desired feasible path or the queue becomes empty. The update of the q is done by following cycle:

1. Remove head of the queue, i.e., the actual leftmost interval in the q and set the value of x as beginning of its white interval (that corresponds to shifting the sweep line to beginning of the white interval).

2. Find next white interval (if there is some) in consecutive chain for the node specified in removed *QueueElement* and insert it into q.

3. For the node specified in removed *QueueElement* find all outcoming edges and their end nodes. For each such end node, update its $ConsecutiveChain$ considering the new value of x. If the beginning of the $ConsecutiveChain$ has changed then update the *QueueElement* for this end node.

4. If white interval (element of *ConsecutiveChain*) with *IntervalDescriptor* $id1$ can be reached for the first time and is reachable by monotone feasible path from white interval with $IntervalDescriptor$ $id0$ then store $id1$ as key and $id0$ as value into map $prevInterval$

5. If the q is not empty, and value of x is smaller then the number of cells in each segment (a constant) it means we haven't found solution yet. The fact that q is not empty means we can continue in search and thus we start another iteration of this cycle.

4.1.3 Path Reconstruction

The third and last part of the mapping algorithm reconstructs the estimated path from map $prevInterval$.

If the cycle in previous part of the mapping algorithm ended without finding a feasible monotone path then no such path exists in the free space diagram for given epsilon.

Otherwise, we reconstruct the path. Using the fact that for each visited white interval we have stored $IntervalDescriptor$ of preceding white interval it is easy to reconstruct the path by simply reversing the order of nodes and finding edges between each two consecutive nodes.

4.1.4 Mapping Algorithm Improvement

Let G be the graph we are trying to map the traces onto. In most cases the mapping algorithm does not have to use the entire graph G to find the mapping. This means that many operations are done in vain. To avoid this situation or at least lower the number of needless operations a simple trick can be used.

It is obvious that the mapping of the trace can consist only of such edges that the distance between the edge and the linear interpolation of given waypoints is at most ϵ. Thus in most cases we can use only part of the graph.

To do so we first find the minimal and maximal values of latitude and longitude of the waypoints (this can be done in linear time with respect to number of the waypoints). Let lat_{min}, lon_{min} be the minimal latitude and longitude (respectively) and lat_{max}, lon_{max} be the maximal latitude and longitude. Let c be a constant corresponding to maximal allowed value of ϵ. Then by adding c to lat_{max}, lon_{max} and by subtracting c from lat_{min}, lon_{min} we ensure that no possible mapping is lost. Now instead of using the graph G we use its subgraph G_1 that is induced by V where V is set of vertices of G such that $\forall v \in V$ are in the area bounded by $lat_{min}, lon_{min}, lat_{max}, lon_{max}$.

This improvement does not have to help if the area bounded by $lat_{min}, lon_{min}, lat_{max}, lon_{max}$ contains the entire graph G but usually reduces the runtime significantly.

The idea of using only part of the entire graph comes from [3], where the authors use a concept of error ellipse to determine the needed part of the graph.

4.2 Data Extraction

So far we have described the algorithm that maps a GPS trace onto the nearest path in a graph (with respect to Fréchet distance). Now we have to assign proper timings to the nodes in every path. This means we need to project the original waypoints onto the path (projection) and then compute timestamp for every node in the path (interpolation).

4.2.1 Projection

Here we describe the projection used. Some other ideas how to project the original waypoints onto the mapped path are mentioned in Section $FutureWork$.

At first we define symbols used in the definition of the projection and the following observation.

- Let $\{w_1, \ldots, w_n\}$ be the waypoints for some GPS trace T.
- Let P be path in given graph such that P is mapping of trace T onto the graph.
- Let $\{n_1, \ldots, n_m\}$ be the nodes in path P.
- Let $\{v_1, \ldots, v_n\}$ be the projections of $\{w_1, \ldots, w_n\}$ onto the P respectively.
- Let $l_w = \sum_{i \in [1, n-1]} (\|w_i - w_{i+1}\|)$ be the sum of distances between each two consecutive waypoints in the trace T.
- Let $l_n = \sum_{j \in [1, m-1]} (\|n_i - n_{i+1}\|)$ be the sum of distances between each two consecutive nodes in the path P.

For better understanding see an example of mapped trace with original waypoints. (Figure 4.2)

Using these symbols we can define the used projection as follows:
Definition (Used projection).

1. $v_1 := n_1$, i.e., We project the first waypoint in T onto the first node in P.
2. $v_n := n_m$, i.e., We project the last waypoint in T onto the last node in P.
3. $\forall i \in [2, n-1] : v_i \in P \wedge \frac{\|v_{i-1} - v_i\|}{l_n} = \frac{\|w_{i-1} - w_i\|}{l_w}$

For better understanding of how the projection is done see an example of mapped trace with original and projected waypoints. (Figure 4.3)

Since we can use only such data that $n, m \geq 2$ (at least two nodes in P and at least two waypoints in T) we can make a simple observation about this projection.

Observation: 1.

$\forall n_i, i \in [1, m], \exists j \in [1, n] : n_i = v_j \oplus ((\|v_j - n_1\| < \|n_i - n_1\|) \wedge (\|n_m - v_{j+1}\| < \|n_m - n_i\|))$

(i.e., For the used projection holds that any node n_i either is identical with projection of some waypoint w_j or n_i is between projections of two consecutive waypoints w_j, w_{j+1}.)

We can easily proof that this observation is correct.

Proof. (using mathematical induction)

1. Base case: For n (number of waypoints) $n = 2$ we project the first waypoint of T onto the first node in P and the second waypoint onto the last node in P. The first node is thus identical with the projection of first waypoint and the last node is identical with projection of second waypoint. All other nodes are between projections of the first and the second waypoint.

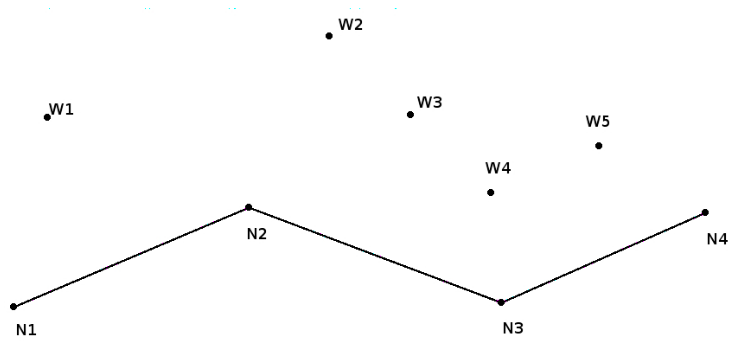


Figure 4.2: An example of mapped trace with original waypoints.

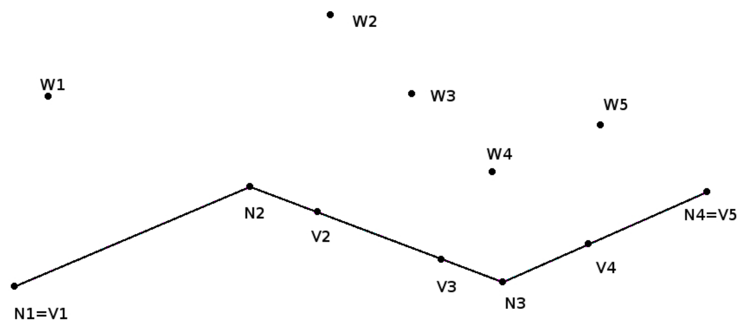


Figure 4.3: An example of mapped trace with original and projected waypoints.

2. Inductive step: Let us suppose that the observation is correct for $n = k; k \in \mathbb{N}$ then we need to prove that it is also correct for $n = k + 1$.

   Let $n = k + 1; k \in \mathbb{N}$ be number of waypoints. Using the definition of the projection we project all waypoints except the pre-last waypoint $w_{n-1} = w_k$ of T onto the P. According the induction hypothesis each node is now identical with a projection of some node, or lies between projections of two consecutive nodes. Let us call the segment of path P between v_{n-2}, v_n projections of waypoints w_{n-2}, w_n respectively segment S. The definition of the projection ensures that the projection v_{n-1} of the pre-last waypoint w_{n-1} is inside of this segment. It is obvious that status of all nodes outside of S remains unchanged if we project waypoint w_{n-1} onto P. Let $S_1 \subseteq S$ be the segment of P between v_{n-2} and v_{n-1}. Let $S_2 \subseteq S$ be the segment of P between v_{n-1} and v_n. This means that $S = S_1 \bigcup \{v_{n-1}\} \bigcup S_2$ and thus $\forall n_i \in S$ holds that n_i is either identical with v_{n-1} or $n_i \in S_1$ or $n_i \in S2$.

$\square$

4.2.2 Interpolation

Let T be some GPS trace mapped onto path P in given graph. This section describes how to assign timestamps to all nodes in P using waypoints in T.

Let us suppose there is some function F such that $\forall p \in P : F(p) = t_p$ where t_p is timestamp assigned to point p in path P. If such function exists we can use it to assign timestamps to all nodes in P.

In the section *Projection* we have described how to project all the waypoints in T onto P and furthermore we have made *Observation* 1. This observation ensures that any node in P is either identical with a projection of some waypoint in T, or is between projections of two consecutive waypoints in T. It means that all nodes in P are between known projections and thus we can create function F using some form of interpolation, i.e., we don't need to extrapolate the data.

The algorithm uses simple linear interpolation. Important and obvious fact is that once we have assigned timestamps to each node in the path P we can also easily obtain duration for any edge in P.

4.3 Model Instantiation

Using previously described parts of the algorithm we map each given GPS trace T onto some path P in given graph and we compute duration for every edge in P. This allows us to create instance of speed profile in the following way.

We create a model representing function p_0 (as described in section *DataRepresentation*) for every edge e in given graph by either of the two options:

- if e is part of mapping of at least one trace then we compute and save average speed for this edge (using average duration and length of e).
- else we do not save any data for this edge since we have none.

If we want to create a model representing p (as described in section $DataRepresentation$) then we apply the same options on every two consecutive edges in the given graph (i.e. on every tuple (e_1, e_2) where e_1 and e_2 are two consecutive edges in the graph).

The model returns null value for any key we have no data for since the internal representation of the model is a $HashMap$.

Chapter 5

Implementation

In this chapter, we further describe the implementation of the algorithm. The algorithm is implemented in JAVA 7 Programming Language and uses Maven dependency management tool[1]. The crucial parts of the algorithm are tested using JUnit testing framework.

5.1 Architecture

The main algorithm is composed of the following parts:

1. Algorithm for generating traces
2. Algorithm for mapping traces
3. Algorithm for projecting original waypoints and computation of timings
4. Algorithm for creating an instance of the speed profile

Each of the four algorithms can be run separately as a command-line tool. All four algorithms are implemented as *Callables* and are located in separate packages. This should allow to modify, or change, each part without necessarily changing all parts. Furthermore, it allows to run more then one thread if sufficient memory and computational power is available to the user. It also enables the user to run only one of the algorithm parts using already computed data.

A method for loading real-world data stored in GPX format was implemented using LGPL GPXParser Java library[2] created by AlternativeVision. This method is a part of *TraceLoader* class that can be used instead of algorithm for generating traces.

Figure 5.1 shows how the the four parts of the system are combined together.

Visualisation of the results is done using GeoTools[3] and graph-provider - a component that provides transport planning graphs for various cities (e,g., Prague, Brno) developed in Agents Technology Center[4].

[1] https://maven.apache.org/index.html
[2] http://gpxparser.alternativevision.ro/pages/index.html
[3] http://www.geotools.org/
[4] https://agents.felk.cvut.cz/

5.2 Testing

For the mapping and pathtiming algorithms as well as for the algorithm creating instances of the speed profile, JUnit tests are implemented to test their results. JUnit test are also implemented for crucial parts of the mapping algorithm, i.e., for creating an instance of a class representing two-dimensional free space diagram cell and for methods computing the left and right pointers.

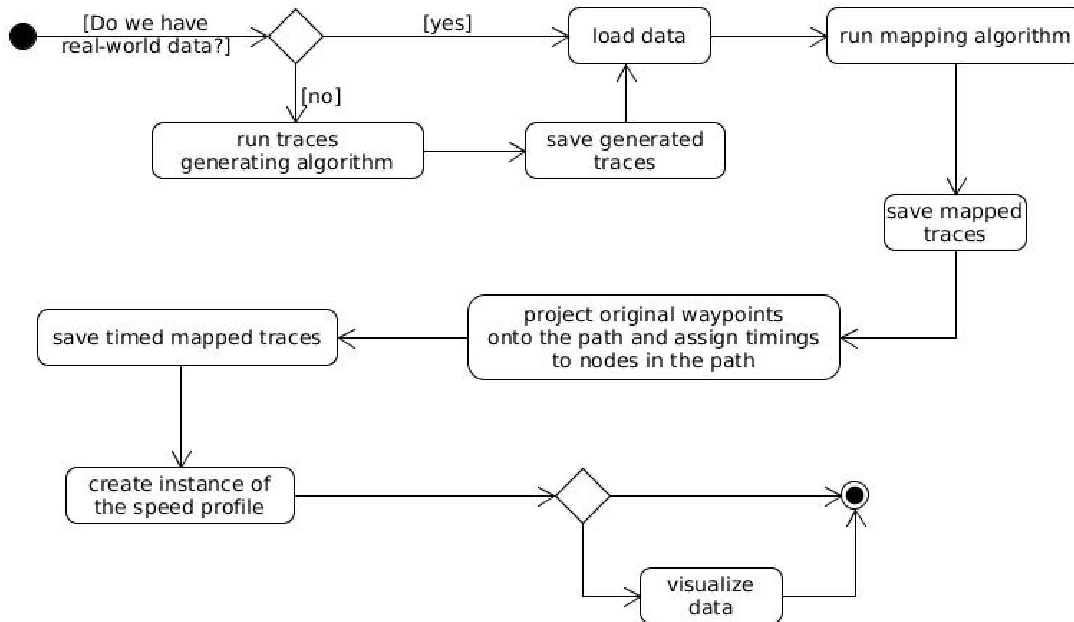


Figure 5.1: Workflow diagram.

Chapter 6

Evaluation

This chapter is dedicated to a description of how the algorithm has been tested and to a presentation of results of the experiments and visualization of the output of the algorithm.

The evaluation of the algorithm was done using the graphs of road networks of cities Prague and Brno. The graph of Brno contains 63029 nodes and 194004 edges out of which 40732 edges are feasible for cars. The graph of Prague is larger and contains 124501 nodes and 407655 edges. The number of edges in the graph of Prague that are feasible for cars is 139405. In the following sections we present these experiments: Mapping Precision, Speed Profile Coverage, Roads Hierarchy Coverage, Speed Profile for Brno, Speed Profile for Prague, Trace Mapping Example Using a Generated Trace and Trace Mapping Using a Real-World Trace.

6.1 Mapping Precision

6.1.1 Experiment Settings

This experiment tests the performance of the mapping algorithm with respect to precision of the mapping. The test is done with multiple sets of randomly generated traces with a distinct degree of noise and a time interval between two consecutive waypoints. For each trace, a percentage of correctly mapped segments is computed as well as time consumed for mapping the trace. The test is done on the graph of Brno.

Used values of degree of noise are: $1, 5, 10, 20, 30$ [m].
Used values of time intervals are: $15, 30, 45, 60$ [s].
This means 20 different sets of traces were used.
Maximum number of segments per trace is not defined.
Number of traces in each set is: 100.

6.1.2 Experiment Results

Axis representing degree of noise ranges from 1 to 30. If the value is n then after adding the noise still 99 percents of the waypoints should be n or less meters from their original

position. Axis representing time intervals ranges from 5 to 60. This value represent time in seconds between two consecutive waypoints. Axis representing average percentage ranges from 0 to 100.

Table 6.1 represents averages of correctly mapped segments for given degrees of noise and time intervals. The data are also shown in a graph (Figure 6.1). The Table 6.2 shows an average time in seconds needed for mapping of one trace under different settings.

From the results we can see that the time interval influences the mapping more than the noise degree. It also might be a key factor in time-precision trade-off. By simply ignoring some waypoints we could map the trace much faster but with lower precision.

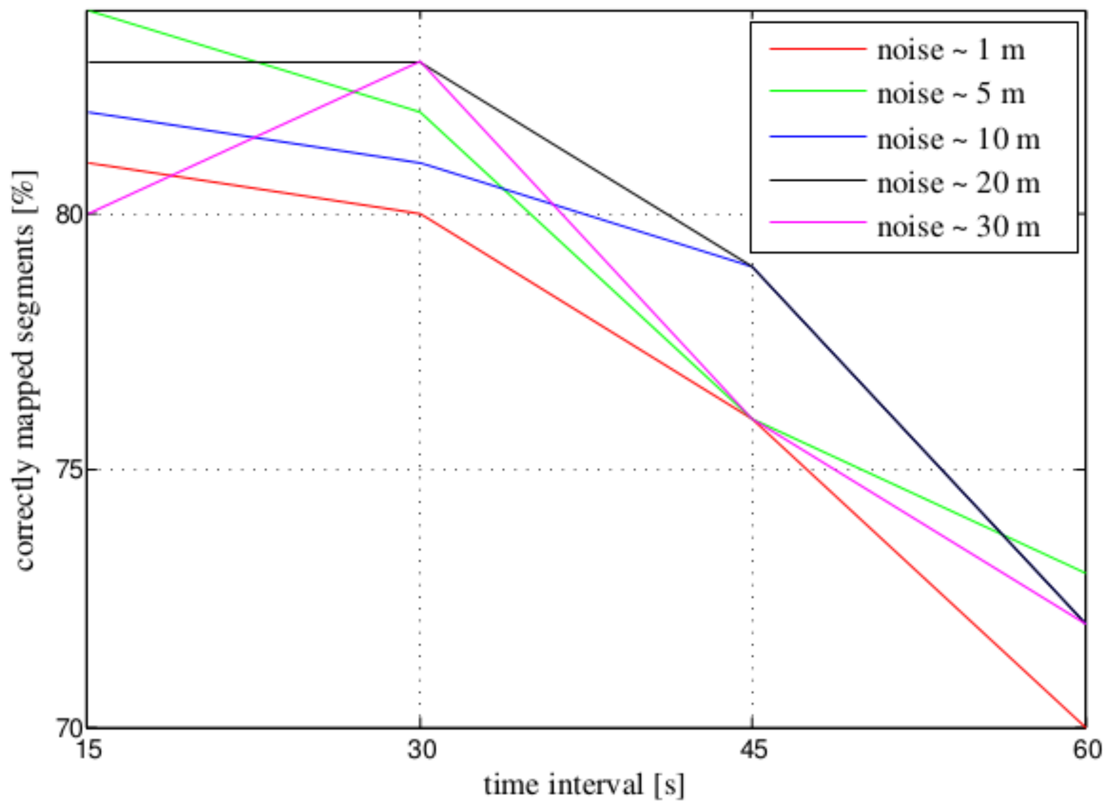


Figure 6.1: Correctly mapped percentage of segments for distinct time intervals and degrees of noise.

Average of correctly mapped segments [%]				
noise [m]:	15s interval	30s interval	45s interval	60s interval
1	81	80	76	70
5	84	82	76	73
10	82	81	79	72
20	83	83	79	72
30	80	83	76	72

Table 6.1: Average of correctly mapped segments [%].

Average time for mapping of one trace [s]				
noise [m]:	15s interval	30s interval	45s interval	60s interval
1	9	8	6	4
5	8	8	5	4
10	10	8	5	4
20	12	8	5	4
30	13	8	5	2

Table 6.2: Average time for mapping of one trace [s].

6.2 Speed Profile Coverage

6.2.1 Experiment Settings

Purpose of this experiment is to get some estimate of how many traces are needed in order to cover 40%, 60% and 80% of the graph of Brno with an instance of speed profile.

The experiment is done in the following way: We generate 3 different sets containing the same number of traces and compute an average covered percentage of the graph for these three sets. The covered percentage is computed as a ratio between the number of edges in the graph that are feasible for cars and the number of edges that are used by at least one trace. The size of generated sets is increased until we find a number that covers 80% of the graph or more at average.

This gives us an idea about how many traces must be generated for each time interval in a day in order to cover desired percentage of the graph (i.e., if m traces cover $n\%$ of the graph on average then we need to have m traces for each time interval in the day we want to include into the speed profile covering $n\%$ of the graph).

6.2.2 Experiment Results

Results of this experiment (see Table 6.3) are used to generate sufficient amount of traces in order to compute speed profiles covering given percentages of graph. Each row of the table contains these three numbers: number of traces per one set of generated traces, how many

percents of the graph we expect to cover by this number of traces and how many percents of the graph are covered on average.

We can see that the number of traces needed to cover some percentage of the graph does not grow linearly. This is mainly caused by the fact that the generator prefers the roads with higher speed limits. This fact is obvious from results of experiment *Roads Hierarchy Coverage* and can also be seen in Figure 6.4.

The number of traces needed to cover 80% is approximately 27000. This is really large amount of traces. Reasonable compromise between the number of traces and covered percentage seems to be somewhere around 3000-5000 traces that are sufficient to cover more than a half of the graph of Brno.

We should keep in mind that the required numbers of traces are so high because we use randomly generated traces. This should correspond to number of traces we would need if we used real-world data collected from 'randomly' moving cars such as taxis. It is obvious that by using some specialized cars to collect the data purposely we could significantly reduce these numbers.

Number of traces needed to cover n percents of the graph		
number of randomly generated traces	expected covered percentage of edges	average covered percentage of edges
1000	40	37
2000	40	46
3000	60	52
4000	60	56
5000	60	59
6000	60	62
17000	80	76
28000	80	82

Table 6.3: Number of traces needed to cover n percents of all edges in the graph of Brno.

6.3 Roads Hierarchy Coverage

6.3.1 Experiment Settings

In the previous experiment, we have shown approximate numbers of traces needed to cover specified percentages of the graph with respect to number of edges in the graph. In this experiment, we try to show how a fix number of traces covers the graph with respect to hierarchy of roads.

The experiment was done for graphs of Brno and Prague using 5000 randomly generated traces for each graph.

6.3.2 Experiment Results

For both graphs we have computed how many edges there are for distinctive types of roads and what is the sum of their lengths. Then we have computed the sum of lengths of the edges that are covered by at least one of the 5000 generated traces.

The results for Brno are shown in Table 6.4 and results for Prague in Table 6.5. We can see that more than 95% of motorways and primary roads are covered. We can also observe that coverage of the motorways, primary, secondary and tertiary roads is very good for both graphs regardless of the fact that the graph of Prague is notably larger. The worst covered road type in both graphs are living streets. Approximately 40% of living streets in Brno and less then 26% of living streets in Prague are covered but since the living streets are usually of little importance while planning a route we do not have to be concerned about the lack of information for this type of roads.

If roads with higher speed limits are preferred to the slower ones in real-world data too, then the speed profile will be always more accurate and more reliable for the roads that are higher in the hierarchy. This means that speed profile for either of the graphs created from 5000 traces should give us some information for most of the important roads.

Coverage of Brno			
type of the road	number of edges	length of all edges for a given type of road [km]	length of covered edges for a given type of road [km]
motorway	988	159.1	155.9
primary	837	55.4	53.6
secondary	1935	252.4	224.9
tertiary	5631	572.6	530.2
living street	809	57.0	23.9

Table 6.4: Coverage of road types in Brno with respect to length.

Coverage of Prague			
type of the road	number of edges	length of all edges for a given type of road [km]	length of covered edges for a given type of road [km]
motorway	1810	409.2	395.7
primary	2644	218.6	209.8
secondary	7988	692.1	632.2
tertiary	16699	1525.9	1227.3
living street	2978	221.1	56.5

Table 6.5: Coverage of road types in Prague with respect to length.

6.4 Speed Profile for Brno

6.4.1 Experiment Settings

In this section we will present two speed profiles for the graph of Brno covering 40% and 80% of edges in the graph for time interval between 0:00 a.m. and 1:00 a.m. To achieve this the results of *Speed Profile Coverage* experiment are used. The first speed profile covering approximately 40% of the graph is computed using 2000 randomly generated traces while the second one is based on 27000 randomly generated traces.

6.4.2 Experiment Results

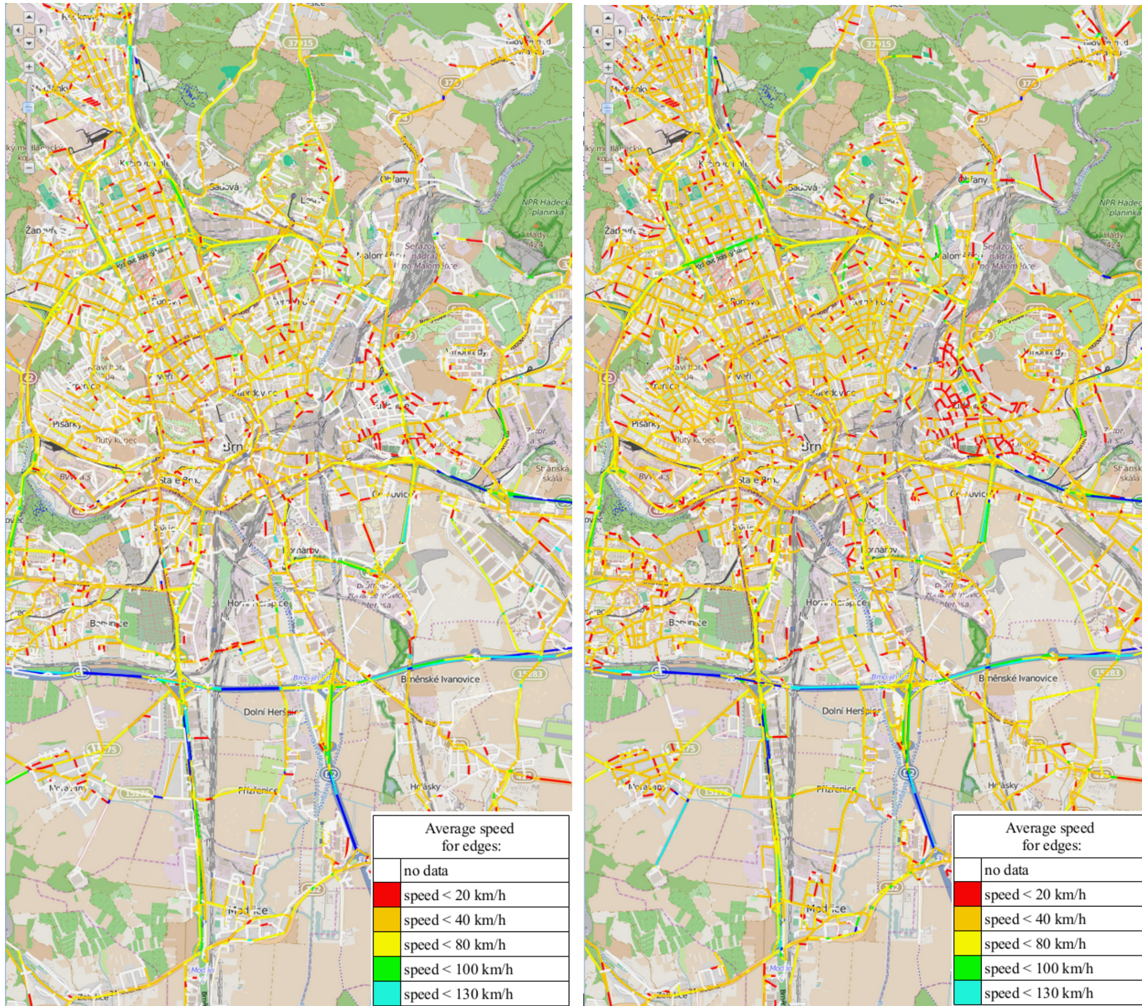
The Figure 6.2a and Figure 6.2b are visualizations of the generated traces for the two speed profiles respectively. By comparing these two pictures we can see that the coverage of living streets is notably higher for the bigger set of used traces. Thus it is obvious that by using more traces we will have speed profile for more edges in the graph.

The visualizations in Figure 6.4a and Figure 6.4b show by how many traces is each edge used. If we compare these two pictures we can observe that also the reliability of speed profile is higher if we use more traces as an input for the algorithm since the edge rates for the important roads are higher. The speed profiles are shown in Figure 6.3a and Figure 6.3b.

(a) 2000 randomly generated traces.

(b) 27000 randomly generated traces.

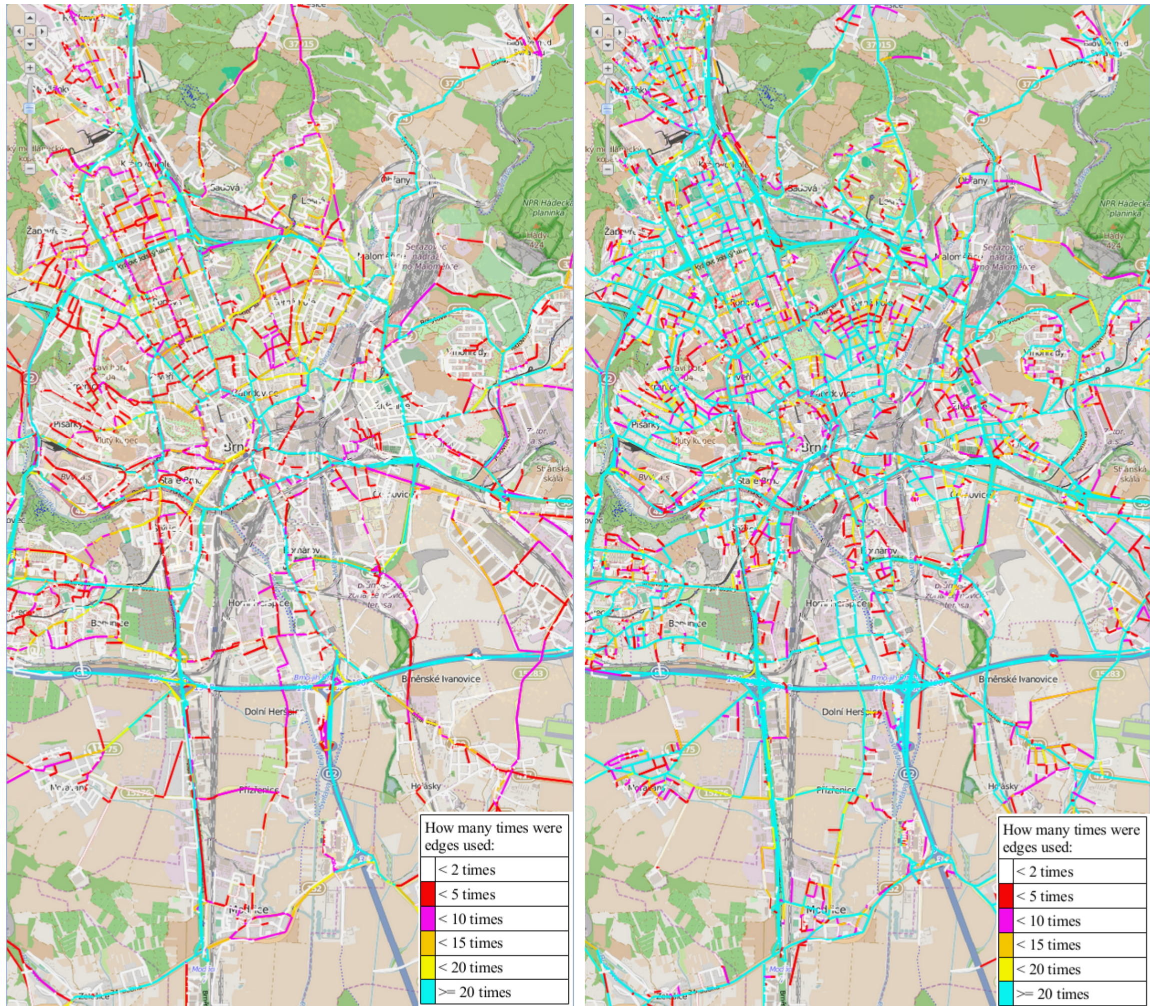
Figure 6.2: Generated traces for percentage coverage of the graph of Brno.



(a) Based on 2000 traces.

(b) Based on 27000 traces.

Figure 6.3: Speed profiles for percentage coverage of the graph of Brno.



(a) Rates for 2000 traces.

(b) Rates for 27000 traces.

Figure 6.4: Usage rates for edges of the graph of Brno.

6.5 Speed Profile for Prague

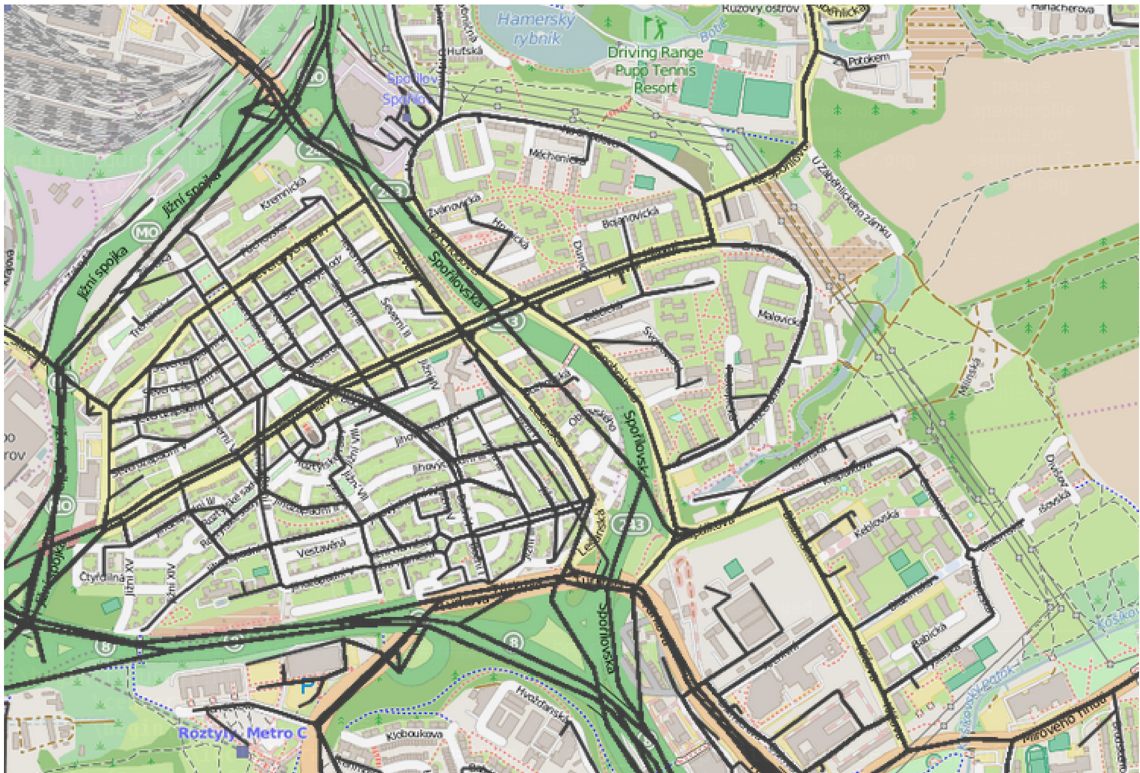
6.5.1 Experiment Settings

To show the scalability of our approach we dedicate this section to evaluation for the speed profile for the graph of Prague. The speed profile is defined for two time intervals in a day and is computed using mappings of 10000 randomly generated traces for each of the two time intervals. The first time interval lasts from 5:00 p.m. to 6:00 p.m. and since this is the time when many people are commuting from their work we will call it peak time interval. The second time interval lasts from 10:00 p.m. to 11:00 p.m. and since this is a time when the level of traffic is usually lower we will call it an off-peak time interval.

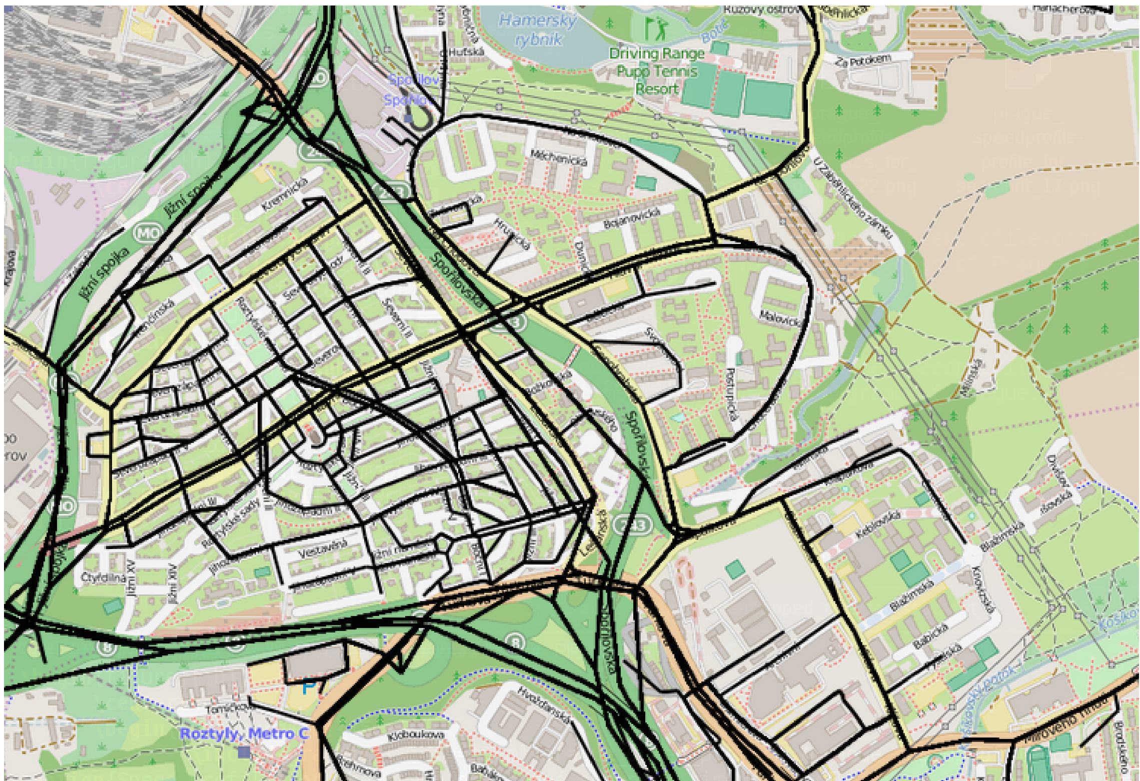
6.5.2 Experiment Results

In Figure 6.5a, you can see detail of the original generated traces for the peak time interval. Figure 6.5b shows a detail of visualization of the mapped traces for the peak time interval. The Figure 6.6a shows the visualization of speed profile for the peak time interval and Figure 6.6b shows the visualization of speed profile for the off-peak time interval.

Since the generator of the traces uses some representation of a speed profile (not a one computed by this algorithm), the average speeds during the peek and off-peak time intervals are different for many roads. We can easily observe by comparing the visualizations that these differences are reflected by the speed profiles created with this algorithm.

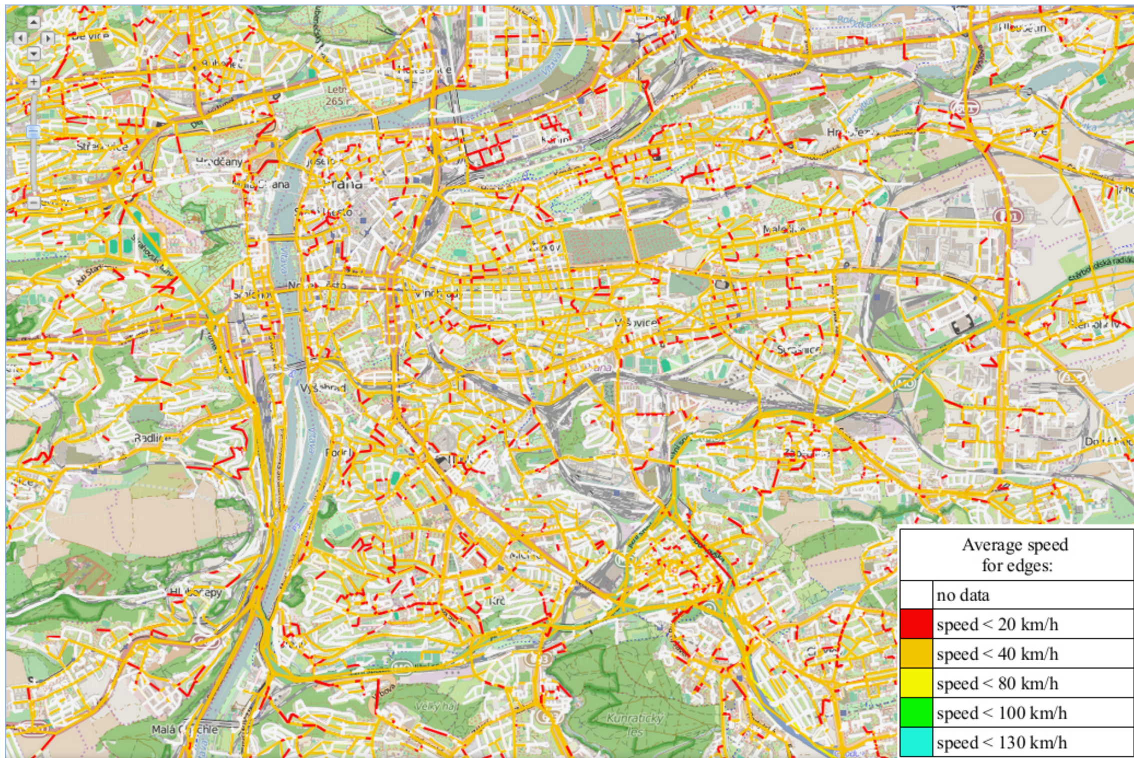


(a) Generated traces for peak time interval and the graph of Prague.

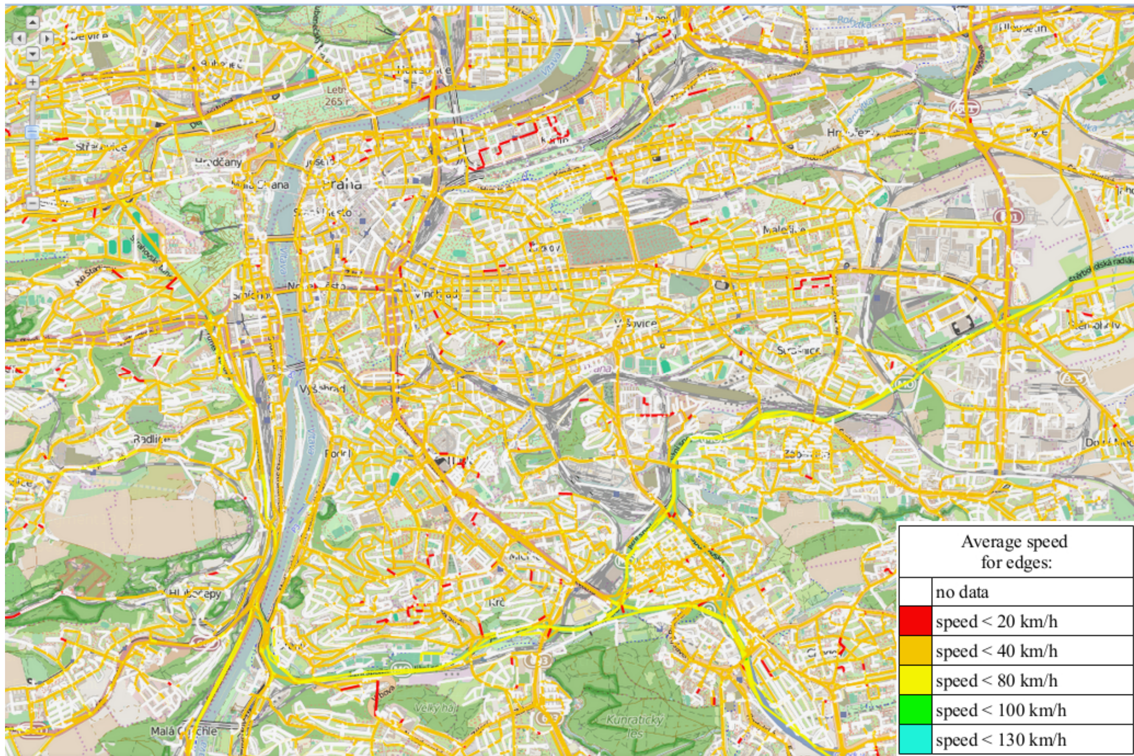


(b) Mapped traces for peak time interval and the graph of Prague.

Figure 6.5: Details of original and mapped traces for Prague.

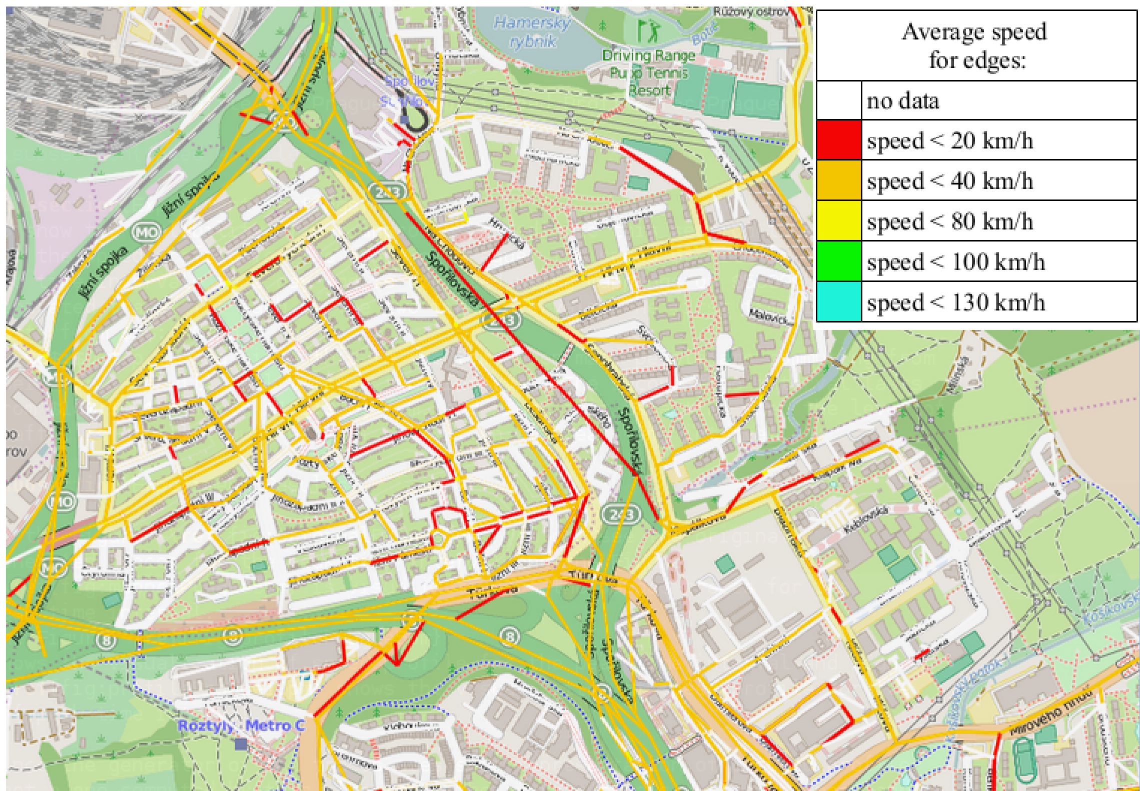


(a) Speed profile for the graph of Prague and peak time interval.

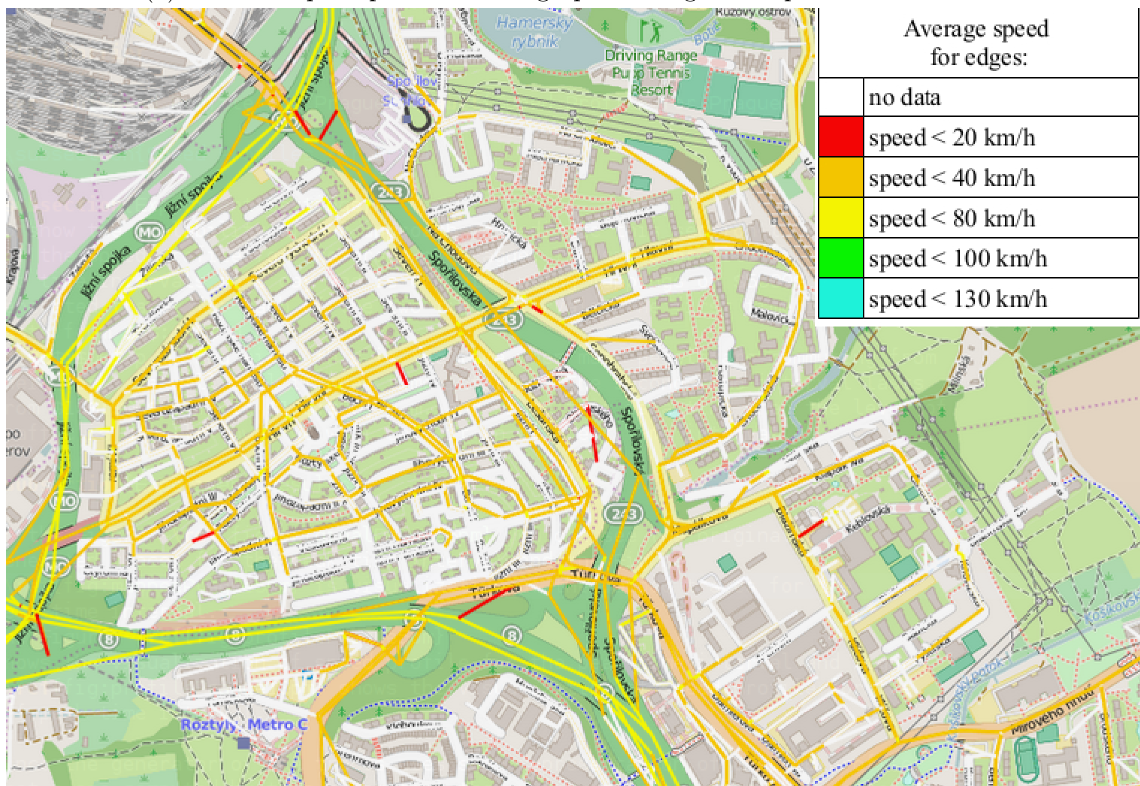


(b) Speed profile for the graph of Prague and off-peak time interval.

Figure 6.6: Speed profiles for the graph of Prague.



(a) Detail of speed profile for the graph of Prague and peak time interval.



(b) Detail of speed profile for the graph of Prague and off-peak time interval.

Figure 6.7: Detail of speed profiles for the graph of Prague.

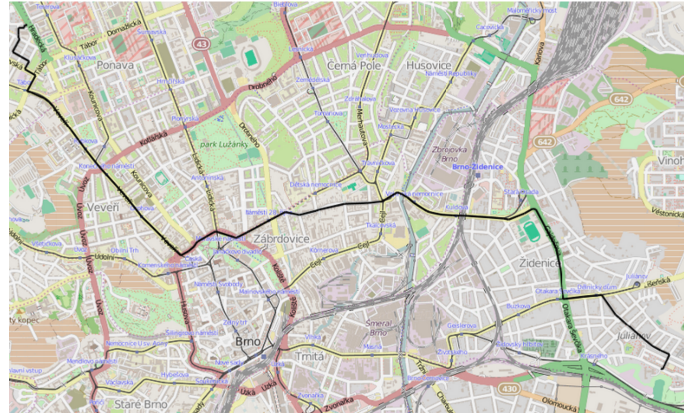
6.5.3 Trace Mapping Example Using a Generated Trace

This section shows the process of mapping of one trace onto the graph of Brno together with projecting the waypoints onto the mapped path in the graph.

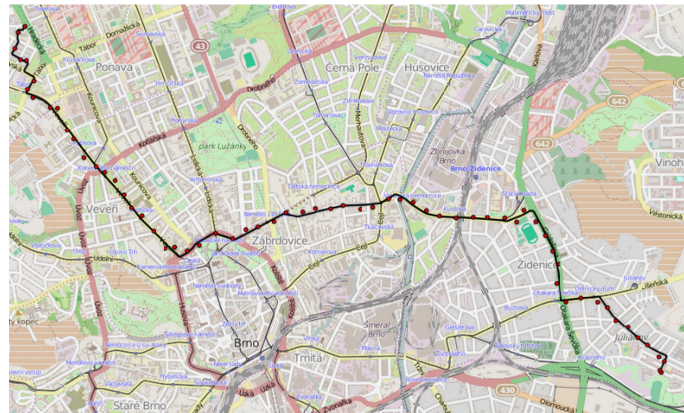
The Figure 6.8a is a visualization of single randomly generated trace for the graph of Brno while the Figure 6.8b shows the original trace with extracted noisy waypoints. The Figure 6.8c shows the waypoints and linear interpolation of the waypoints. In Figure 6.9 is visualization of the interpolated waypoints and the mapping that was found by the mapping algorithm. Projected waypoints are shown in Figure 6.10 together with interpolation of the waypoints.

Time between two consecutive extracted waypoints is 5 seconds which is relatively high frequency that increases the computational cost. The noise added to the waypoints location should keep 99% of all extracted waypoints at most 60 meters away from their original location. This level of noise is unusually high but the mapping is still done almost perfectly.

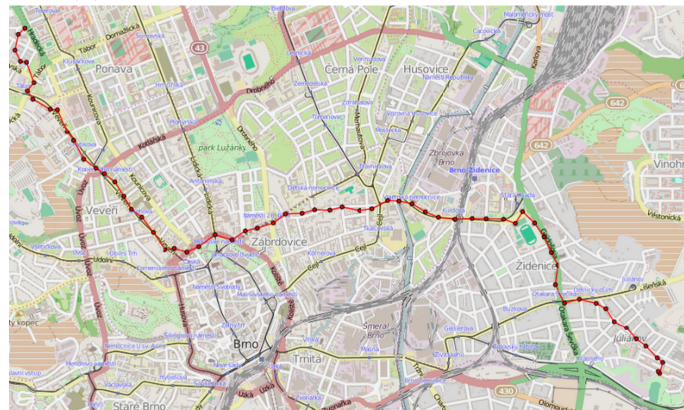
The purpose of this example is to demonstrate how the mapping of one trace is done and that even mapping of very noisy data can be successful.



(a) The original trace.



(b) The original trace with extracted waypoints.



(c) Linear interpolation of the waypoints.

Figure 6.8: Single trace for Brno.

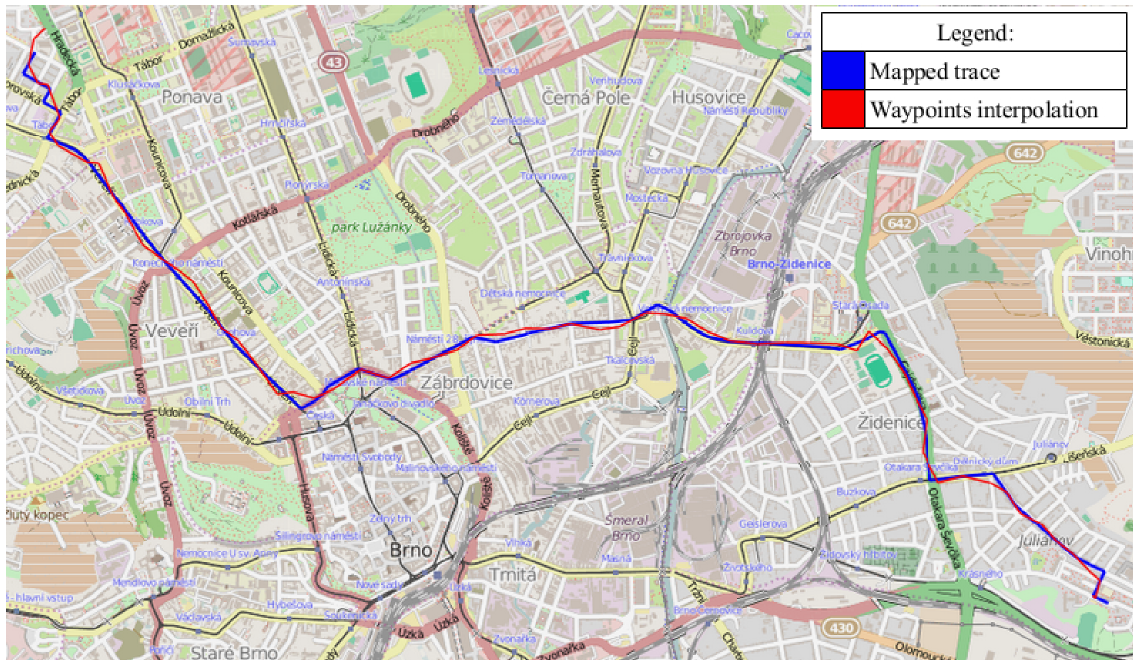


Figure 6.9: Single mapped trace for the graph of Brno with linear interpolation of the waypoints.

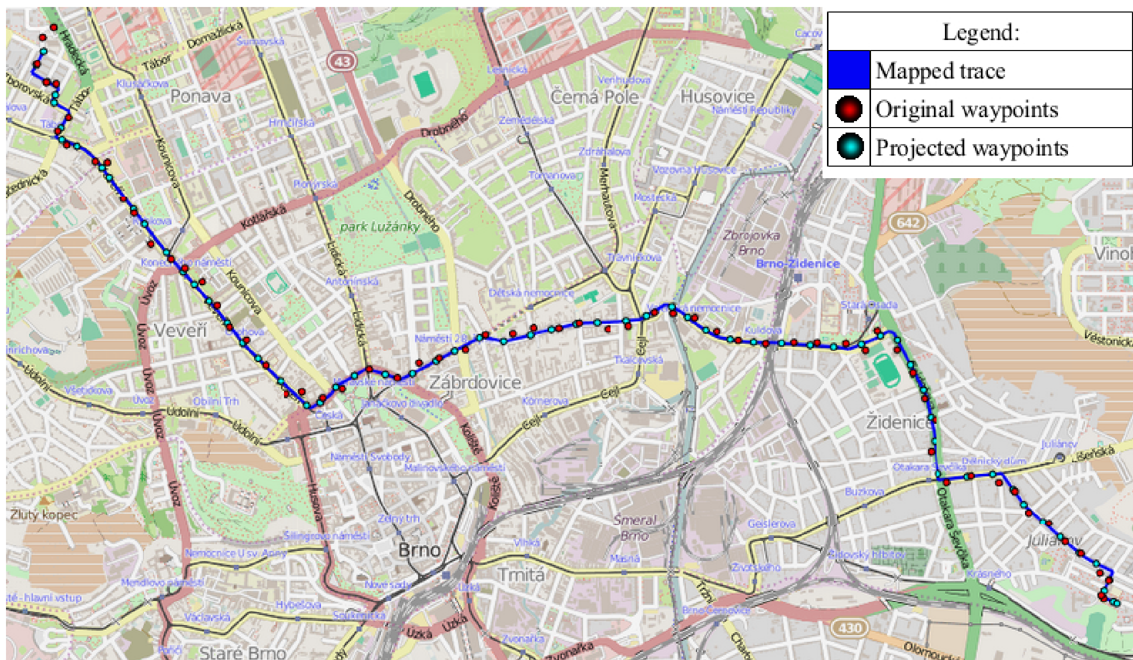


Figure 6.10: Single mapped trace for the graph of Brno with projected waypoints.

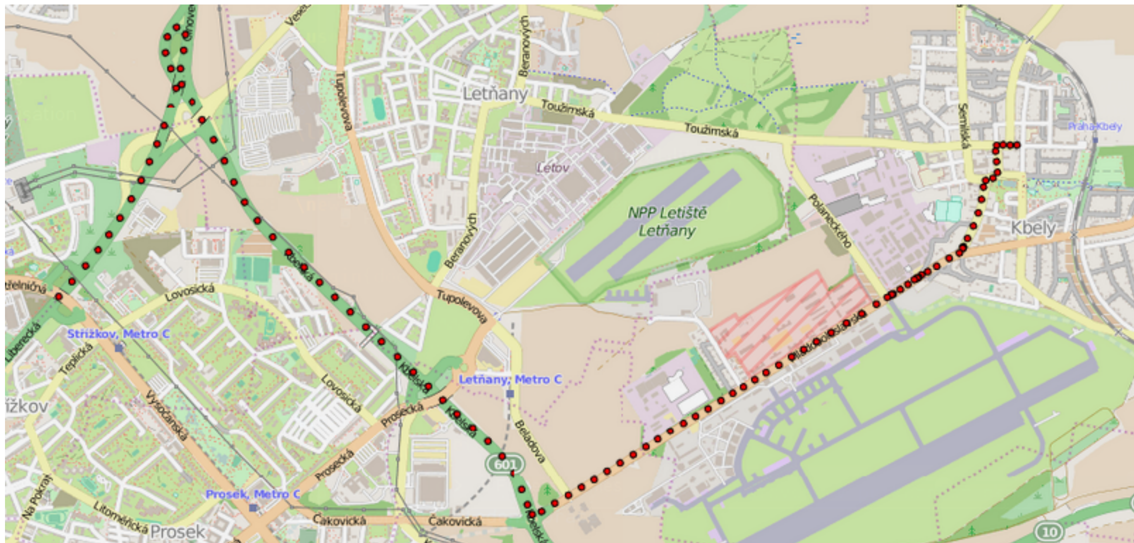
6.6 Trace Mapping Using a Real-World Trace

Finally we present results of the mapping algorithm applied on real-world data. The trace was logged on April 27, 2015. It is a trace for the city of Prague and the original complete trace leads from Kbely to Karlovo Náměstí. The trace was logged with a frequency of 5 seconds between two consecutive waypoints.

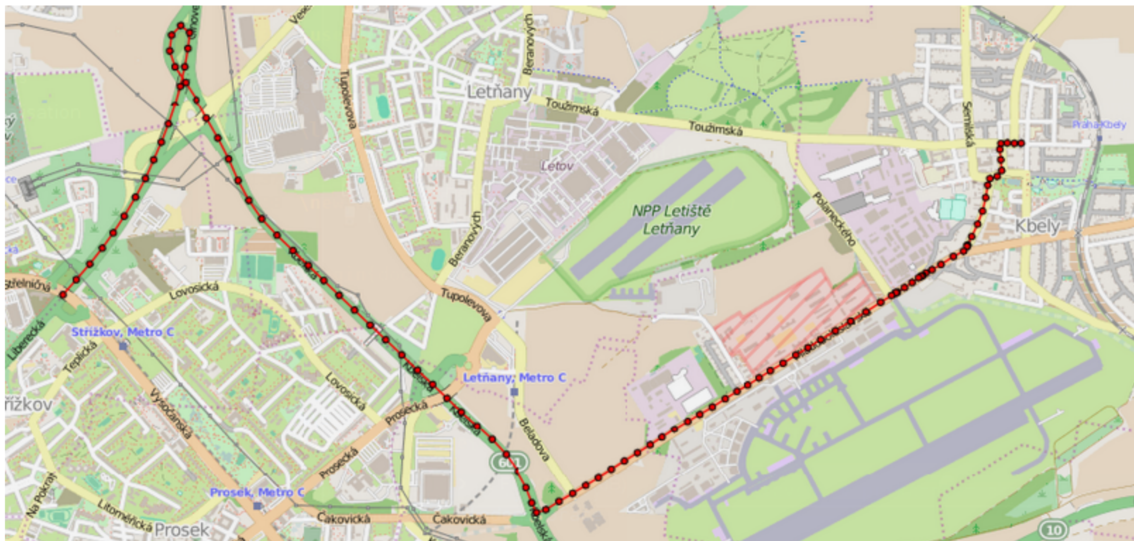
The visualized part of the trace is at its very beginning. In Figure 6.11a you can see the waypoints for the part of the trace. The Figure 6.11b shows the interpolation of the original waypoints and the Figure 6.11c presents the path in the graph the trace was mapped onto.

The reason why most of the evaluation is done on non-real-life (generated) data is that the real-life data are valuable and thus not available for free. This is why this section evaluates only one trace in Prague.

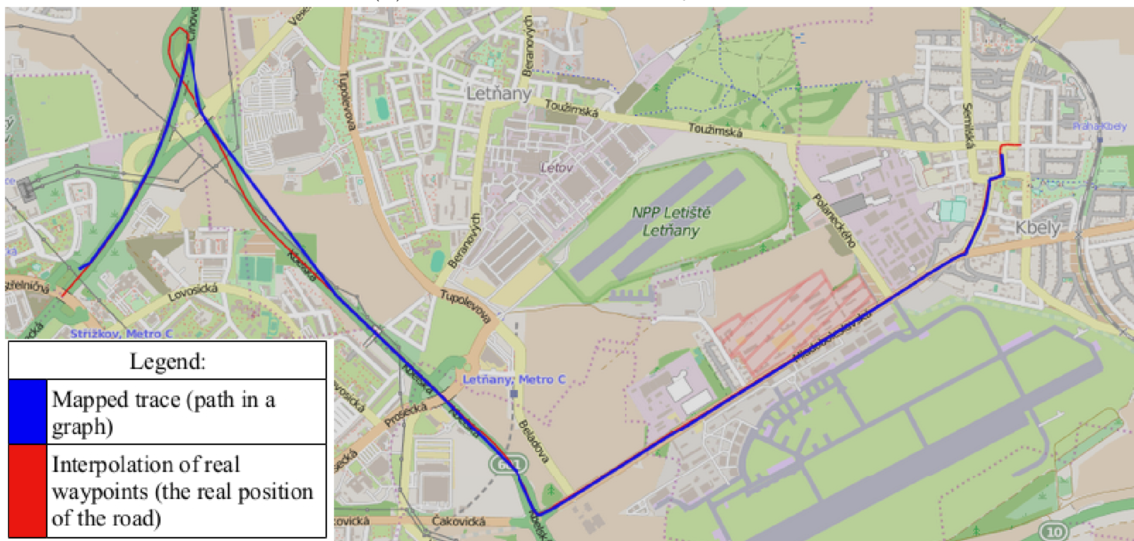
Another complication is that the used graphs are simplified. The simplification causes that some edges of the graph are far from the real position of the road. This means that the Fréchet distance between the interpolation of the waypoints and the mapping of the trace is increased. Thus it is necessary to use higher value of ϵ in order to find some mapping of the original trace and thus more edges have to be taken into account. This increases the amount of time and memory needed to solve the problem. The difference between positions of the real roads and positions of edges in the graph can be seen in Figure 6.12.



(a) Part of the waypoints of the trace.



(b) Interpolation of the waypoints.



(c) Mapping of the part of the trace.

Figure 6.11: The real-world trace.

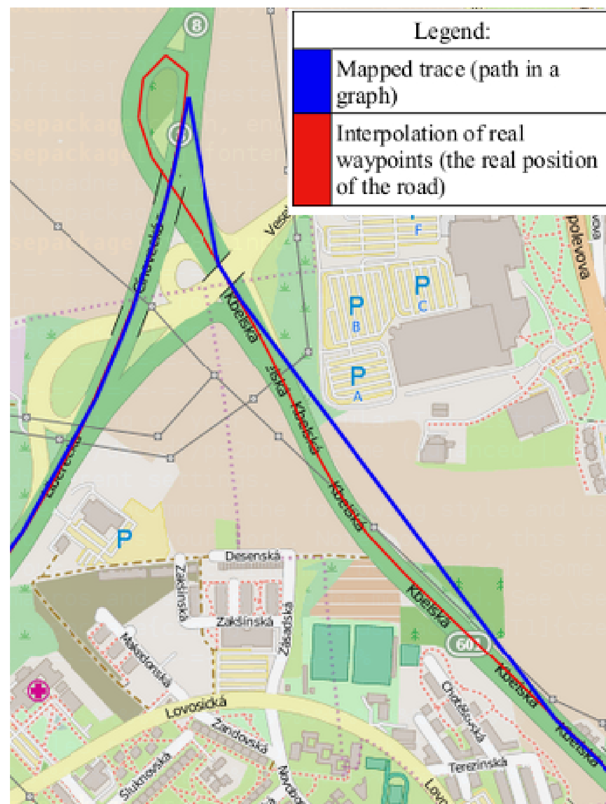


Figure 6.12: Detail of the mapping of real-world trace.

Chapter 7

Conclusion

In this thesis, we formalized the GPS traces graph matching problem and a model for the journey duration estimates. We used an existing GPS traces matching algorithm to find a path in the graph with minimum Fréchet distance from interpolation of the original waypoints. We proposed a method for data extraction and instantiation of speed profile. The proposed solution is implemented and we present its evaluation using generated traces for the graphs of Prague and Brno and using real-world GPS trace for Prague. To evaluate the algorithm we created two speed profiles for Brno, covering approximately 40% and 80% of the graph. The first one is based on 2000 traces while the second uses 27000 generated traces. We also created a speed profile for two time intervals and the graph of Prague based on 10000 generated traces for each of the time intervals.

Furthermore, a variation of the mapping algorithm that saves time and space needed was proposed and implemented. This feature allows us to compute the mappings for extremely large sets of traces such as the 27000 traces used to cover 80% of the graph of Brno with data. The trade-off for the time we save is lower precision of the mappings. However, the quality still remains within reasonable limits. It also allows us to work with larger graphs where the number of edges as well as an average length of traces grows significantly.

As shown in Section Speed Profile for Prague, the speed profile created by the algorithm reflects different levels of traffic. This result is crucial since it shows that the algorithm is capable of using GPS data for journey duration estimates in transport networks. In the following section we mention some ideas on how to further enhance the performance of the algorithm.

7.1 Future Work

The most hardware demanding part of the algorithm is the mapping part. By using different mapping algorithm (e.g., the iterative algorithm presented in [3]) we could trade-off precision for shorter mapping time. This could be a good idea for very large sets of traces and/or graphs of large road networks.

Another thing that we might experiment with is the projection of waypoints onto the mapped path. The used projection assumes implicitly that the ratios of distances between the original waypoints correspond to the ratios of distances between the real locations where the

waypoints were logged. Although it does not have to be true for noisy data, the projection is used since it is fast to compute. It also ensures that we are able to assign some timing to any node in the path by interpolating the data we have. One possible alternative we suggest is to project each original waypoint onto the closest point in the path (with respect to Euclidian distance). This alternative has two main problems. First problem is that there can be more than one solution (since there can be more than one point with the same distance from the waypoint). This can be resolved by either combining this attitude with the implemented one, or by the use of heading that is often included in the GPS data. If we know the heading in all waypoints then we can use it to project the waypoint onto the edge with closest heading. The second problem is that in many cases, we need to extrapolate the data in order to assign some timing to the first and the last node in the path.

Next part of the algorithm we can improve is the method for assigning timings to nodes in the path. The algorithm uses linear interpolation while assigning timings to nodes in the path. This is very fast, but a more complex interpolating methods (e.g., cubic spline) might lead to better results. How the precision of the speed profile changes and whether or not it is worth the increase of computational complexity, is another thing that might be interesting to test.

Finally, we should stress out that although the implementation of the algorithm is using a graph representing roads, it can be easily transformed for different modes of transport (e.g., computing speed profiles for cycling infrastructure from the data gathered in bachelor thesis of Jan Linka [6]). The mapping algorithm could also be used to show which parts of the transport network are the most saturated ones.

Bibliography

[1] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 589–598, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[2] H. ALT and M. GODAU. Computing the frÉchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 05(01n02):75–91, 1995.

[3] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 853–864. VLDB Endowment, 2005.

[4] P. Castro, D. Zhang, and S. Li. Urban traffic modelling and prediction using large scale taxi gps traces. In J. Kay, P. Lukowicz, H. Tokuda, P. Olivier, and A. Krüger, editors, *Pervasive Computing*, volume 7319 of *Lecture Notes in Computer Science*, pages 57–72. Springer Berlin Heidelberg, 2012.

[5] S. Kim, M. Lewis, and I. White, C.C. Optimal vehicle routing with real-time traffic information. *Intelligent Transportation Systems, IEEE Transactions on*, 6(2):178–188, June 2005.

[6] J. Linka. Android app for bicycle route planning and navigation. Bachelor's Thesis, 2015.

[7] G. Nannicini. Point-to-point shortest paths on dynamic time-dependent road networks. *4OR*, 8(3):327–330, 2010.

[8] Wikipedia Contributors. Fréchet Distance. Wikipedia. Wikimedia Foundation. http://en.wikipedia.org/wiki/Fréchet_distance , n.d. Web. 24 Jan. 2015.

Chapter 8

User Guide

The implemented software is provided on the attached DVD in a form of Maven project. Each part of the algorithm is also provided as an executable JAR file.

The traces generating algorithm ($duration-estimates-generator-runner.jar$) serves to generate random GPS traces. The mandatory arguments for this algorithm are expected to be passed in the following order:

1. Positive integer representing number of traces to be generated.
2. Positive integer representing noise degree i.e., the maximal distance (in meters) between the position of the original waypoint and its noisy representation.
3. Positive integer representing time in seconds between two consecutive waypoints.

Optional argument for this algorithm is:

- Name of the city to generate data for. Brno and Prague are accepted. Default value is Brno.

Algorithm that transforms GPX files into trace representation used by the rest of algorithms ($duration - estimates - gpx - loader.jar$) requires two arguments in the following order:

1. Name of the city to prepare data for. Brno and Prague are accepted.
2. Path to the folder containing the GPX files.

Map-matching algorithm ($duration - estimates - mapping.jar$) maps the generated or real-world data onto the graph and the obligatory arguments for this algorithm are expected to be passed in the following order:

1. Non-negative integer representing number of traces to be mapped. If zero is passed, the algorithm maps all available traces.

2. Non-negative integer representing maximal allowed distance (in meters) between the interpolation of the original waypoints and the path in the graph. If set to zero, algorithm uses constant value of 310 meters that is empirically tested to be sufficient for most of the traces that were generated with degree of noise set to 10 meters, and time interval between two consecutive waypoints set to 15 seconds.

3. Name of the city to map the traces onto. Brno and Prague are accepted.

The algorithm that assigns timings to nodes in the mapped trace ($duration-estimates-timing.jar$) requires the Name of the city as the only obligatory argument. Brno and Prague are accepted.

The algorithm that creates an instance of the speed profile has to be run with two arguments in the following order:

1. Name of the city to create speed profile for. Brno and Prague are accepted.

2. Non-negative integer representing number of traces. The algorithm produces speed profile using this number of traces for each time interval. If zero is passed, the algorithm uses all available traces for each interval.

The algorithm for creation of the speed profile creates a representation of the function p as defined in Section 3.2.3. However, if we want to visualize the speed profile, it is possible to pass an additional argument and thus create and visualize the representation of function p_0 that is defined in the same Section. The additional argument is word *vis* (abbreviation for visualize).

All arguments can be passed in both the lower-case and upper-case. Furthermore, all five algorithms can be switched into verbose mode by optional argument v.

All five JAR files are located in directory:

```
duration_estimates/target/.
```

To use them, copy these files to some of your local directories (e.g., $foo/$). Then all the data produced by the algorithms will be stored in

```
foo/DATA_4_BRNO
```

or

```
foo/DATA_4_PRAGUE
```

Thus running:

```
$ java -jar duration-estimates-generator-runner.jar 20 10 15 BRNO v
```

will generate 20 random traces for the graph of Brno and all 24 one hour long time intervals of the day. The used degree of noise is set to 10 meters and time between two consecutive waypoints is set to 15 seconds. The algorithm will run in verbose mode and the generated traces will be stored into folder

```
foo/DATA_4_BRNO
```

The rest of the algorithms can be run in a similar way.

It is recommended to use at least 4GB of memory while working with the graph of Brno, and at least 8GB for working with Prague.

Chapter 9

DVD content

The DVD attached to this thesis contains the implemented software in the form of Maven project and executable JAR files, this thesis and

`README.txt`

describing the content of the DVD.
The Bachelor's thesis is located in the folder:

`text`

The implemented software is located in the folder:

`code`

The folder

`code/duration_estimates`

represents a Maven project. The packages containing the source codes are therefore lacated at folder

`code/duration_estimates/src/main/java/cz/agents/studentproject/slunedan`

The software divided into several executable JAR files is located in:

`code/duration_estimates/target`

Approximately 27000 generated traces for one time interval and graph of Brno are stored together with their mappings, timed traces and speed profiles, in the folder:

`code/duration_estimates/DATA_4_BRNO`

Approximately 10000 generated traces for each of the two time intervals and the graph of Prague are stored together with the speed profile, in the folder:

`code/duration_estimates/DATA_4_PRAGUE`

One real-world trace is stored in the folder:

`code/duration_estimates/DATA_4_PRAGUE_REALWORLD`

Javadoc for the code can be found in the folder:

`code/duration_estimates/doc`

The complete hierarchy of files and folders on the DVD is located in file:

`hierarchy/tree.txt`