MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Argirova Margarita** Personal ID number: **464925**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science and Engineering**

Study program: **Open Informatics**

Branch of study: **Artificial Intelligence**

II. Master's thesis details

Master's thesis title in English:

Enhancing transport network graphs using GPS tracking data

Master's thesis title in Czech:

Zpřesňování grafů dopravních sítí na základě GPS záznamů projetých tras

Guidelines:

Transport network graphs representing the topology of road, cycleway and footpath networks are the basis for automated route planning systems. Unfortunately, network graphs generated from map data contain errors -- edges are either labelled with wrong attributes or completely missing. Such errors can be detected and corrected by analysing GPS tracking data on people or vehicle movement. The aim of this thesis is to explore this approach.
Specific instructions:
1) Familiarize yourself with transport network graphs and their construction from OpenStreetMap maps.
2) Survey existing approaches for using GPS tracking data to enhance transport network graphs.
3) Design and implement a suitable algorithm for GPS tracks-based transport network graph enhancement.
4) Evaluate the enhancement algorithm on real-world map and GPS tracking data.

Bibliography / sources:

[1] Ahmed, M., Karagiorgou, S., Pfoser, D. and Wenk, C., 2015. A comparison and evaluation of map construction algorithms using vehicle tracking data. GeoInformatica, 19(3), pp.601-632.
[2] Liu, X., Biagioni, J., Eriksson, J., Wang, Y., Forman, G. and Zhu, Y., 2012, August. Mining large-scale, sparse GPS traces for map inference: comparison of approaches. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 669-677). ACM.
[3] Cao, L. and Krumm, J., 2009, November. From GPS traces to a routable road map. In Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems (pp. 3-12). ACM.

Name and workplace of master's thesis supervisor:

doc. Ing. Michal Jakob, Ph.D., Department of Computer Science and Engineering, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **19.09.2018** Deadline for master's thesis submission: _____

Assignment valid until: **19.02.2020**

doc. Ing. Michal Jakob, Ph.D. Supervisor's signature	Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

ENHANCING TRANSPORT NETWORK GRAPHS USING GPS TRACKING DATA

by

Margarita Argirova

Bachelor of Engineering, Bauman Moscow State Technical University, 2016.

Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of Master of Computer Science

in the Department of Department of Computer Science

Faculty of Electrical Engineering

Adviser:

Doc. Ing. Michal Jakob, Ph.D.

Czech Technical University

January 2019

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used. I have no objection to usage of this work in compliance with the act §60 Z´akon ˇc. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on December 7, 2019 . . . . . .

ABSTRACT

The aim of this work is to detect to localize missing edges in road graph and propose changes in the map.

The map of Prague was chosen for implementation, testing and visualization as an example of the city with very basic bicycle infrastructure, but the algorithm can be applied to any city or country.

The given data was GPS trajectory from cyclists, who recorded their routes in Prague. This data was preprocessed and filtered to fit the algorithm.

In order to solve the problem of missing edges algorithmically, it was decomposed to some less complexed known ones. First one is map matching, which was solved with modified Dijkstra algorithm with minimization the area between track and graph path.

Optimal paths did not always match with original track, which proved existence of missing edges in the graph. Thus the points lying too far from track were detected as outliers. For experiments were used many algorithms and attributes, but the most precise results were obtained with modified z-score outlier detection technique on interpolated graph paths.

After having the outliers for each path the closest points to outliers were recovered with kd-tree, where the missing edge can exist. From obtained nodes some edges were added until the area between initial track and new graph path decrease.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1 MOTIVATION

According to [24] only 52 out of 1000 inhabitants of Czech Republic own a bicycle and use it for everyday transportation. For Prague the reasons of such low usage are simple, the city is hilly and there is no infrastructure for bikes.

However, there are many software applications, which suggest the shortest route for bicycles between given locations. These applications usually use Google Maps or Open Street Maps for the space representation. But some cyclists report, that some of the routes are far from optimal because of possible errors in the maps, caused by incorrect labeling of the elements.

It happens because cyclist behave both as cars and pedestrians at the same time, therefore they need special treatment in routing.

1.2 AIM

The goals of the thesis are described further.

1. To match the dataset of recorded GPS routes into road network.
2. To detect imperfections of the map for cyclists based on the matched tracks.
3. The errors they should be localized and based on their positions, corrections of the map should be proposed.

CHAPTER 2. PROBLEM SPECIFICATION

2.1 FORMAL DEFINITION

The problem of missing edges detection in the road graph can be decomposed to several less complex ones.

The given data can be represented as set of tracks T and the road graph G. Let the track $T \in \mathrm{T}$ be the sequence of n GPS points recorded from a user route $T = \{(lat_1, lon_1), (lat_2, lon_2), \dots, (lat_n, lon_n)\}$. The map can represented as directed weighted graph $G = (V, E, D)$, where V is the set of vertices with GPS coordinates latitude and longitude, E is the set of edges between nodes and D is the set corresponding distances between nodes.

Graph path P is the 'projection' of the track on the graph. *Map matching* is the problem of how to match recorded geographic coordinates to a logical model of the real world and relate them to edges in an existing street graph (network), more details are presented in Section 3.1.

The path P is the optimal sequence of m GPS points obtained after matching the track T into graph G, $P = \{(lat_1, lon_1), (lat_2, lon_2), \dots, (lat_m, lon_m)\}$. The distance

between track in corresponding path is defined as $F(T, P)$ and its formal definition will be discussed in Section 4.2.

Based on the distance F it has to be decided, if the path P is optimal path for the given track T.

Let F_i be the distance between i^{th} point of track T and path P. Then $F(T, P)$ is a list of F_i. After processing of all input tracks T and recording the distances of their points, there can be applied anomaly or outlier detection techniques.

Outlier is an observation point that is distant from other observations. Anomaly detection (also outlier detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. [] Three broad categories of anomaly detection techniques exist [22, 23]. Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal by looking for instances that seem to fit least to the remainder of the data set. Supervised anomaly detection techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier (the key difference to many other statistical classification problems is the inherent unbalanced nature of outlier detection). Semi-supervised anomaly detection techniques construct a model representing normal behavior from a given normal training data set, and then test the likelihood of a test instance to be generated by the learnt model. [22]

It is not too convenient to label all processed data based on visual evaluation. Thus further in Section 6.2 there will be described unsupervised anomaly detection techniques to classify the track-path pair into ‘successful’ or ‘failure’.

2.2 POSSIBLE SOURCES OF THE PROBLEM

There might be various sources of these errors. Usually they are missing or excess segments of the real world path.

- Some of the ways are wrongly “oneway” tagged or bikes are were not mentioned.

- Other cases can be caused by multimodality of the graph with no connection points between pedestrian and car modes (as bicycle might behave like both).
- Also it is possible to be a wrong tagging of the way or node.
- The real city is always changing and the map can not always track these changes.

2.3 OUTLINE OF THE ALGORITHM

The algorithm for graph refinement requires further steps:

1. Parsing the street network (Section 5.3)
2. Creating the transport graph (Section 5.3)
3. Filtering the tracks (Section 5.2)
4. Matching tracks with the map (Sections 5.4, 4.2)
5. Visually detecting the errors
6. Classification each point as inlier or outlier (Sections 3.1, 5.8)
7. Localization the problematic graph nodes (Section 5.9)
8. Refining the transport graph (Sections 5.9, 6.4.4)

CHAPTER 3. RELATED WORK

3.1 MAP MATCHING ALGORITHMS

There are many different algorithms for matching the GPS-track with the transport graph. They can be roughly divided in three groups: geometrical, probabilistic, other.

3.1.1 ALGORITHMS BASED ON GEOMETRY

In [2] further algorithm was proposed. Given a series of position samples representing a vehicle trajectory, the map-matching algorithm pursues a position-by-position sample and edge-by-edge strategy. To match a position p_i to a road network edge, given that its previous position p_{i-1} has already been matched, the algorithm proceeds as follows (figure 1). First, the candidate edges to be matched to the current position are identified as the set of the incident edges "exiting" the last matched edge (including also the matched edge itself). In figure 1, these edges are labeled c_1, c_2 and c_3, with c_3, being the edge matched to p_{i-1}.

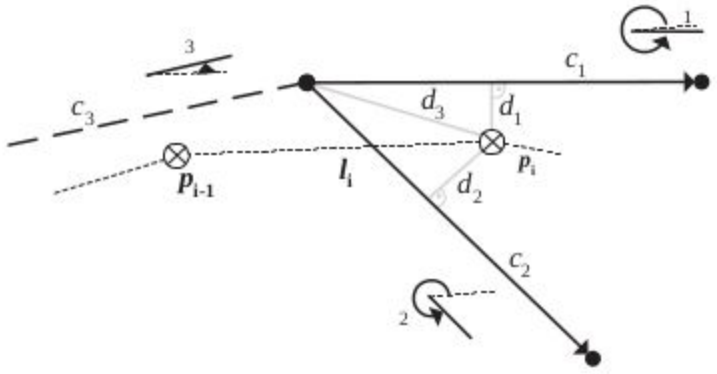


Figure 1. Incremental map-matching example. [2]

Two similarity measures are used to evaluate the candidate edges.

The measure c_d reflecting the distance from the position sample to the various edges is computed based on the weighted line segment distance, d, of p_i from each candidate, c_j , using the scaling factors μ_d and n_d as:

$$s_d(p_i,\ c_j) \ = \ \mu_d \ - a \cdot d(p_i,\ c_j)^{n_d} \qquad (1)$$

The measure s_α reflects the orientation of the trajectory with respect to the candidate edge. It is computed based on the angle difference $\alpha_{i,j}$ between the directed candidate edge c_j and the directed line segment $l_i = \overline{p_{i-1},\ p_i}$, using the scaling factors μ_α and n_α as:

$$s_\alpha(p_i,\ c_j) \ = \ \mu_\alpha \cdot cos(\alpha_{i,j})^{n_\alpha} \qquad (2)$$

The scaling factors $\mu_{[d|\alpha]}$ and $n_{[d|\alpha]}$ represent the maximum score and a power parameter, respectively. Choosing a higher μ_d compared to μ_α means that distance weighs more than orientation. The power parameter determines the rate of decrease for

the respective weight with an increasing line segment distance or angle difference. The use of the cosine further implies that with an increasing angle difference the score of s_α decreases and with angle differences $90 < \alpha < 270$ and the choice of an odd number for the power n_α and a positive constant μ_α, s_α even becomes negative.

The combined similarity measure is computed as the sum of the individual scores, i.e.,

$$s = s_\alpha + s_d \tag{3}$$

The higher the score of this measure, the better is the match.

Depending on the type of projection/match of p_i to c_j, i.e., (i) its projection is between the endpoints of c_j, or, (ii) it is projected onto the end points of the line segment, the algorithm does, or does not advance to the next position sample. Following the example of Figure 2, after matching p_1 to edge e_1, the algorithm advances to p_2 (case (i)) and matches it also to e_1. Advancing to p_3, it tries to match this point to e_2 and since this projection reflects case (ii) it does not advance to the next position sample but finally matches e_3 to p_3. The edge e_2 is recorded as a traversed edge. In Figure 2, the mapped position samples are drawn as gray circled crosses.

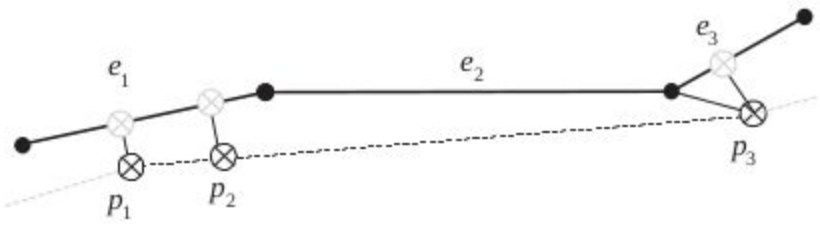


Figure 2. Matching example: advancing position samples

and edges, and matching result. [2]

[2, 3, 4]

3.1.2 PROBABILISTIC MAP MATCHING ALGORITHM

In [1] was introduced probabilistic map matching algorithm Hidden Markov Models (HMM). The HMM models processes that involve a path through many possible states, where some state transitions are more likely than others and where the state measurements are uncertain. the states of the HMM are the individual road segments, and the state measurements are the noisy vehicle location measurements. The goal is to match each location measurement with the proper road segment. This state representation naturally fits the HMM, because transitions between road segments are governed by the connectivity of the road network.

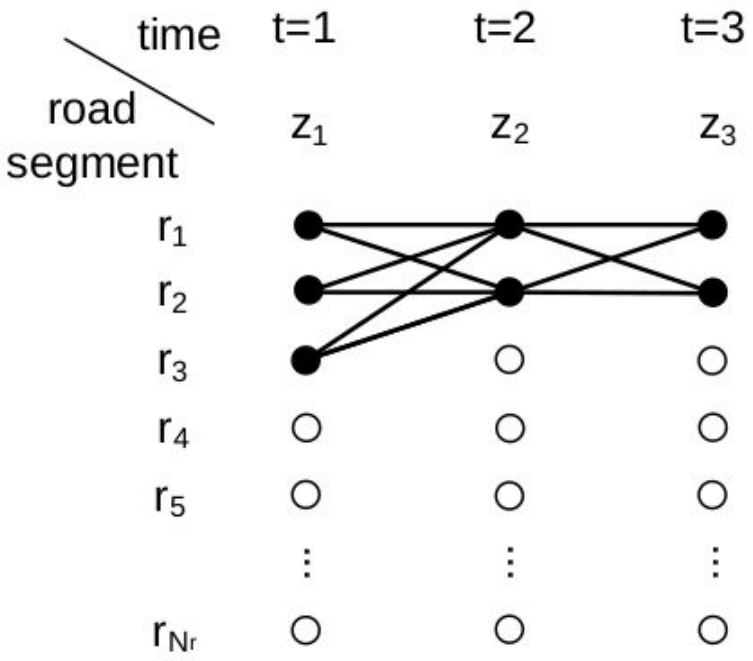


Figure 3. For each measurement z_t , the HMM considers all the road segments r_i as well

as all the transitions between the road segments. [1]

More formally, the discrete states of the HMM are the N_r road segments, r_i, i = 1 ... N_r. In this representation, distinct road segments run between intersections. For each 2D latitude/longitude location measurement z_t , the goal is to find the road segment that the vehicle was actually on. Figure 3 shows an illustration of the HMM for the map matching problem. Here, each vertical slice represents a point in time corresponding to a location measurement z_t for the three times t = 1, 2, 3. At t = 1 there are three roads near z_1 , shown as three black dots in the first column. There is a feasible driving path, possibly very circuitous, from each of the nearest points on these three roads to points on the two roads near z_2 at t = 2, and similarly for t = 3. The goal of the algorithm is to find the most probable path through the lattice by picking one road segment for each t. This path should be sensitive to both the measurements and the reasonability of the paths between the road segments. This tradeoff is made based on the probabilities governing the measurements and probabilities governing the transitions between the road choices at each time.

This algorithm utilizes measurement probabilities, the likelihood that a measurement resulted from a given state, based on that measurement alone. The transition probabilities are the probabilities of a vehicle moving between the candidate road matches at moments t and t+1. These two kinds of probabilities are used in Viterbi algorithm to compute the best path through the HMM lattice. The Viterbi is essentially dynamic programming, and quickly finds the path through the lattice that maximizes the product of the measurement probabilities and transition probabilities. In constructs an inference of the correct road segment for each location measurement. [1]

3.1.3 OTHER ALGORITHMS

In [8] there was vaguely described an idea of algorithm similar to Dijkstra's shortest path: [5]

The graph *G (V, E, F)*, where V, E - vertices and edges of the transport graph, F, unlike D, are distances between given track T and the path P in the transport graph. The algorithm incrementally builds the shortest path P point by point, minimizing the distance between current point and the track. The details of this approach and its modifications are described in Section 4.2.

3.2 OUTLIER DETECTION ALGORITHMS

There were considered some methods of anomaly detection: K-means algorithm and its varieties and three statistical methods.

3.2.1 K-Means

On 2 dimensional area-distance data was tested K-Means clustering algorithm. The definition of the algorithm is: Let $X = \{x_i\}$, $i = 1, ..., n$ be the set of n d-dimensional points to be clustered into a set of K clusters, $C = \{c_k, k = 1, ..., K\}$. K-means algorithm finds a partition such that the squared error between the empirical mean of a cluster and the points in the cluster is minimized. Let μ_k be the mean of cluster c_k. The squared error between μ_k and the points in cluster c_k is defined as:

$$J(c_k) = \sum_{x_i \in c_k} ||x_k - \mu_k||^2 \qquad (4)$$

The goal of K-means is to minimize the sum of the squared error over all K clusters:

$$J(C) = \sum_{k=1}^{K} \sum_{x_i \in c_k} ||x_k - \mu_k||^2 \qquad (5)$$

Minimizing this objective function is known to be an NP-hard problem (even for K = 2). Thus K-means, which is a greedy algorithm, can only converge to a local minimum, even though recent study has shown with a large probability K-means could converge to the global optimum when clusters are well separated. K-means starts with an initial partition with K clusters and assign patterns to clusters so as to reduce the squared error. Since the squared error always decreases with an increase in the number of clusters

K (with $J(C) = 0$ when $K = n$), it can be minimized only for a fixed number of clusters. The main steps of K-means algorithm are as follows:

1. Select an initial partition with K clusters; repeat steps 2 and 3 until cluster membership stabilizes.
2. Generate a new partition by assigning each pattern to its closest cluster center.
3. Compute new cluster centers.

The K-means algorithm requires three user-specified parameters: number of clusters K, cluster initialization, and distance metric. The most critical choice is K. While no perfect mathematical criterion exists, a number of heuristics are available for choosing K. Typically, K-means is run independently for different values of K and the partition that appears the most meaningful to the domain expert is selected. Different initializations can lead to different final clustering because K-means only converges to local minima. One way to overcome the local minima is to run the K-means algorithm, for a given K, with multiple different initial partitions and choose the partition with the smallest squared error. K-means is typically used with the Euclidean metric for computing the distance between points and cluster centers. As a result, K-means finds spherical or ball-shaped clusters in data. K-means with Mahalanobis distance metric has been used to detect hyperellipsoidal clusters, but this comes at the expense of higher computational cost.

In K-means the outlier is an element of the group which has few data and is located far from other groups and can be considered as new cluster.

It was tested the basic version of K-means with different cluster number and Euclidean distance metric. [9, 10, 11, 12, 13]

3.2.2 K-Medoids

K-medoids is very similar to K-means, but represents each cluster using an actual point and a radius rather than a prototype (average) point and a radius. K-medoids is robust to outliers as it does not use optimisation to solve the vector placement problem but rather

uses actual data points to represent cluster centres. K-medoids is less susceptible to local minima than standard k-means during training where k-means often converges to poor quality clusters. It is also data-order independent unlike standard k-means where the order of the input data affects the positioning of the cluster centres and it is shown that k-medoids provides better class separation than k-means and hence is better suited to a novelty recognition task due to the improved separation capabilities. However, k-means outperforms k-medoids and can handle larger data sets more efficiently as k-medoids can require O(n^2) running time per iteration whereas k-means is O(n). [14]

3.2.3 Z-Score

Let $X_1, X_2, \ldots, X_n$ be random samples of some value. The mean is calculated as:

$$m = \frac{\sum_i^n x_i}{n} \tag{6}$$

And the standard deviation:

$$s = \sqrt{\frac{\sum_i (x_i - m)^2}{n-1}} \tag{7}$$

Z-score expresses a particular score in terms of how many standard deviations (figure 4) it is away from the mean and can be computed as:

$$Z_i = \frac{x_i - m}{s} \tag{8}$$

Outliers are detected based on the value of |Z-score|, if it exceeds 3 or 4. In case of distances it makes sense to discard the absolute value, because even values less than Z-score are technically outliers, but closer distance is better. [16]

3.2.4 Modified (robust) Z-Score

Although it is common practice to use Z-scores to identify possible outliers, this can be misleading (particularly for small sample sizes) due to the fact that the maximum Z-score is at most $(n-1)/\sqrt{n}$. In [17] there was proposed using the modified Z-score:

$$Z_i = \frac{0.6745(x_i - median)}{MAD} \quad (9),$$

where

$$MAD = median(|x_i - median(x)|) \quad (10)$$

The outliers are those values, modified Z-score of which is greater than 3.5. The method is robust to potential outliers because of using median value instead of mean. [17]

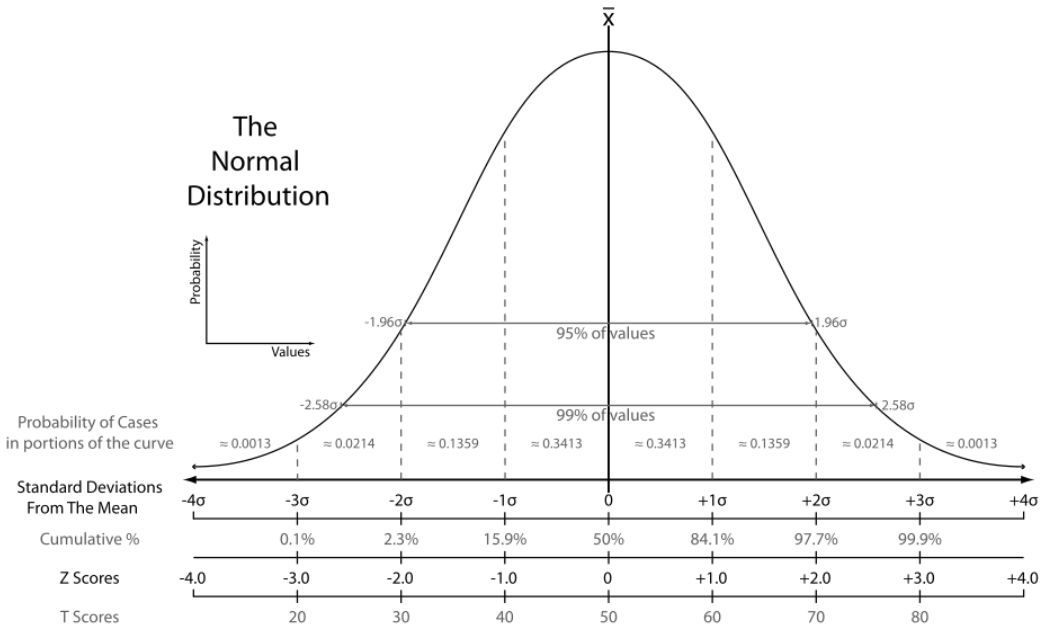


Figure 4. Normal distribution with mean, standard deviation and Z-scores [21].

3.2.5 Interquartile range

Interquartile range or IQR is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles,

$$IQR = Q_3 - Q_1 \qquad (11).$$

Percentile is a number where a certain percentage of scores fall below that number. It is calculated as:

$$R = P/100\ (n + 1) \qquad (12),$$

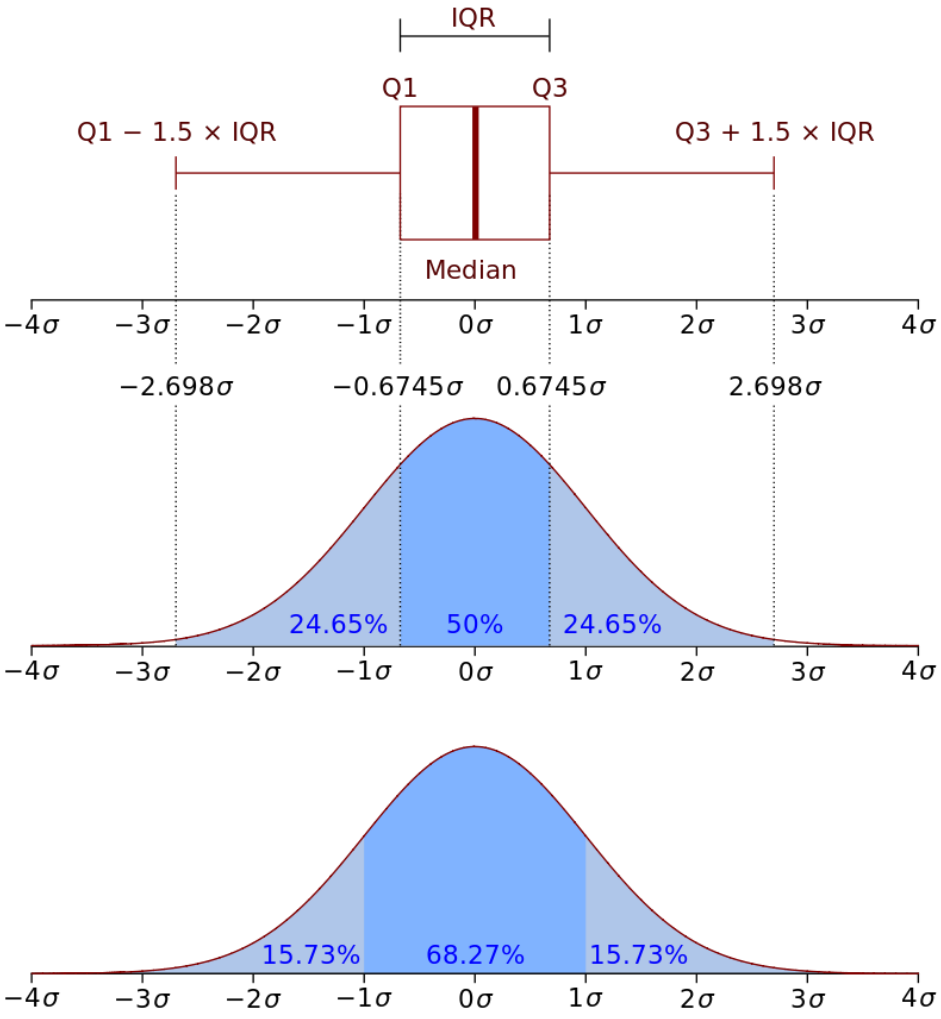


Figure 5. Normal distribution with Interquartile ranges. [20]

where P is percent, n - number of measurements in the data.

These quartiles can be clearly seen on a box plot on the data (Figure 5). It is a trimmed estimator, defined as the 25% trimmed range, and is a commonly used robust measure of scale.

The IQR is a measure of variability, based on dividing a data set into quartiles. Quartiles divide a rank-ordered data set into four equal parts. The values that separate parts are called the first, second, and third quartiles; and they are denoted by Q1, Q2, and Q3, respectively.

Outliers here are defined as observations that fall below Q1 − 1.5 IQR or above Q3 + 1.5 IQR . As in Z-scores, for deviating distance detection was used only Q3 + 1.5 IQR. [18, 19]

CHAPTER 4. SOLUTION

4.1 SOLUTION ARCHITECTURE

For map matching there was implemented and refined the Dijkstra shortest path algorithm and its functions for optimization. Details are described in this chapter in Section 4.2.

After processing the tracks T and obtaining the paths P on the dataset, all the distances F were stored to detect the outliers. For anomaly detection there were implemented K-means, K-medoids, Z-scores, modified Z-scores and IQR and then compared in Section 6.4.

After the data classification there were implemented point localization algorithm, described in Section 5.9 and 4.4.

After point localization there were also implemented naive approach to refine the graph (Section 4.5).

The testing, evaluation and results are discussed in Section 6.4. Details and justification of algorithmical options are explained as well.

The final pipeline of the algorithm is presented below:

1. Parse the given data, obtain graph G and tracks T.
2. For each input track perform the graph matching modified Dijkstra shortest path algorithm with minimization of the area between track T and path P.
3. Collect the statistics for each path and for each path point, representing the distance to closest track point and store it together.
4. Perform the modified (robust) Z-score outlier detection on the dataset obtained above.
5. Localize the points, where the distance starts to grow and store them to another dataset.
6. Suggest the improvements in the graph.

4.2 MODIFIED DIJKSTRA SHORTEST PATH ALGORITHM

These distances can be defined as Fréchet distances between two curves. Practical definition of Fréchet distance is Hausdorff distance. Let A = { a_1 , a_2 , ... , a_n } and B = { b_1 , b_2 , … , b_m } in E^2 . The one-sided Hausdorff distance from A to B is defined as [6, 7]:

$$\delta_H (A, B) = max_{a \in A} min_{b \in B} ||a - b|| \quad (13)$$

In Dijkstra's algorithm each node appears incrementally, therefore the aligned path changes its length. The Hausdorff distance can be reformulated incrementally:

$$\delta_H (A, b) = min_{a \in A} ||a - b||, for\ b \in B \quad (14)$$

Thus, the problem of fitting the track to the graph can be formulated as optimization task of minimization the distance between two discrete set of points. [8]

The last algorithm shows neat results and will be used in further implementation with several enhancements.

Firstly, there are several approaches to define minimum distance objective function.

1. Minimization of vanilla Hausdorff distances (minimum).
2. Minimization of the sum of incremental minimum distances.
3. Minimization of the area between two discrete sets of points.

Computation of the area between two curves is not trivial task, therefore there was suggested further approximation algorithm.

1. The first point in the matched track (curve B) has the closest first point in original track (curve A). Initial area between tracks is equal to zero (figure 6).

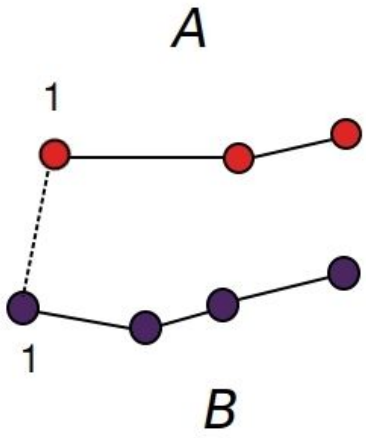


Figure 6. First step of area approximation algorithm.

2. Dijkstra algorithm produces the next point. It's neighbours' distances can be computed for two situations:
    a. The closest point in A of previous point in B is known and is X. If the closest point of neighbour is also X, the area is computed as a triangle △12X (figure 7).

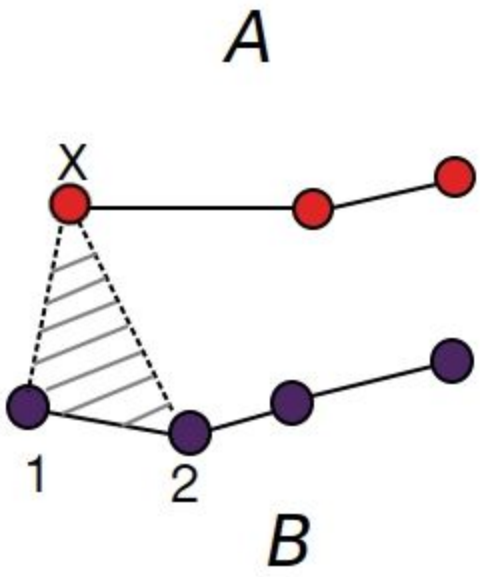


Figure 7. The second step of the area approximation algorithm.

b. The closest point of the previous point is X, of the neighbour is Y. Therefore the area is computed as △12X plus △3XY or more additional triangles (figure 8):

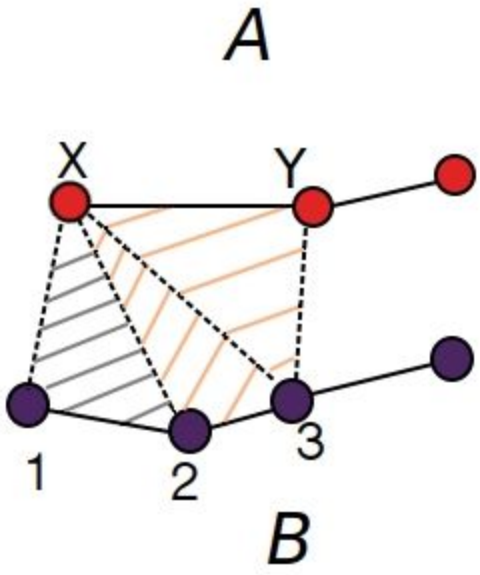


Figure 8. The third step of the approximation algorithm.

Before match starts, the first closest node should be found. This can be done using kd-trees, containing all nodes with coordinates from the graph.

Kd-tree is data structure, proposed in []. It implements search in d-dimensional space and each level has a “cutting dimension”. The search performs by cycling through the dimensions down the tree. Each node contains a point P = (x,y), to find desired point it is necessary to compare coordinate from the cutting dimension.

The final matching algorithm pseudocode with all optimizations and enhancements is presented on the next.

```
function modified_dijkstra()
    input: Graph G(V, E, D), track T
    output: path P
    begin
        target = get_closest_node_in_G_to_last_point_in_T
        starts = get_closest_3_nodes_in_G_to_first_point_in_T
        for (start in starts)
            pq = initialize_priority_queue
            pq.insert(start)
            if (start is target)
                break
            endif
            while (pq is not empty)
                node = pq.poll()
                for (n in node's neighbourhood)
                    area1 = node.get_area_so_far
                    area3 = compute_area(node, n)
```

```
                        area = area3 + area1
                        if (n is visited) and (pq.contains(n))
                            area2 = n.get_area_so_far
                            if area > area2
                                n.set_area_so_far(area)
                                n.set_parent(node)
                            endif
                        endif
                        else
                            n.set_parent(node)
                            n.set_area_so_far(area)
                            pq.insert(n)
                        endelse
                    endfor
                endwhile
                Ti = backtrack()
            endfor
            T = get_track_with_minimal_area_between(Ts)
        end
```

4.3 PROBLEMATIC POINTS EXTRACTION

After extraction of all outliers it is necessary to detect actual nodes in graph, where there can be a problem. It is clear, that the outliers themselves are not these points, but first approximations to them, therefore special detection algorithm should be implemented. The algorithm is presented below.

1. If point is inlier, pass.
2. If point is first outlier in sequence, remember the point, go backwards to find previous inlier.
3. If point is an outlier after first processed, pass, further outliers in sequence are dependent on the first one.

The result of the algorithm is an approximation of the actual node with missing edge.

4.4 GRAPH REFINEMENT

For demonstration of correctness of suggested approach, there was implemented simple semi-bruteforce algorithm to refine the road graph. It iterates through the detected points and its' closest neighborhood:

1. Search n closest nodes to detected one using kd-trees.
2. If the node is in path, select those neighbours, which do not have edge forward, but have the backward.
3. Recalculate the path,
4. If it shows significantly smaller area between track and path, this edge should be added.

CHAPTER 5. IMPLEMENTATION DETAILS

Java was used for code implementation. It is relatively fast and has many of necessary libraries.

There were implemented the majority of described algorithms, necessary data structures and visualizations tools for presentation, evaluation and debugging.

5.1 GIVEN DATA

Firstly, the graph based on the map should be created. The data was taken from open-source project Open Street Maps. The data is stored in XML-like format. The file with the map contains 3 types of entities, Nodes, Ways and Relations. Node represents the GPS-location of some entity, Way is an array of Nodes and Relation is a structure of Nodes, Ways or other Relations representing some object.

Secondly, there are available more than 2000 GPS tracks from real users. Each file represents a list of GPS points, recorded during the bike trip.

Some basic structures for creating a graph, such as GraphBuilder, Node, Edge etc. were provided by Umotional company.

5.2 DATA INSPECTION

Before processing the data should be inspected and filtered. Common problems were:

1. Too long routes. Processing them causes OutOfMemory error and these tracks should be divided into several of them.
2. Routes with “teleportation”. Between some of the points distance is too big to cover, it means, that the tracker lost the connection, for example in metro. These tracks are too hard to map, therefore they should be broken into smaller once.
3. Routes or parts of them outside the given map.
4. Circular routes or those where starting and ending points are too close to each other. Unfortunately, proposed algorithm cannot deal with these tracks and they be discarded.

5.3 PARSING THE DATA AND CREATING THE TRANSPORTATION GRAPH

Open Street Maps data is stored in XML-like format. For parsing was chosen SAX XML parser. Parsed objects were stored in classes OSMNode and OSMWay.

Original file contains unnecessary data, so only data related to routes has to be extracted.

Nodes and their attributes are listed in the beginning of the OSM-file. So at first the list of them is created without any filtering. Next, there listed all the Ways with references to contained nodes. On this step only ways with attribute ‘highway’ have to be picked. Highways also include pedestrian routes, paths and any other type of the roads. The Way can have the attribute ‘oneway=yes’, which indicates one-way direction, and ‘oneway=yes:bikes’, which indicates both-ways only for bikes. This data has to be stored in boolean attribute ‘oneway’ of OSMWay class.

The section of OSM-file containing Relations can be easily skipped.

Before the graph creation, OSM-nodes and OSM-ways have to be converted to graph nodes and edges. Only nodes mentions in at least one way are added to the graph. Also the single edge is created according to the flag ‘oneway’ is ‘true’ and both-way edge, if the flag is ‘false’.

The last step is creating the graph itself with help of the Graph class from the base-structures.

5.4 MAP MATCHING DETAILS

For track matching there is GraphProcessor class, implementing dijkstra() function and other helping functions.

Using the kd-tree to find several closest to the first track point, it iterates through them, implementing the algorithm, presented in Section 4.2. It also records the statistics, discussed in 5.6.

5.5 GEO VISUALIZATION TOOLS

For better representation of results and also for debugging purposes there were implemented visualization instruments.

The main application extends PApplet class from Java Processing and Unfolding Maps libraries. Depending on the purpose of visualization there can be several modes:

1. Track mode. Simply represents the GPS-track from the file as a red line on the map (figure 9).

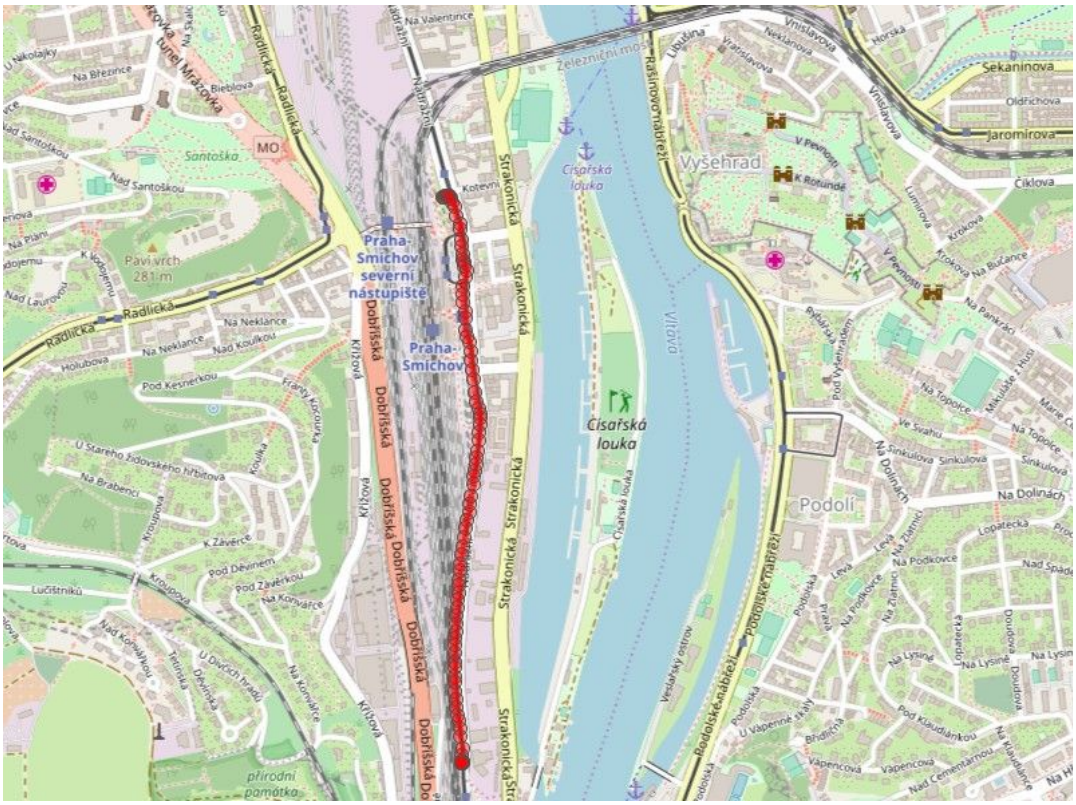


Figure 9. The “track mode”.

2. Track + result mode (“fit mode”). Represents the given track as a red line and resulting track from the next sections as a blue line (figure 10).

Figure 10. The “fit mode”.

3. Track + result + graph mode (“all mode”). All the same above with part of the graph on this area, represented as light green nodes and edges (Figure 11).

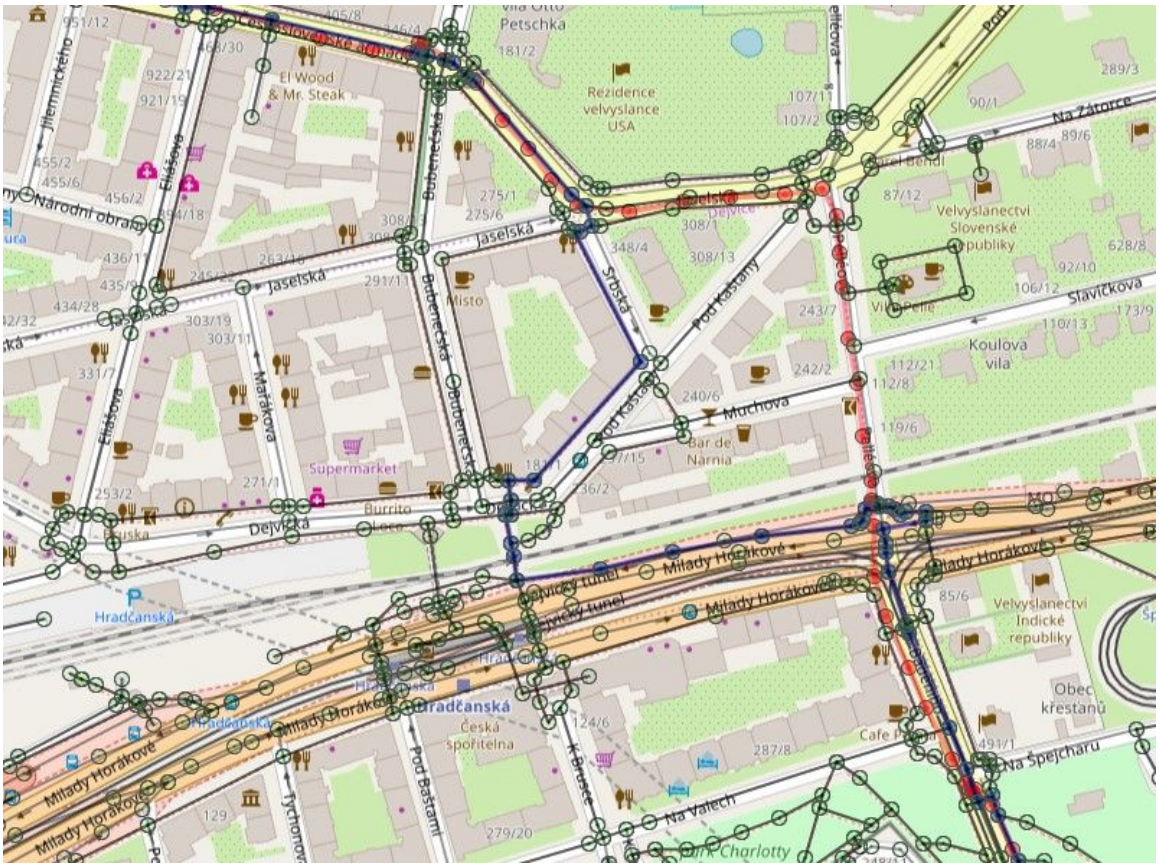


Figure 11. The “all mode”.

4. Classified fit mode. After outlier detection it was necessary to show (figure 12) the original track (red line) and the path (blue line) with outliers highlighted (green dots).

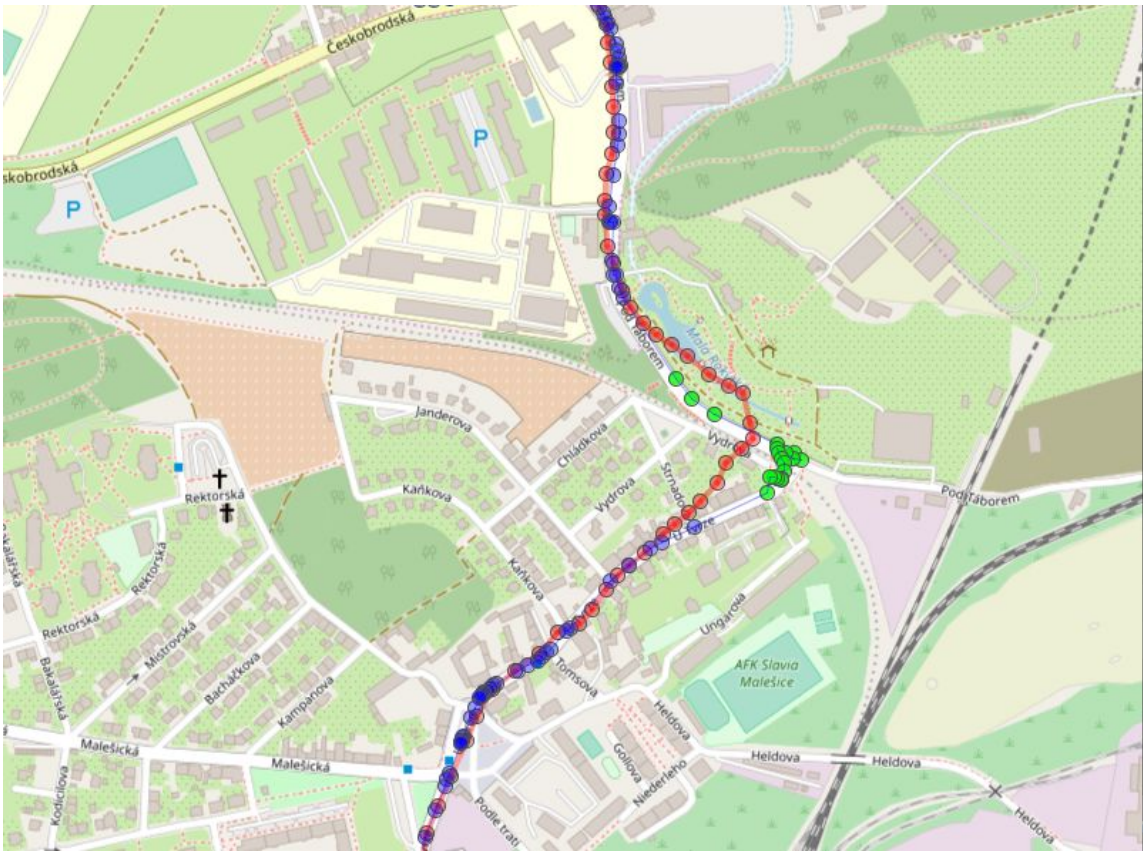


Figure 12. The “classified mode”.

5. There were also implemented other debugging tools, such as raw nodes/edges viewer, graph-only mode and so on.

To help the user navigate the map with lines there were implemented simple zoom and pan tools with possibility to save the picture.

5.6 STATISTICS CALCULATION

During the track fitting there were also performed track and graph path statistics recording. It is necessary to transform them later into features for clustering and classification.

Important information to keep about the tracks is distances between points. In most of the input tracks these distances are more or less homogeneous as the cyclist

moves with constant speed. The opposite is for the graph paths, on highways distances are usually longer than in crossroads.

The modified Dijkstra algorithm minimized the area between input track and output graph path. So the area increment is next attribute for each path point. The original Dijkstra approach minimized the Hausdorff distance between tracks and however it was modified, the distances are available to extract for each point and it is the next feature to consider.

For processing there was chosen the combination of distance and area for each point in each path. In the code it was represented as List of StatUnitAreaDistance.

Another approach utilized average distance between points in track and in path. They were used to interpolate the track and the path to approximate the normalization of each distance between the curves, which were calculated after interpolation.

Interpolation was just simply adding points between the nodes, if the distance between them was more than minimum of distance averages of path and track.

$$latNew_i = lat_1 + (lat_2 - lat_1)\, i / n$$

$$lonNew_i = lon_1 + (lon_2 - lon_1)\, i / n \qquad (15),$$

where i is point index, n - number of points to insert.

In the Figure 13 there are interpolated curve (blue) and initial track (red).

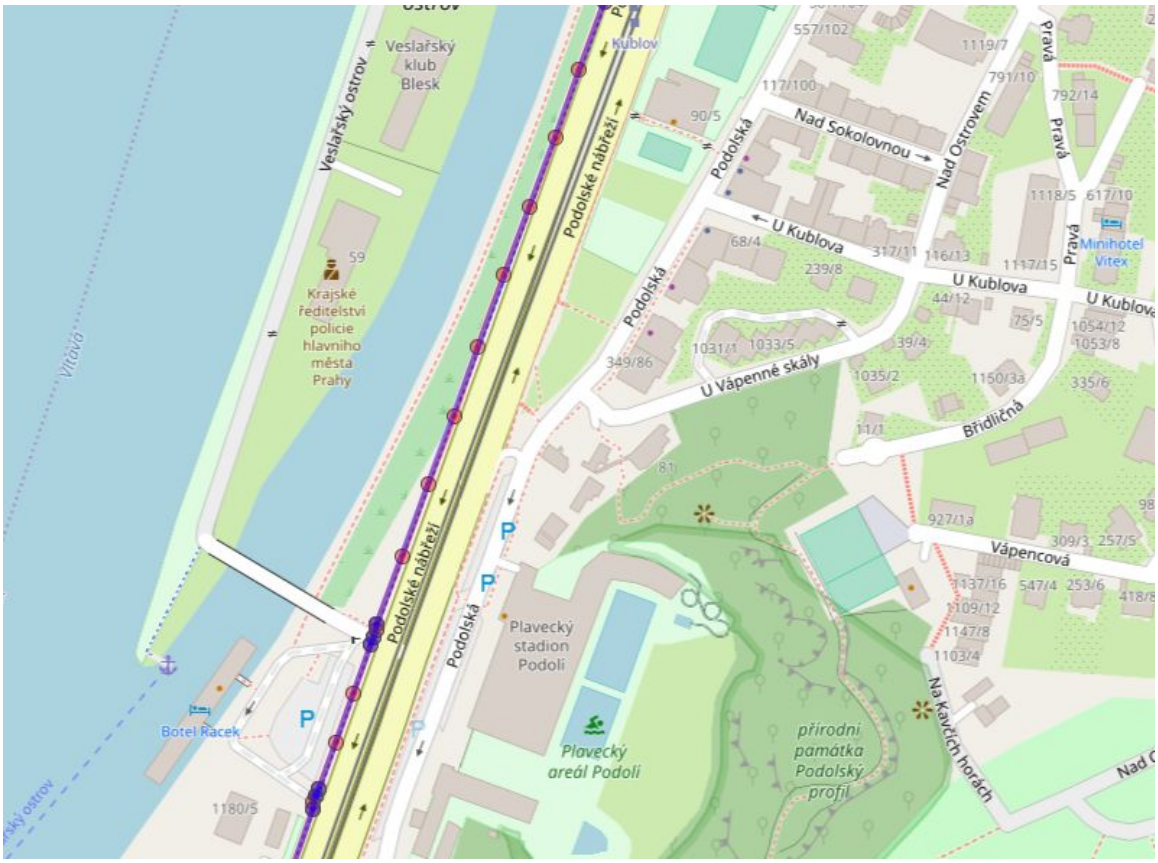


Figure 13. Input track (blue), interpolated track (red).

In the code this statistic was represented as List of StatUnitInterpolatedDistance.

The last statistic named as StatUnitInvDistance was constructed as minimal distance from track point, apart from previous approach, where the distance from path to track was used, to path.

In experiments there were tested all three statistics, in further sections there are results provided.

5.7 STATISTICS VISUALIZATION

Tracks and paths were better to visualize on the map, but for areas and distances different type of representation was implemented.

JavaFX library provides simple Application to extend and simple scatter, line and bar plots.

Scatter plot was used to visualize 2D points with attributes area and distance (Figure 14). Area is function of distance, but experiments showed, that they are not fully replaceable and behave differently sometimes.

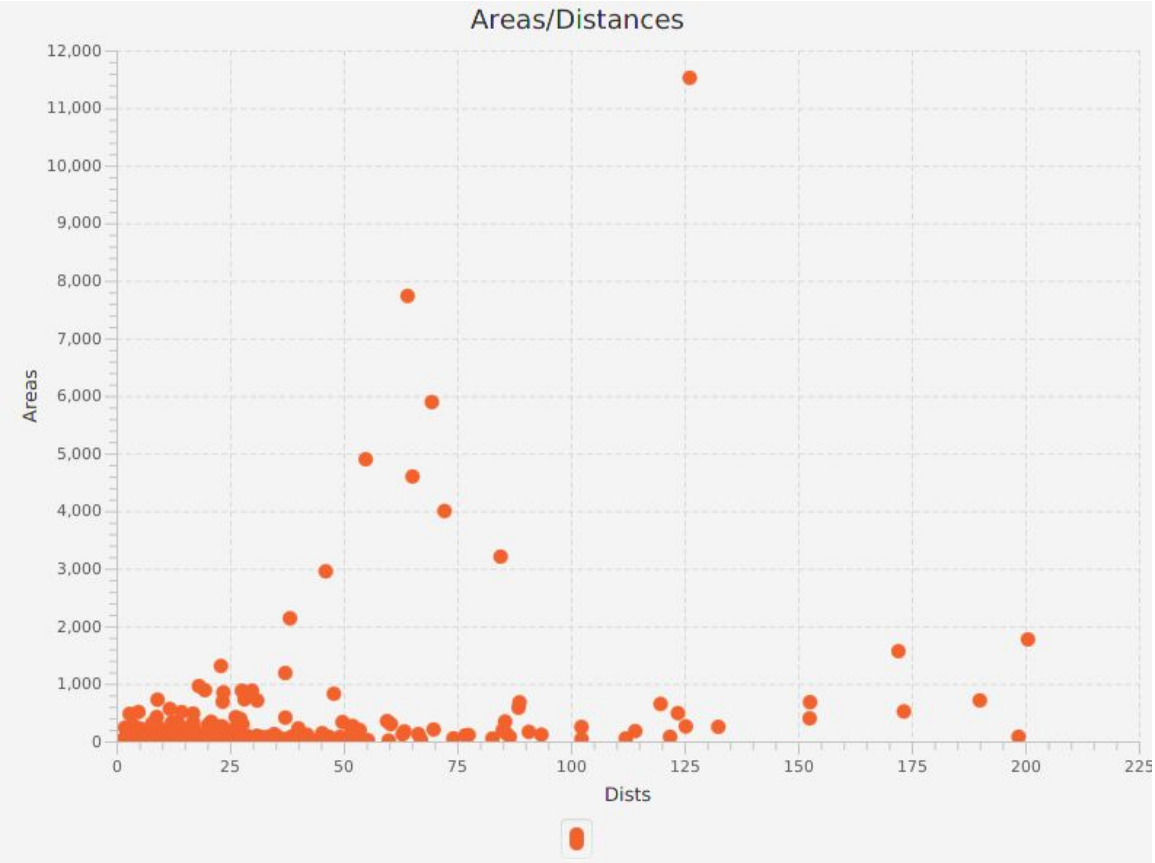


Figure 14. 2D scatter plot.

On the plot it is visible, that many points with high distance are do not show the growth in area attribute.

For visualization of distances through ordered points the line plot was chosen. Unlike the 2D scatter plot, this one preserves the order (Figure 15).

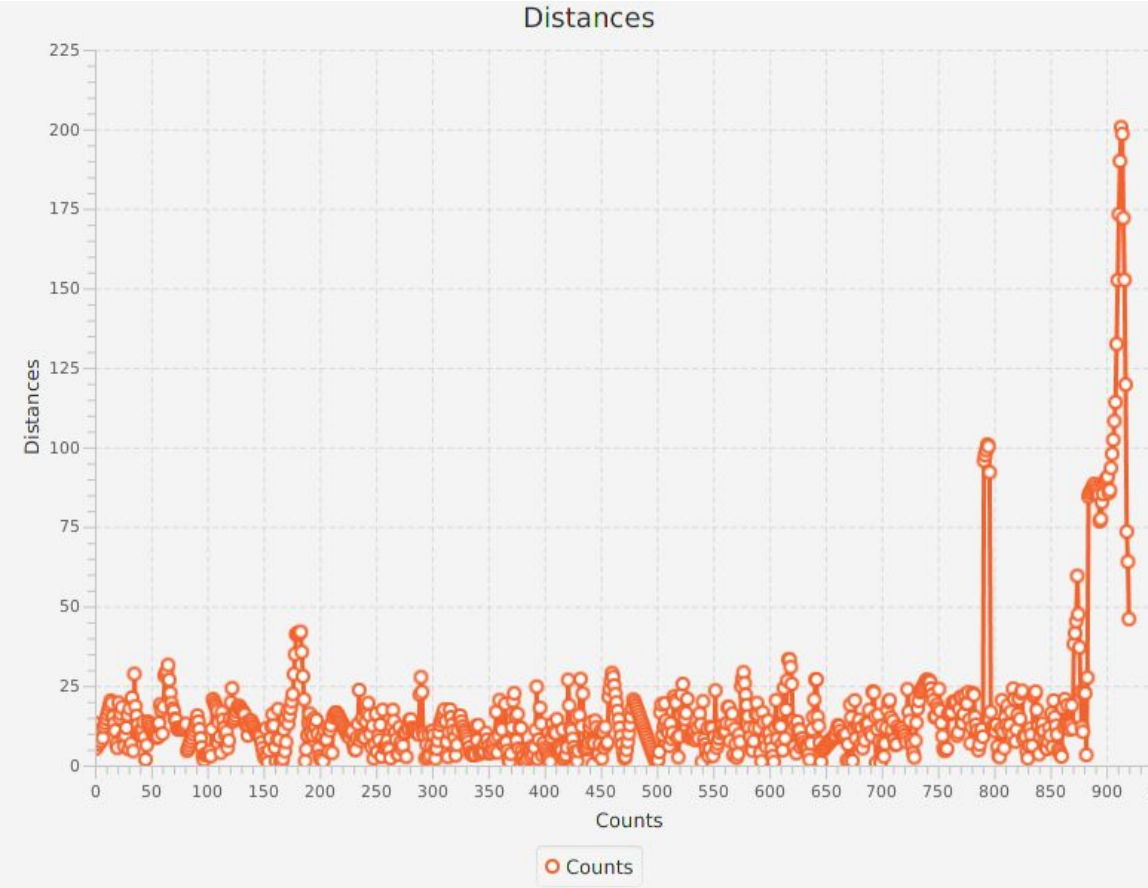


Figure 15. 1D line plot.

Together with Geo visualization, this methods suit good for debugging, inspecting and presenting the results.

5.8 OUTLIER DETECTION DETAILS

With help of JavaML library (implementation from scratch was rejected due to performance low speed and being poorly optimized) there was implemented KMeansClassifier class with K-means and K-medoids classification.

For other statistical methods there is a class named Statistic1DClassifier with Z-score, modified Z-score, IQR anomaly detection methods implementation.

5.9 POINT LOCALIZATION DETAILS

For localization of the problematic points there was created Detector class, which contains the function, accepting the list of classified points as an input, performing the algorithm described in Section 4,3 and returning the list of problematic points.

CHAPTER 6. EVALUATION

6.1 GENERAL METRICS

In the Table 1 there is general statistic about given tracks. There were given 2302 tracks, but after filtering (tracks outside the city, tracks with glitches, etc.) there were only 1685 left to process.

Table 1.

Initial size of dataset	**Processed size of dataset**	**Track length average, km**	**Track length standard deviation, km**	**Distance between points average, m**	**Distance between points standard deviation, m**
2302	1685	8.4	0.14	21	0.15

Statistics of the graph, representing the Prague road network are presented in Table 2.

Table 2.

Area of Prague, km^2 [26]	**Number of nodes in the graph**	**Number of edges in the graph**	**Number of unique processed nodes by algorithm**	**Approximated area covered, %**
496	457744	990734	369202	81

The statistics of graph paths obtained after matching tracks into graph are placed in Table 3.

Table 3.

Distance between interpolated path points average, m	**Distance between interpolated path points standard deviation, m**	**Path length average, m**	**Path length standard deviation, m**
15	0.08	8.4	0.15

6.2 EVALUATION METRICS

The first and the most precise method to control the correctness of map matching is examining it visually. But there are more than 1500 tracks to check and there is numerical solution.

The Modified Dijkstra algorithm for map matching optimizes the area between the track and the path, and for control was used maximum distance between curves. This distance has lack of precision, but works well for approximation and evaluation.

In the Table 4 there are statistics for the maximum distances for all datasets and initial classification for the paths, if they have/ do not have outliers. Paths were classified based on simple Z-score from Section 3.2.3. Paths with maximum distance more than or equal Z-score was classified as “negative”, paths with its maximum distances less than Z-score were classified as “positive”.

Table 4.

Maximum distance average, m	**Maximum distance standard deviation, m**	**Paths without outliers ("positive")**	**Paths with outliers ("negative")**
7.03	3.42	993	692

Paths were also classified by the presence of outliers with all anomaly detection methods. Path was assigned with "positive" class, if there were detected no outliers and "negative", if there was at least one. Knowing the approximate "ground truth" from the Table 4, there tracks can be divided into "true positive" (TP), "true negative" (TN), "false positive" (FP) and "false negative" (FN).

"True positive" are the "positive" paths in both classifications, "true negative" are the "negative" those and "false positive/negative" are misclassified ones [25].

The metrics to evaluate the performance of the algorithms are precision, recall and F1-score.

Precision is a metric revealing the ability of the algorithm to identify only the positive samples. It can be calculated as:

$$precision = \frac{TP}{TP + FP} \tag{16}$$

Recall is the the ability of the algorithm to find all the positive cases within a dataset:

$$recall = \frac{TP}{TP + FN} \tag{17}$$

Precision and recall are unified into single metric F1-score:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{18}$$

[25]

F1-score is the main score used to identify the performance of proposed algorithm.

6.3 TESTING SCENARIOS

The suggested algorithm for graph matching was tested on all the 1685 available and filtered tracks. Results can be classified as “decent” and “problematic”. Typical “decent” result looks like on the figure 16. Tracks are almost identical, the area between the curves is minimal.

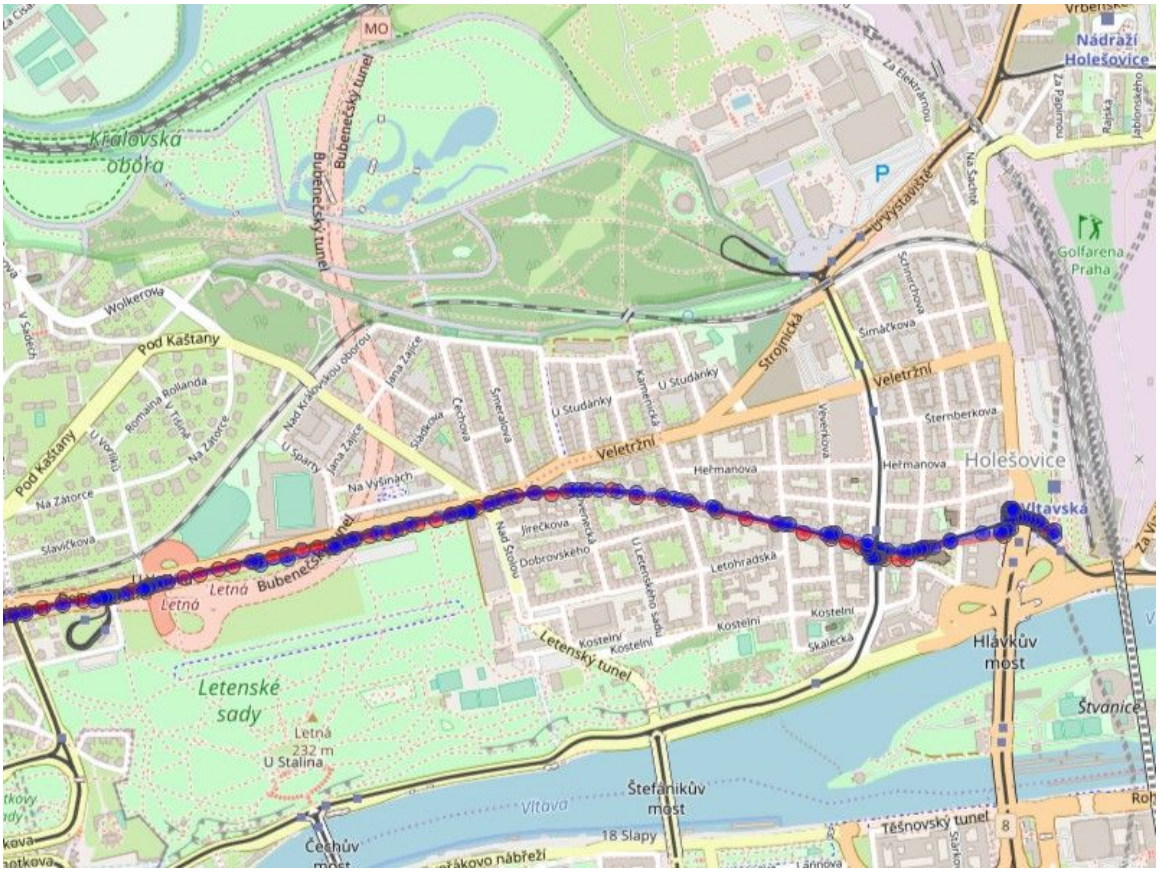


Figure 16. The illustration of situation, when algorithm finds the perfect path.

One of the “Problematical” situations is presented below.

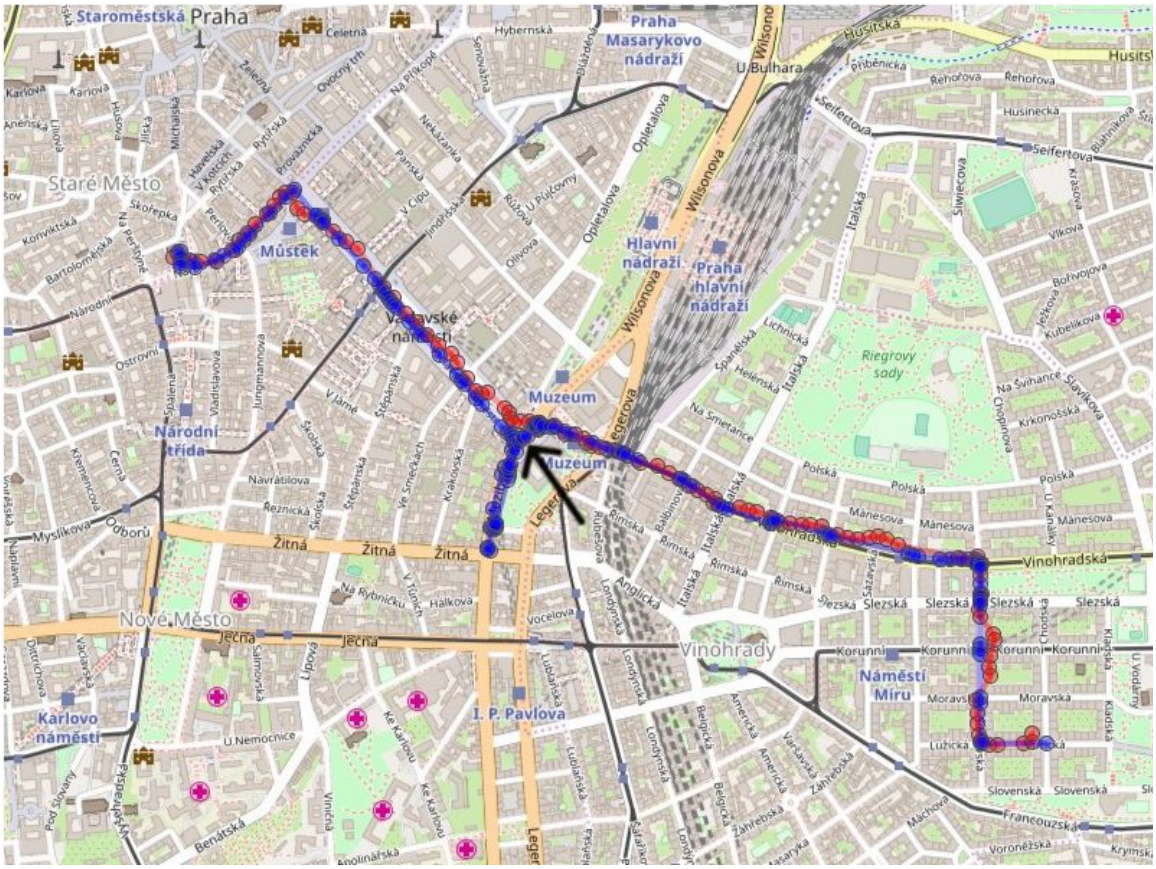


Figure 17. The illustration of suboptimal result.

On the figure 17 the black arrow points on extra turn from Václavské náměstí before it goes back on the track, there is no addition to the graph between 2 curves.

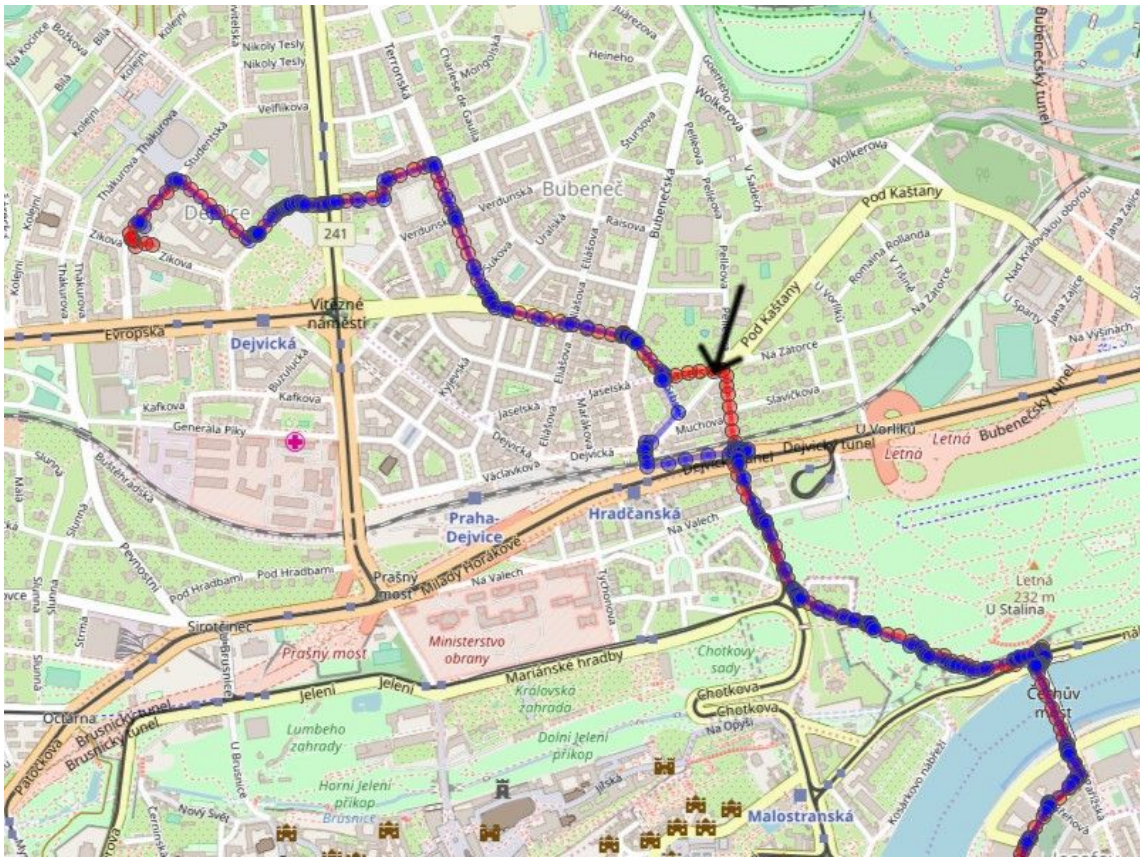


Figure 18. The illustration of suboptimal result, another case.

In the figure 18 the situation is a bit different, there is addition in area, but probably caused by the same errors in the graph.

These three types of scenarios were the input for outlier detection procedure.

6.4 RESULTS

6.4.1 OBJECTIVE FUNCTIONS FOR MODIFIED DIJKSTRA ALGORITHM TESTING

Figure 19. On a) is showed minimum Hausdorff distance objective, on b) the sum of Hausdorff distances, on c) minimum area objective.

There were implemented all the approaches and the third one shows the best results (figure 19). The first approach (a) does not show feasible result at all, the sum of distances is much better (b), the area approach (the last) calculates the path in the best way.

In further testing there were used only results of area objective function.

6.4.2 RESULTS OF OUTLIER DETECTION

There could be two approaches for clustering the collected data, the first one to process each track-path pair separately and then detect the outlier. But experiments on small amount of tracks showed, that there is high false positive rate, because some of tracks do not contain visible outliers and algorithms assign them anyway.

The second approach is to collect all data and detect outliers in all of it and then

match outliers with all tracks. This approach was chosen for further experiments and evaluations.

Each considered combination of statistic and outlier detection approach was evaluated according to the metrics from Section 6.2. The results are presented in the Table 5 and 6. The approach implementing the interpolated statistic + robust Z-score outlier detection has the highest F1 score and it is presented in final pipeline, discussed in Section 4.1.

Table 5.

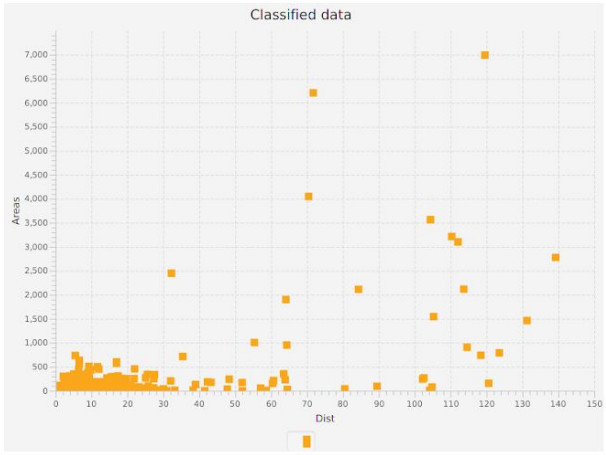
	Number of "positive" paths	**Number of "negative" paths**	**TP**	**TN**	**FP**	**FN**
K-means, k = 16	1413	272	991	270	422	2
Z-score	1643	42	993	42	650	0
Robust Z-score on interpolated statistics	889	796	883	686	6	110
Robust Z-score on inverse statistics	816	869	816	692	0	177
IQR on interpolated statistics	887	798	880	685	7	113
IQR on inverse statistics	810	875	796	678	14	197

Table 6.

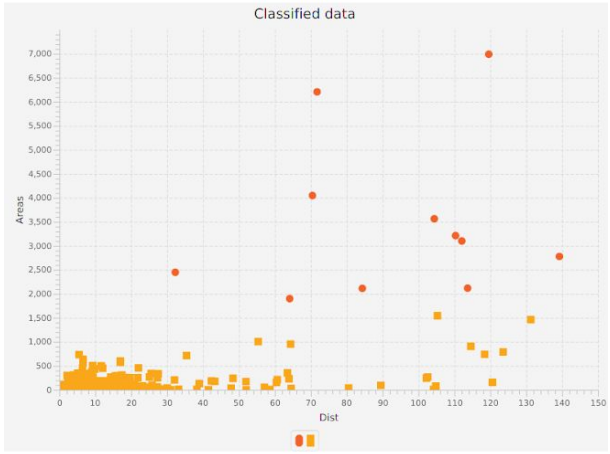
	Precision, %	**Recall, %**	**F1-score, %**
K-means, k = 16	70	99	82

Z-score	60	100	75
Robust Z-score on interpolated statistics	99	88	93
Robust Z-score on inverse statistics	100	82	90
IQR on interpolated statistics	99	87	92
IQR on inverse statistics	98	80	88

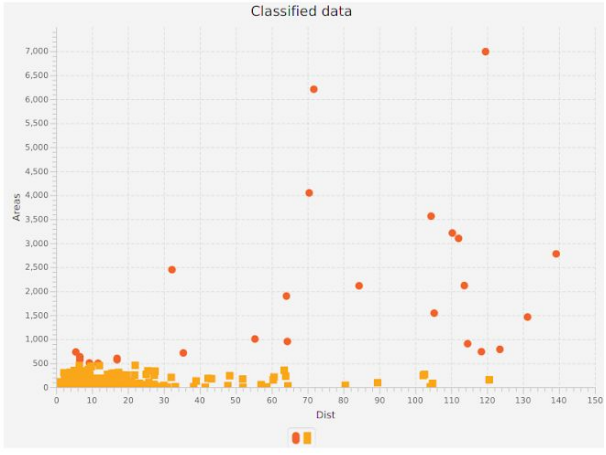
The area-distance measurements were clustered with K-means and K-medoids (figure 20), there is no need to show two graphs, as they are the same and both algorithms did not manage to handle the outliers, as it was necessary to increase number of clusters significantly, which raises processing time and also should be optimized by unknown criteria. [15]



a



b



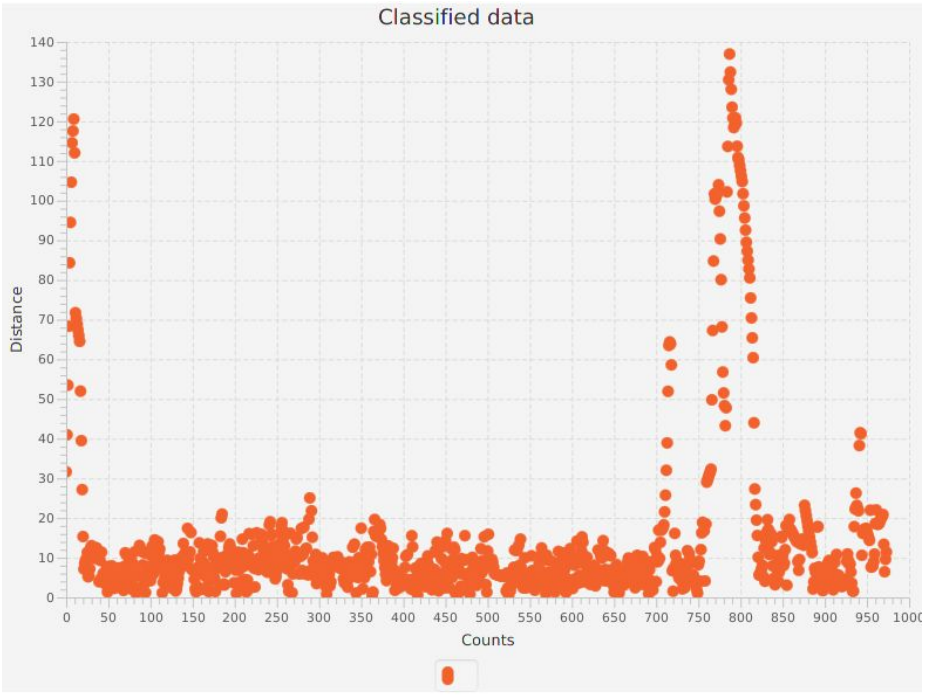
c

Figure 20. a. K-means clusters, k=4, computational time 5.7 seconds; b. K-means clusters, k=16, computational time 4 minutes; c. K-means clusters, k=32, computational time 7 minutes approximately. Yellow dots belong to the biggest class, red dots belong to

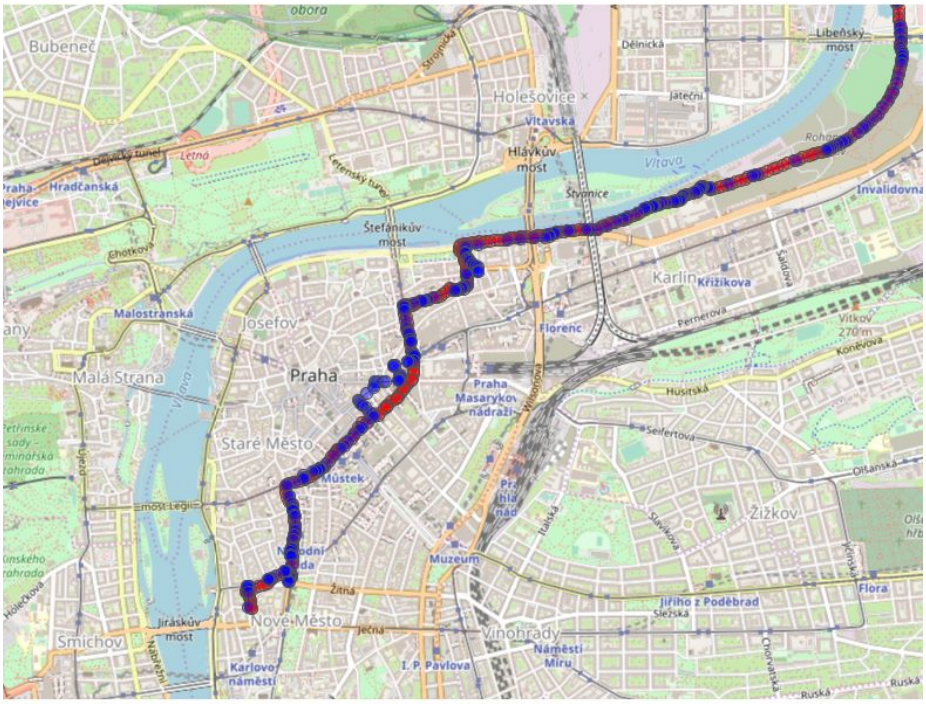
the other ones.

For inverse distance and interpolated distance statistics were applied Z-score, modified Z-score and IQR.

For both inverse and interpolated distances Z-score totally failed (figure 21). Mean-based methods perform badly with significant amount of outliers and from the beginning the fraction of them was unknown. It explains the mean drift towards the bigger values.



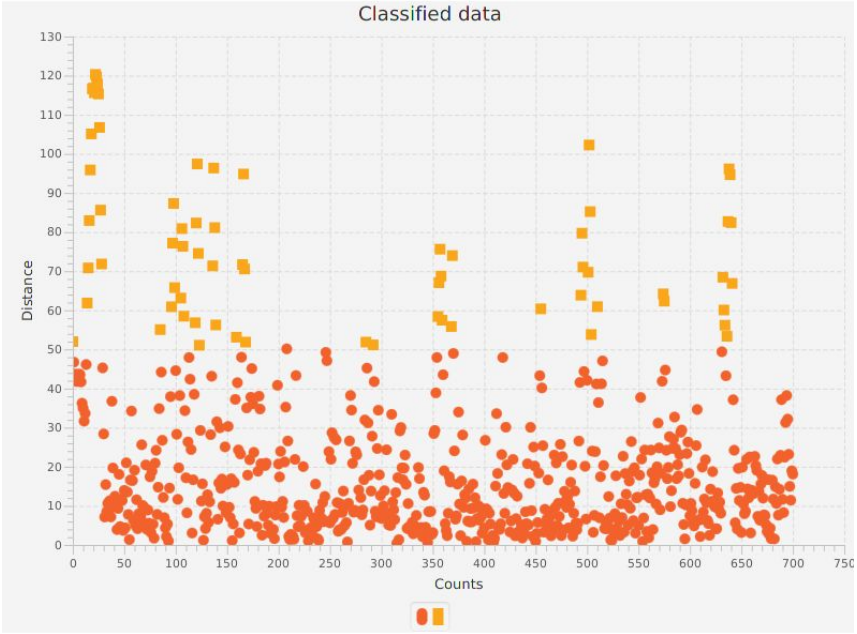
a



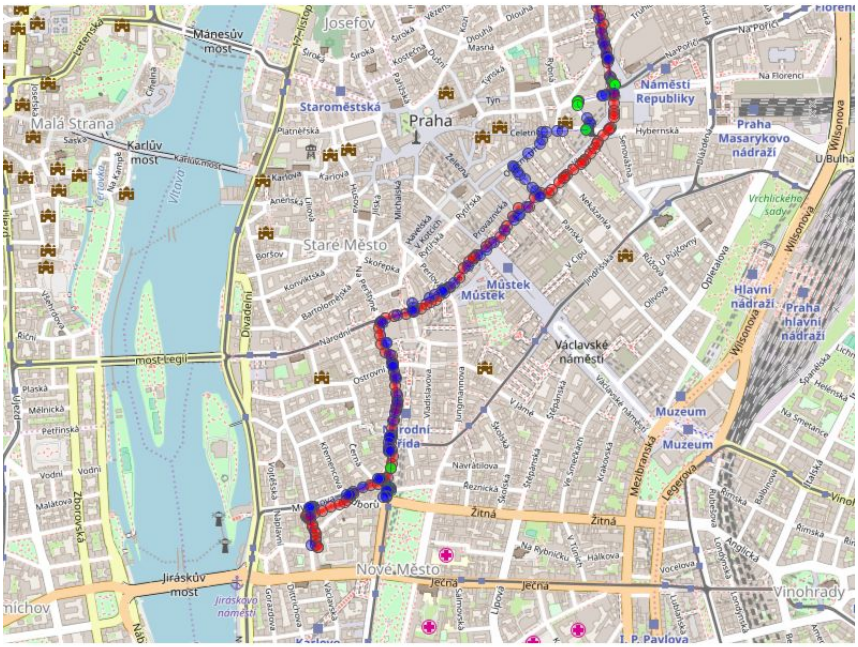
b

Figure 21. a. Z-score outlier detection distances plot; b. Z-score outlier detection classified track on map.

Modified Z-score and IQR performed slightly better on inverse distance measurements (figures 22 and 23) and significantly better on interpolated distances (figures 24 and 25).



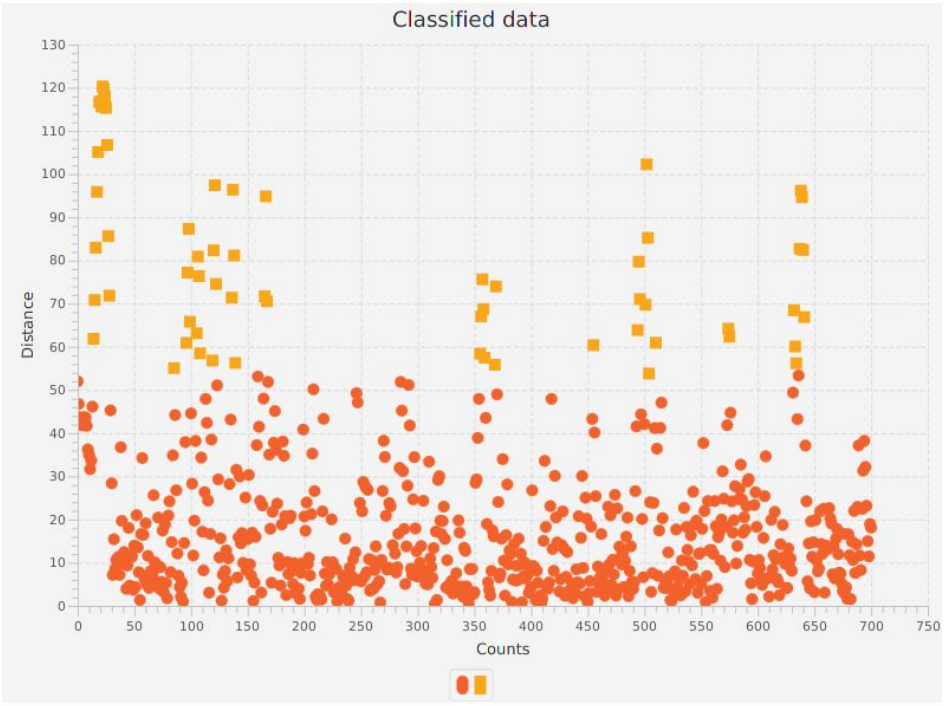
a



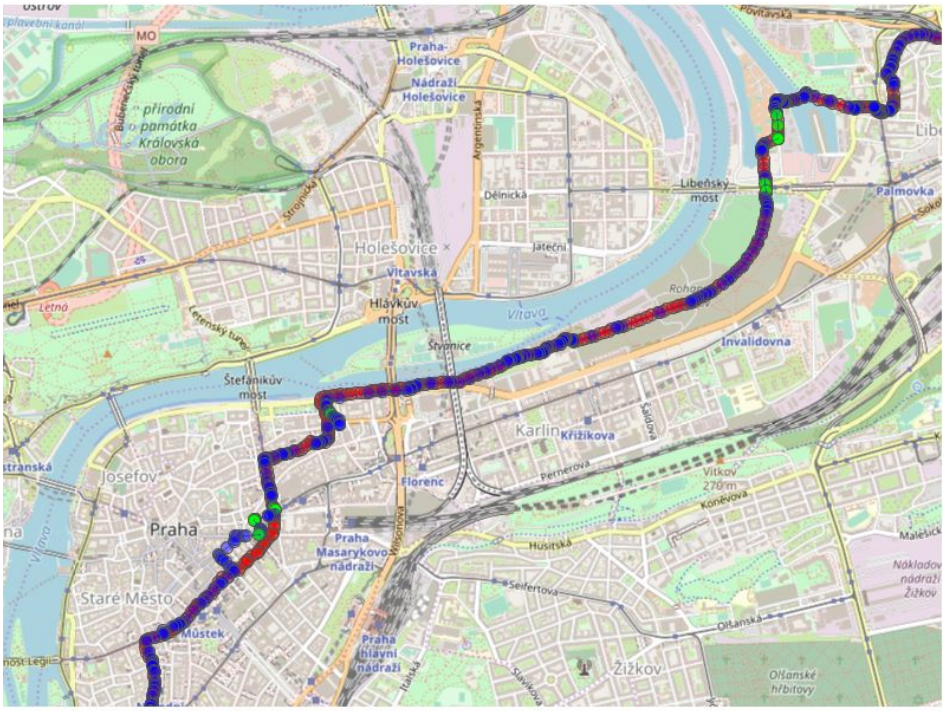
b

Figure 22. Robust Z-score outlier detection on inverted distance statistic: a. classified

distances distribution; b. classified track on map.



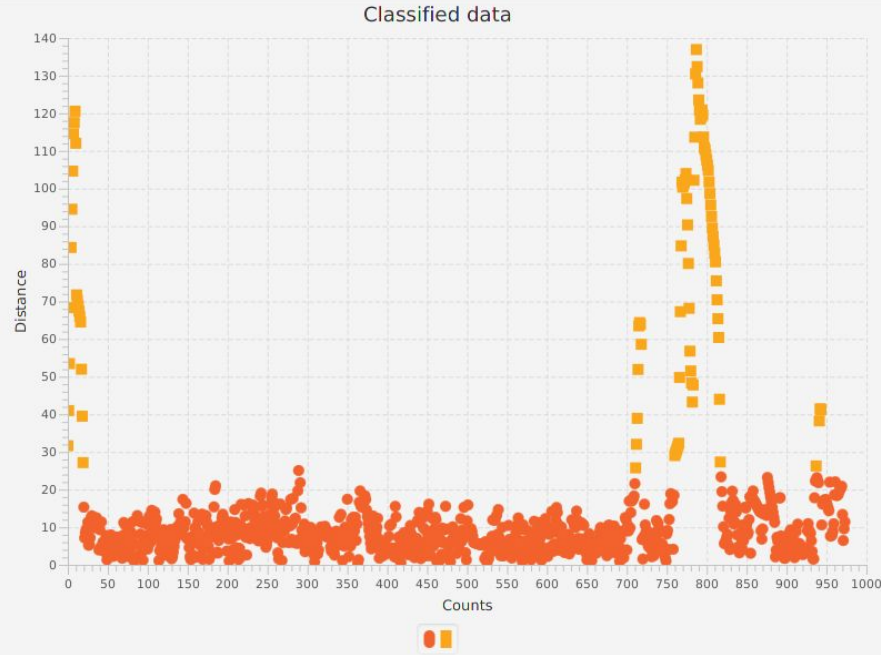
a



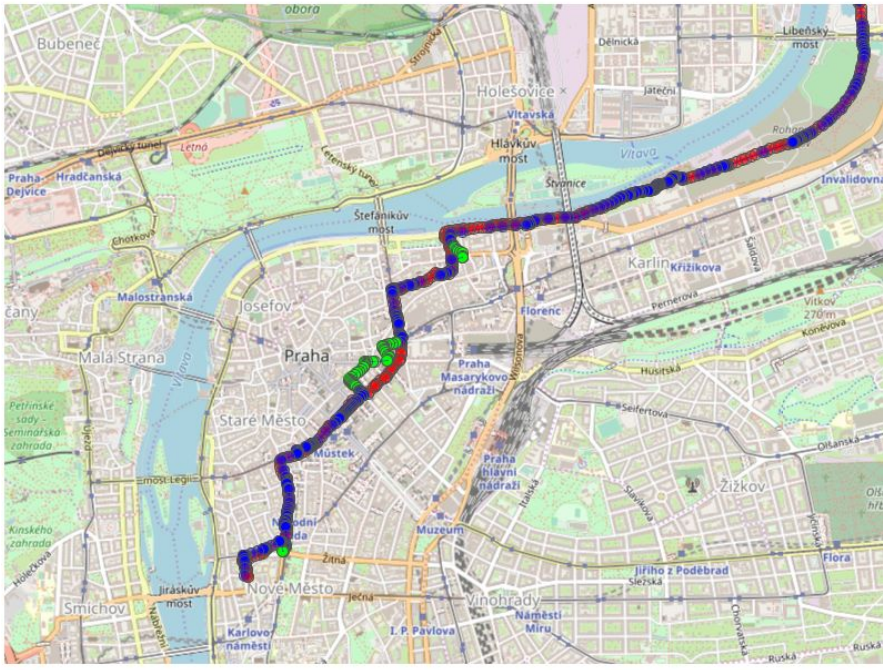
b

Figure 23. IQR outlier detection on inverted distance statistic: a. classified distances

distribution; b. classified track on map.

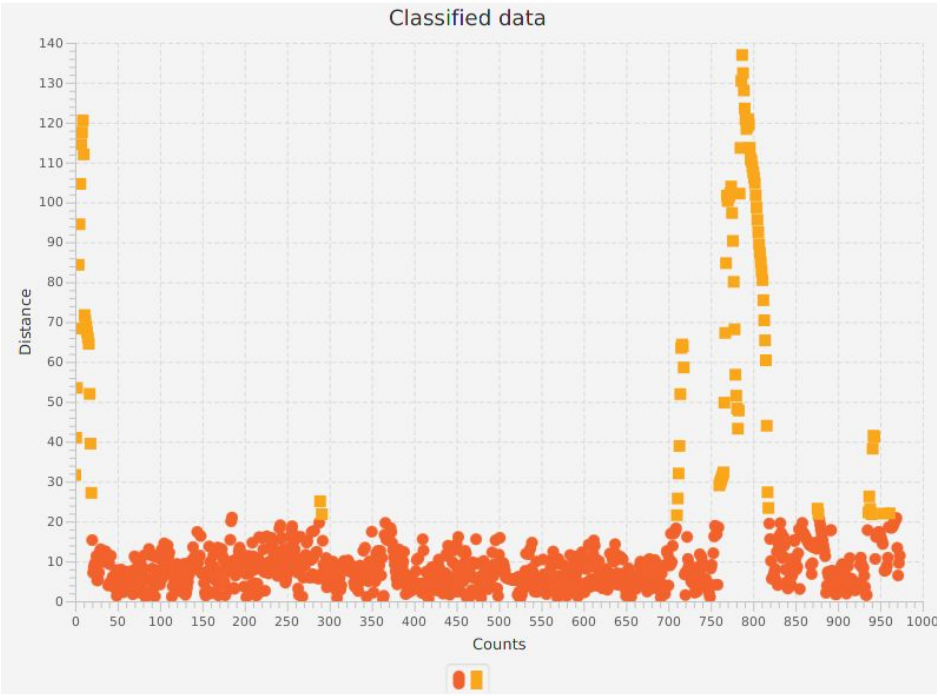


a

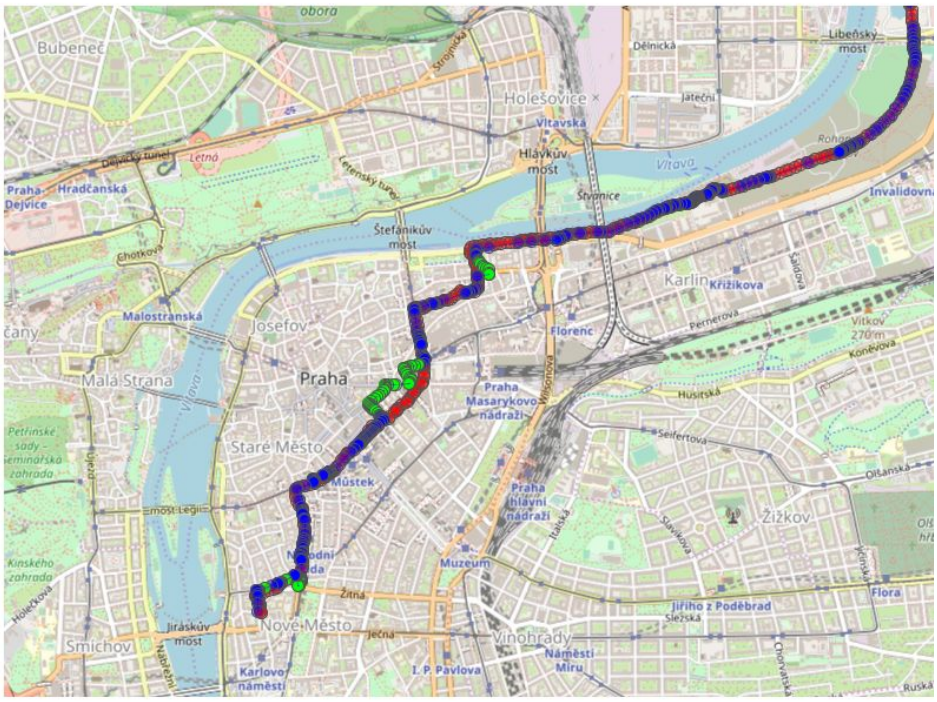


b

Figure 24. Robust Z-score outlier detection on interpolated distance statistic: a. classified distances distribution; b. classified track on map.



a



b

Figure 25. IQR outlier detection on interpolated distance statistic: a. classified distances distribution; b. classified track on map.

By default the threshold for modified Z-score was chosen as 3.5 as recommended in [17], but experiments showed, that 3.8 worked better, as it does not produced occlusions.

For further processing only interpolated data with modified Z-score outliers was picked.

6.4.3 PROBLEMATIC POINT LOCALIZATION RESULTS

This phase gives the second approximation to desired points. Results are presented in figure 26.

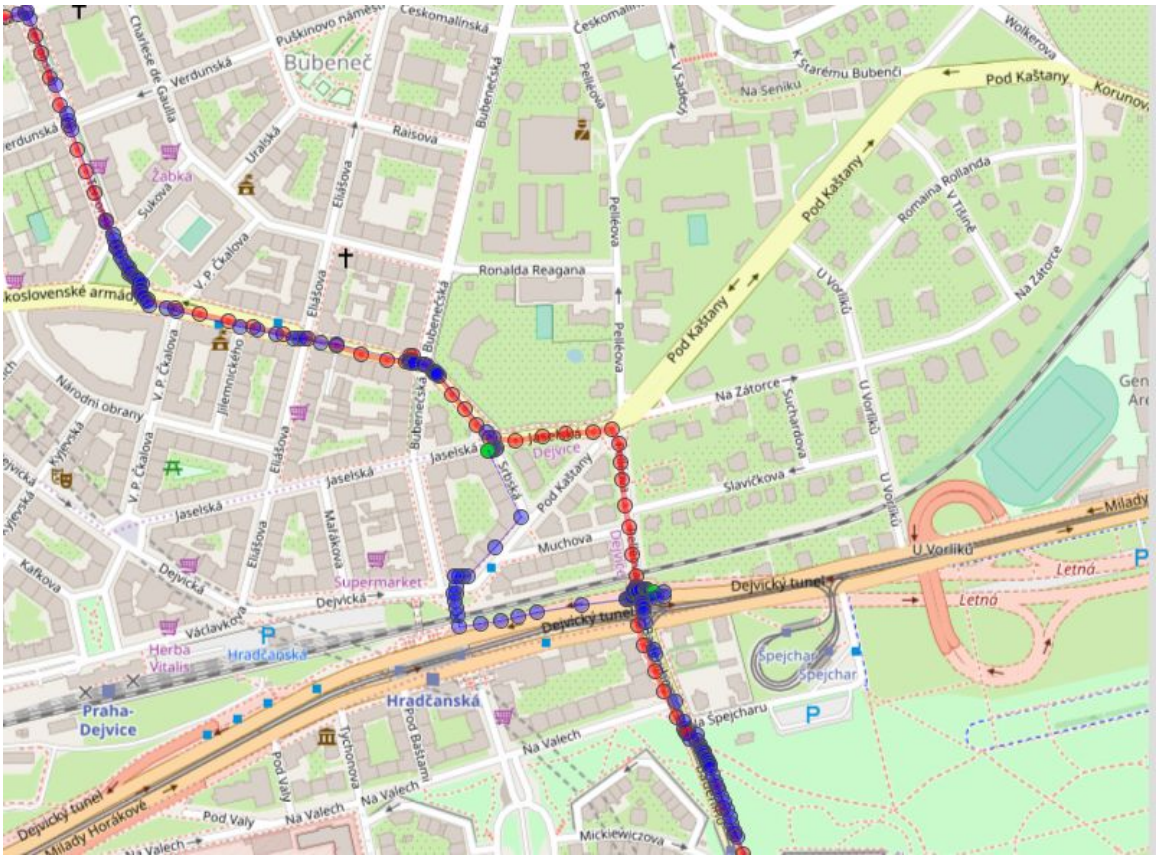


Figure 26. Result of the 1st phase point detection, red line is an input track, blue line is an output track, green dots are detected points.

6.4.4 GRAPH REFINEMENT DEMONSTRATION

In Section 4.4 there was presented the description of naive and slow algorithm to refine the graph after localizing the approximation of the point with missing edge. The demonstration of the result is presented in Figure 27.

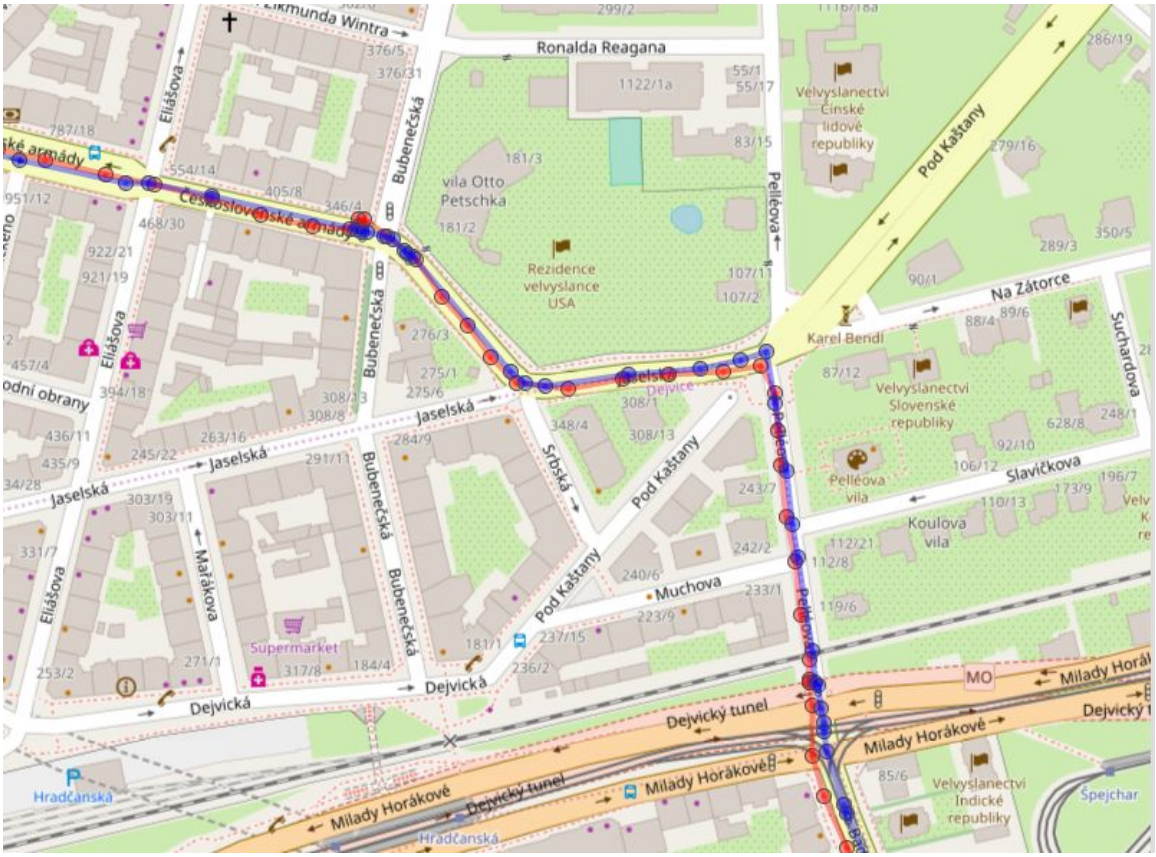


Figure 27. Demonstration of map matching on refined graph.

CHAPTER 7. CONCLUSION

7.1 DISCUSSION

Suggested algorithm matches the GPS tracks to the road graph, collects statistics, classifies points to inliers and outliers based on them and detects the node, where the missing edge can start.

Described algorithm is a powerful tool to evaluate the road network and automatically produce possible places of mislabelling in the map. Due to offline nature of the algorithm the speed of its performance is not prior, as the quality of proposed result.

7.1.1 APPROACH DRAWBACKS

Presented approach is slow on significant data amount. The modified Dijkstra algorithm can be further optimized.

Point detection procedure performs suboptimally as well. The algorithms has limitations on routes with loops-like sections.

7.1.2 APPROACH ADVANTAGES

On pre-classified data (new track for example) algorithms work relatively fast with decent precision. Suggested by outlier detection thresholds cover all tracks, additional fitting to individual track is not needed.

7.2 FUTURE WORK

The pipeline could be further polished and optimized for faster convergence and generalization for certain track types with loops. The proof of the algorithm's accurate work will be improved as well.

REFERENCES

1. Newson, P., Krumm, J.: Hidden Markov map matching through noise and sparseness. Microsoft Research (2009).
2. Brakatsoulas, S., Pfoser, D., Salas, R., Wenk, C.: On map-matching vehicle tracking data. The 31st VLDB Conference, Trondheim, Norway (2005).
3. Greenfeld, J.: Matching GPS observations to locations on a digital map. In 81th annual meeting of the transportation research board (Vol. 1, No. 3, pp. 164-173).
4. Axhausen, K.W.: Map-matching of large GPS data sets - Tests on a speed monitoring experiment in Zurich. Researchgate, 2014.
5. Knuth, D.E.: "A Generalization of Dijkstra's Algorithm". Information Processing Letters (1977).
6. Singhal, N.: Shape matching and structural comparison. Handout for CS273: Algorithms for Structure and Motion in Biology, Stanford University (2004).
7. Wylie, T., Binhai Zhu: Following a curve with the discrete Fréchet distance. Theoretical Computer Science 556 (2014) 34–44.
8. Huabei Yin, Wolfson O.: A Weight-based Map Matching Method in Moving Objects Databases. SSDBM '04 Proceedings of the 16th International Conference on Scientific and Statistical Database Management (2004).
9. Anil K. Jain: Data clustering: 50 years beyond K-means. Pattern Recognition Letters 31 (2010) 651–666.
10. Drineas, P., Frieze, A., Kannan, R., Vempala, S., Vinay, V., 1999. Clustering large graphs via the singular value decomposition. Machine Learn. 56 (1–3),

9–33.

11. Meila, Marina, 2003. Comparing clusterings by the variation of information. In: COLT, pp. 173–187.
12. Jain, Anil K., Dubes, Richard C., 1988. Algorithms for Clustering Data. Prentice Hall.
13. Mao, J., Jain, A.K., 1996. A self-organizing network for hyper-ellipsoidal clustering (HEC). IEEE Trans. Neural Networks 7 (January), 16–29.
14. Kaufman K., Rousseeuw P. J.: Clustering by means of medoids. Faculty of Mathematics and Informatics, Delft (1987).
15. Kyung-A Yoon, Oh-Sung Kwon, Doo-Hwan Bae: An Approach to Outlier Detection of Software Measurement Data using the K-means Clustering Method. First International Symposium on Empirical Software Engineering and Measurement (2007).
16. Shiffler R.E.: Maximum Z Scores and Outliers. The American Statistician, Vol. 42, No. 1. (Feb., 1988), pp. 79-80.
17. Iglewicz B., Hoaglin D.: (1993), How to Detect and Handle Outliers. The ASQC Basic References in Quality Control: Statistical Techniques, Volume 16 (1993).
18. Upton G, Cook I.: Understanding Statistics Oxford University Press. ISBN 0-19-914391-9 p. 55 (1996)/
19. Zwillinger, D., Kokoska, S.: CRC Standard Probability and Statistics Tables and Formulae, CRC Press. ISBN 1-58488-059-7 p. 18 (2000).
20. Interquartile range. In Wikipedia. Retrieved December 20, 2018, from https://en.wikipedia.org/wiki/Interquartile_range.
21. Standard Score. In Wikipedia. Retrieved December 20, 2018, from https://en.wikipedia.org/wiki/Standard_score.
22. Zimek, A., Schubert, E. : "Outlier Detection", Encyclopedia of Database Systems, Springer New York, pp. 1–5, doi:10.1007/978-1-4899-7993-3_80719-1, ISBN 9781489979933 (2017).
23. Chandola, V., Banerjee, A., Kumar, V. :Anomaly detection: A survey. ACM

Computing Surveys. 41 (3): 1–58. doi:10.1145/1541880.1541882 (2009).

24. ECMT Safety in road traffic for vulnerable users Organisation for Economic Co-operation and Development OECD, Paris (2000).
25. Powers, David M. W.: Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. Journal of Machine Learning Technologies. 2 (1): 37–63 (2011).
26. Prague. In Wikipedia. Retrieved December 20, 2018, from https://en.wikipedia.org/wiki/Prague.