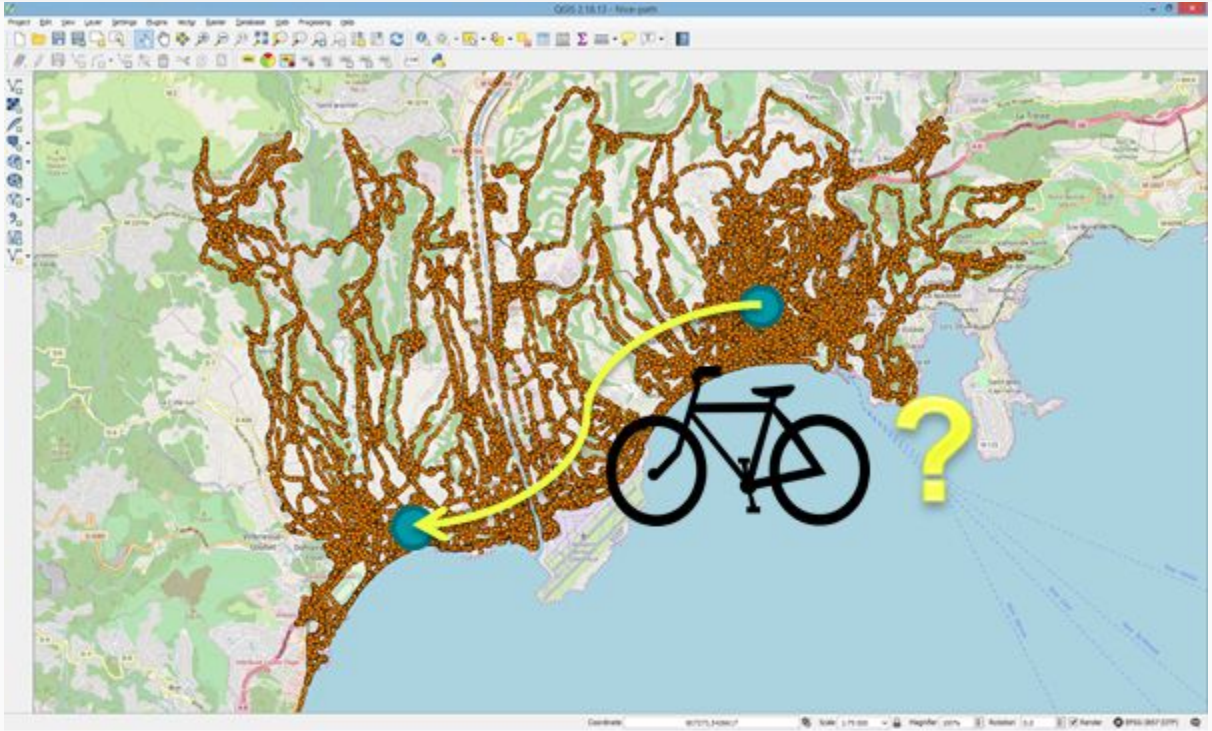
# Routing in Multimodal Networks With Bicycles

Student:
Mykhailo Zima

Supervisors :
David Coudert, Nicolas Nisse

# Motivation



- Bicycles - An increasingly popular means of transport.
- Need to develop an algorithm that finds an A to B optimal path.
- Cyclists' paths preferences depend not only on distance but on a lot of other path features (slopes, traffic etc)

# Personalized route planner for bicycles

- Allows users to navigate road networks optimally.
- Based on individual driving styles as well as personal preferences.
- Takes as input
  - 1) A road network $G = (V, E)$ and a set of cost functions $c_1, c_2, \ldots c_r$ with $c_i : E \rightarrow [0,\infty)$ for every metric i.
  - 2) A starting point
  - 3) A destination
  - 4) A set of weights $(w_1, w_2, \ldots, w_r)$ that determine which metrics the optimal path should be computed based upon.
  - 5) A set of parameters that determine some cyclist's individual riding features.
- Produces as output :
  - A set of vertices that corresponds to the path with minimal weighted cost

# State of the art - Graph Compression and Dijkstra Variations

1. Direct continuation of N. Vadakke-Palangatt and M. Zima PFE work
2. Graph compression during preprocessing : A very popular approach (18.0 million vertices and 42.5 million edges : Memory, preprocessing time & Query time). Dijkstra - 0.4 Gb, -, 2.2 s.
   a. Hub-labeling - 18.8 Gb, 0:37 h, 0.56 µs [D. Delling, A. Goldberg, R. Werneck, 2013]
   b. Contraction Hierarchy - 0.4 Gb, 0:05 h, 110 µs [Robert Geisberger, Peter Sanders, 2012]
   c. Customizable Route Planning - 0.9 Gb, 1:00 h, 1650 µs [D. Delling, A. Goldberg, T. Pajor, R. Werneck, 2014]
   d. Pruned Landmark Labeling [T. Akiba, Y. Iwata, Y. Yoshida, 2013]
3. Dijkstra speedups : Bidirectional Dijkstra, Heuristic Based Dijkstra (A-star)
4. Tools used: a) OpenStreetMap b) QGIS c) Sagemath

# Contribution during the internship

## Objectives:

- Find solution for optimal path search that takes into account features specific for cyclists;
- Create a real-world graph that contains these features;
- Make these features balanced in comparison to each other;
- Find different optimal paths for different users according to their preferences.

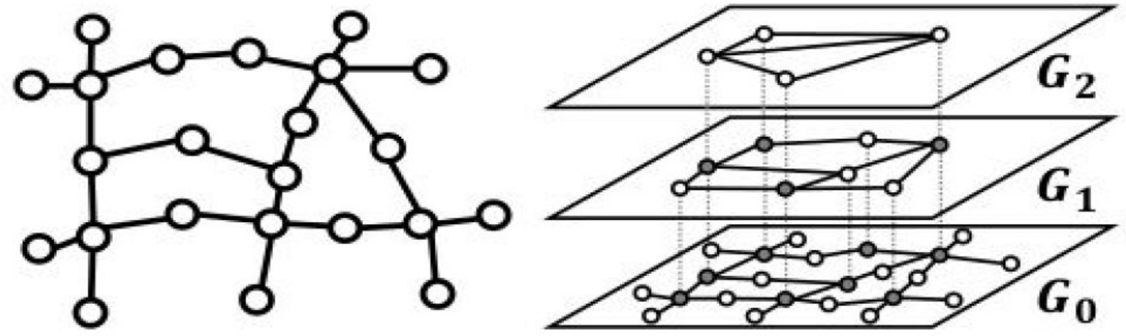
## End result:

Working implementation of the algorithm on Nice’s graph.

# The approach to Graph Compression : K-path Covers (Funke et al, 2014) & Cover Hierarchy (Akiba et al, 2016)

**k-Path Cover**: In Graph G(V, E), a set C ⊆ V such that C ∩ P ≠ ∅ for any path P of length k.

- Minimum k-path Cover : A NP-Hard problem
- **k-all-path-cover hierarchy**: Based of vertex-covers (Akiba et al, 2016)
- Idea: Nth layers of vertex cover is the $2^N$-path cover of the original graph.




(a) Original graph $G_0$ (b) Layers of 2-APCs

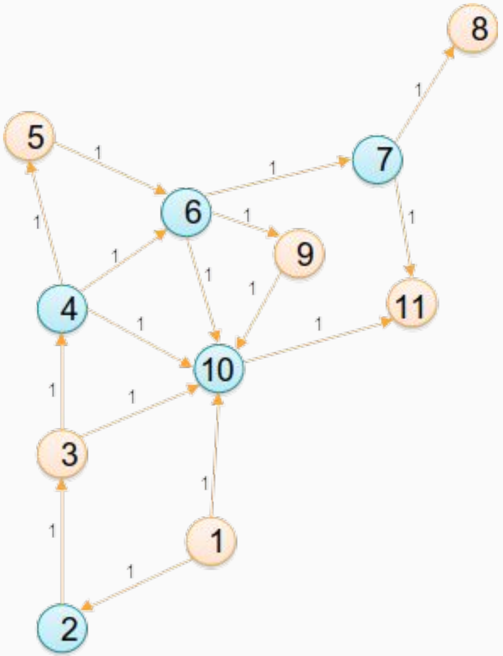
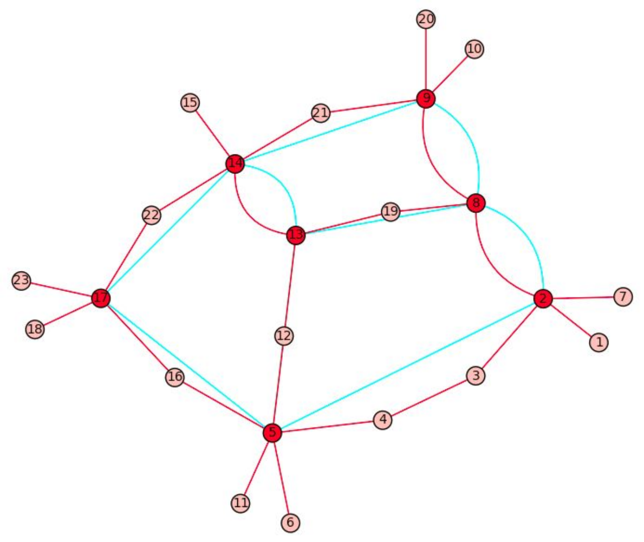
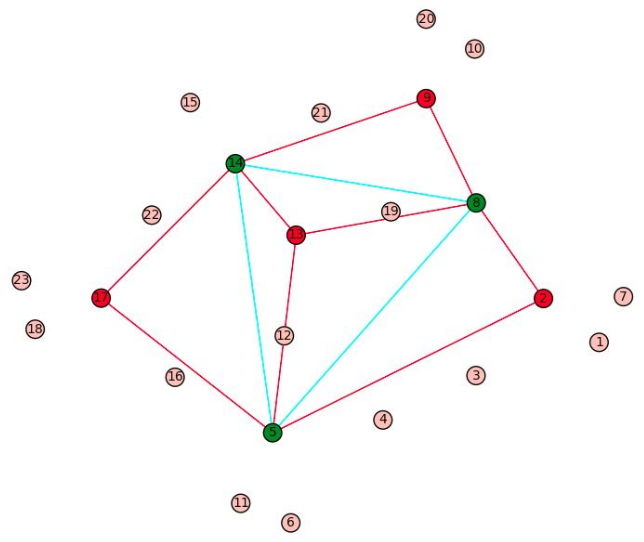



Fig : A 2-KPC Example(K-path cover in blue)

# Algorithm modification during internship




Densifying the graph

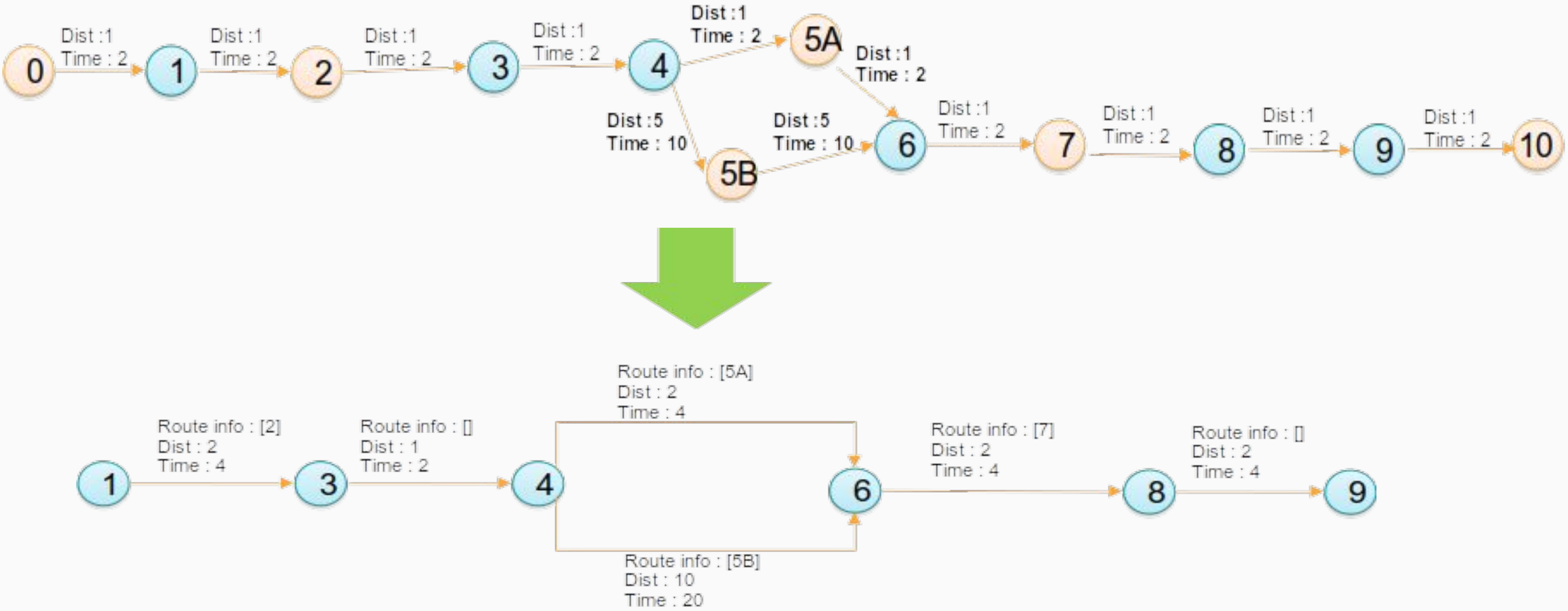



A vertex cover on densified graph

- Reason: to make the algorithm less dependent on graph topology;
- Before creating overlay layers leave as access nodes only those vertices which have more than 2 neighbours.
- Proved to be efficient.

# The approach to Graph Compression : Overlay Graphs

- Maintain all the routes possible among the compressed vertices.
- Route info: To relate edge to corresponding route in original graph.
- Cost info (dist, time etc): Single lookup retrieval of the route cost.



# Client - server socket system

## Server:

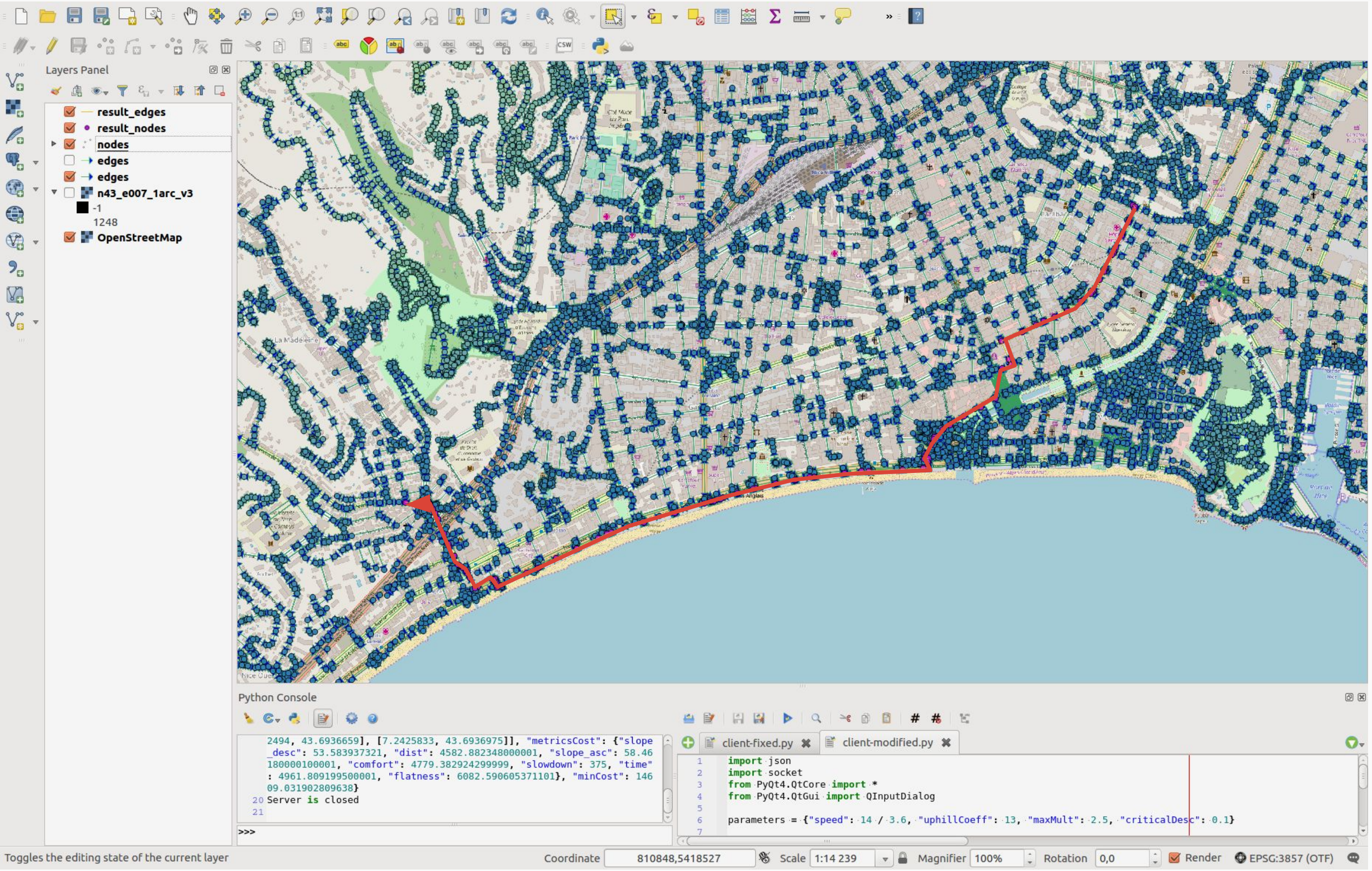
- operates in sage;
- contains precomputed overlay graph;
- receives queries from clients, processes them and sends responses back.
- responses contain lists of vertices that correspond to the optimal path and cost of paths

## Client:

- operates in QGIS;
- contains full graph;
- sends queries which contain:
  - source node;
  - destination node;
  - user weights;
  - parameters.
- receives responses and presents them to users.

Client and server communicate with each other using socket system

# Client part




- User interface for simple source and destination point, user weights selection;
- The result is presented as a line with highlighted nodes.

# Retrieving the graph

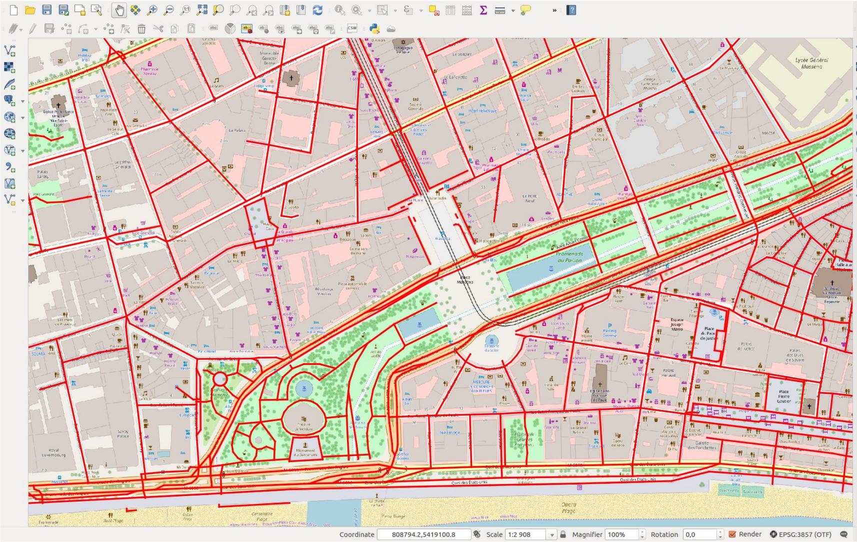
- Data downloaded from OpenStreetMap.
- Presented as Shapefile (.shp)
- Afterwards converted to Sage object (.sobj)
- Steps to retrieve a working graph:
  1) Convert multilines (lines with intermediate points) to edges;
  2) Make the graph directed - according to ‘oneway’ tag;
  3) Make the graph strongly connected;
  4) Add cliques to squares - they are denoted with ‘place=square’ tag.

| | osm_id | name | highway | waterway | aerialway | barrier | man_made | other_tags |
|---|---|---|---|---|---|---|---|---|
| 1 | 164192418 | | | | | | | "boundary"=>"political","political_division"=>"canton" |
| 2 | 135463949 | | pedestrian | | | | | |
| 3 | 243900477 | | footway | | | | | "bicycle"=>"dismount","wheelchair"=>"yes" |
| 4 | 560091690 | | pedestrian | | | | | "tunnel"=>"building_passage" |
| 5 | 4246694 | Descente Crotti | residential | | | | | "surface"=>"asphalt" |
| 6 | 560091691 | | service | | | | | "oneway"=>"yes" |
| 7 | 4246695 | Descente du Marché | steps | | | | | "reg_name"=>"calada dóu Mercat" |
| 8 | 560091692 | | pedestrian | | | | | "tunnel"=>"building_passage" |
| 9 | 105040811 | | primary_link | | | | | "oneway"=>"yes","surface"=>"asphalt" |
| 10 | 243591053 | | footway | | | | | "bicycle"=>"dismount","wheelchair"=>"yes" |
| 11 | 560091693 | | pedestrian | | | | | |
| 12 | 243591054 | | footway | | | | | "bicycle"=>"dismount","wheelchair"=>"yes" |
| 13 | 560091694 | | pedestrian | | | | | "tunnel"=>"building_passage" |
| 14 | 560091695 | | pedestrian | | | | | |
| 15 | 243591056 | | footway | | | | | "bicycle"=>"dismount","wheelchair"=>"yes" |
| 16 | 560091696 | | pedestrian | | | | | |
| 17 | 560091697 | | pedestrian | | | | | "tunnel"=>"building_passage" |
| 18 | 560091698 | | pedestrian | | | | | "tunnel"=>"building_passage" |
| 19 | 560091699 | | pedestrian | | | | | |
| 20 | 243591060 | Allée Albert Camus | footway | | | | | "bicycle"=>"dismount","lit"=>"yes","wheelchair"=>"yes" |
| 21 | 137418706 | | | | | | | "height"=>"25" |
| 22 | 560091700 | | pedestrian | | | | | |

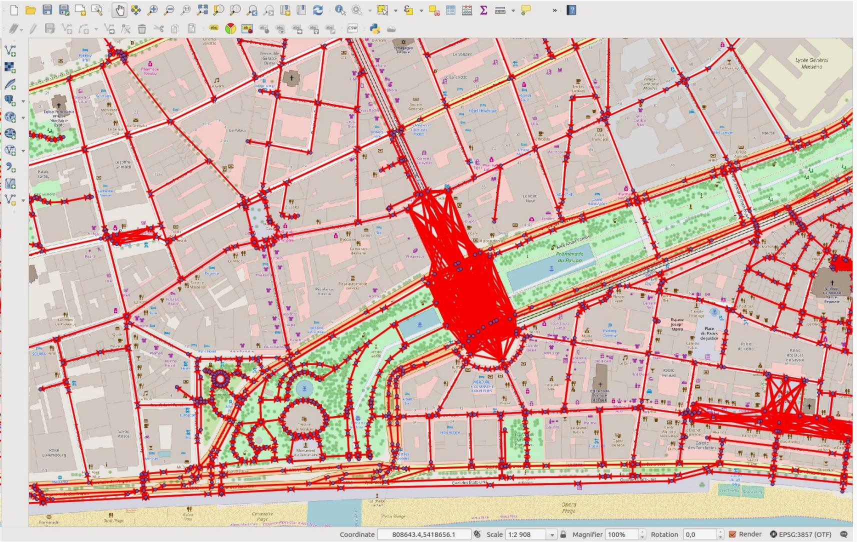
Show All Features

Example of OpenStreetMap tags in edges represented in QGIS

# Retrieving the graph



The initial graph



The processed graph

# User’s input during query

The user has to choose values for these metrics:

- Travel time [0..1] - how fast a user can reach destination;
- Comfort [0..1] - how comfortable is user’s ride;
- Flatness [0..1] - how many slopes will the route contain.

and these parameters:

- Speed (m/s);
- Uphill penalty - how much uphill ride slows down the user;
- Downhill speed multiplier - maximum value of downhill speed;
- Critical downhill grade - value when maximum downhill speed is achieved.

More detailed information in appendix.

# Path features which affect cyclist’s choice

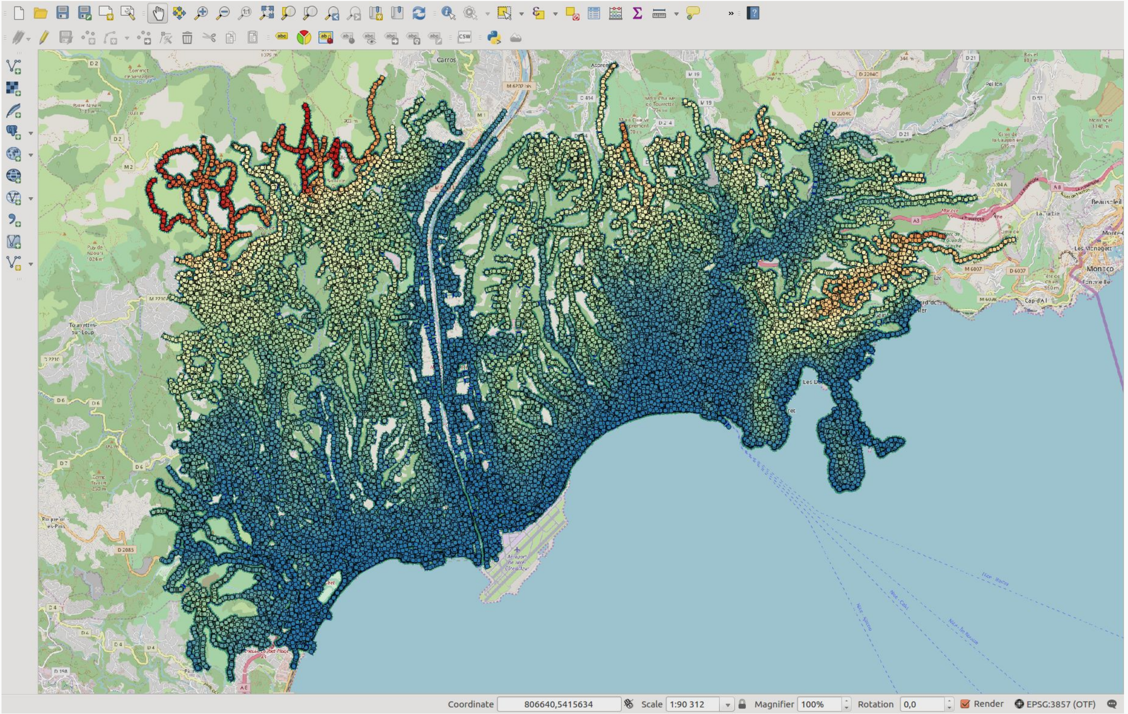
| Feature | Affects | Description | Tags in OpenStreetMap |
|---|---|---|---|
| Distance | time, comfort, flatness | length of the edge in m | |
| Slope | time, flatness | relation of vertices height difference and distance | |
| Surface | time, comfort | type of surface which affects speed and comfort (asphalt, cobblestone, gravel etc) | smoothness, surface, tracktype |
| Highway | comfort | type of road (cycleway, primary, residential, pedestrian etc) | highway, bicycle, cycleway |
| Slowdown | time | obstacles that make the cyclist stop (crossings, traffic signals, steps etc) | crossing, highway |

# Elevation data

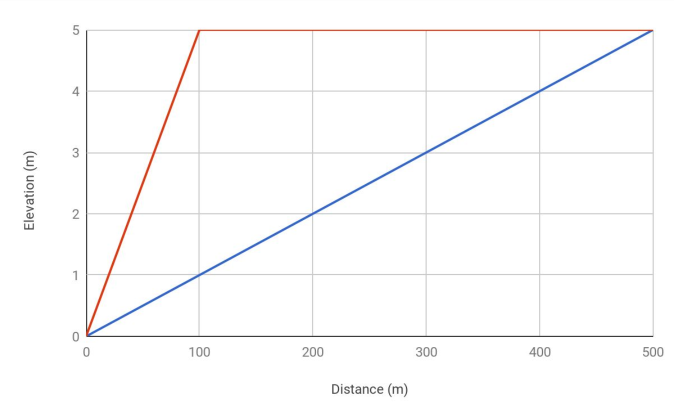
- Elevation data absent from OpenStreetMap;
- Used SRTM 1 Arc-Second Global from EarthExplorer;
- Every edge given slope value by this formula:

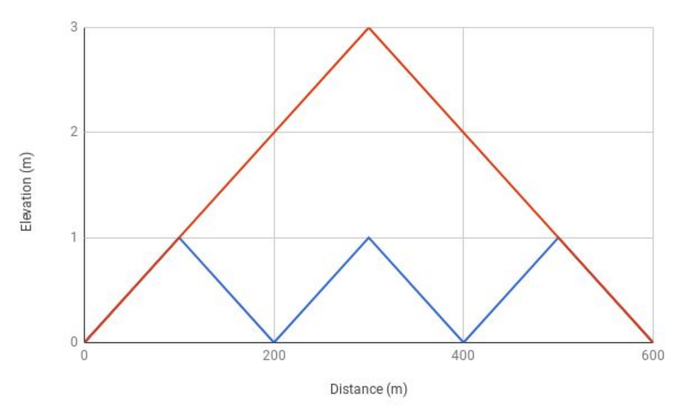
$$slope(u, v) = \frac{height(v) - height(u)}{distance(u, v)}$$

- Positive if uphill, negative if downhill.



# Slopes metric problems




Gentle slopes are easier for cyclists even if they are longer.

Solution: Flatness coefficient which polynomially depends on gradient value. (Crispin H.V. Cooper, 2016)

Less continuous slopes are easier for cyclists.

Solution: Currently an open question.

# Analysis of the algorithm's performance on the map of Nice

| Overlay layer | k | Constr. time (s) | # vertices | # edges | D avg | D max | Dijkstra (ms) | Search (ms) | Speed up |
|---|---|---|---|---|---|---|---|---|---|
| Initial | - | - | 100768 | 200155 | 3.97 | 56 | - | - | - |
| 0 | 1 | 12.9 | 17513 | 43276 | 4.94 | 56 | 873 | 421 | 2.07 |
| 1 | 2 | 16.8 | 9356 | 35529 | 7.59 | 146 | 873 | 322 | 2.71 |
| 2 | 4 | 20.4 | 5800 | 36819 | 12.7 | 304 | 873 | 276 | 3.16 |
| 3 | 8 | 24.2 | 3899 | 51293 | 26.31 | 1183 | 873 | **240** | 3.64 |
| 4 | 16 | 38.3 | 2830 | 137394 | 97.1 | 9081 | 873 | 280 | 3.12 |
| 5 | 32 | 287 | 2190 | 694848 | 634.56 | 49508 | 873 | 614 | 1.42 |

- Overlay layer #3 has the best performance for the graph of Nice with 240 ms of search;
- Compare it to 358 ms by the previous version of the algorithm.

Test Machine: Linux machine with 2.10 GHz Intel(R) Core(TM) i3-2310M CPU and 4GB of memory.

# Analysis of the algorithm’s performance on the map of New York City

| Overlay layer | k | Constr. time (s) | # vertices | # edges | D avg | D max | Dijkstra (ms) | Search (ms) | Speed up |
|---|---|---|---|---|---|---|---|---|---|
| Initial | - | - | 240474 | 431371 | 3.59 | 12 | - | - | - |
| 0 | 1 | 27.9 | 66435 | 167805 | 5.05 | 12 | 2226 | 1464 | 1.52 |
| 1 | 2 | 38.4 | 38874 | 153786 | 7.91 | 24 | 2226 | 1206 | 1.85 |
| 2 | 4 | 50.3 | 26105 | 171377 | 13.13 | 72 | 2226 | 947 | 2.35 |
| 3 | 8 | 61 | 19110 | 231061 | 24.18 | 172 | 2226 | 994 | 2.24 |
| 4 | 16 | 81 | 15100 | 375574 | 49.74 | 1129 | 2226 | 1061 | 2.1 |

- About 2.2 times larger than graph of Nice;
- Overlay layer #2 has the best performance for the graph of New York with 947 ms of search;
- The speed-up is worse than the graph of Nice had.

# Conclusions

## Achievements

- A shortest path algorithm that takes into account cyclist’s needs is designed and successfully tested;
- Its performance was made more independent on graph topology;
- A real-world graph with data important for cyclists was created;
- Implemented the diameter search DiFUB algorithm.

# Further development

- Find a good default ratio between metrics;
- Develop a continuous slopes metric;
- Consider individual user’s preferences;
- Expand the graph to the whole PACA region;
- Implement less curves metric;
- Implement several optimal paths search, not only one.

# Thank you!

# Appendices

# The Vertex Cover Problem and solution

## Main Steps in my compression implementation :

1. Create an overlay graph where crossroads are access points
2. Find Vertex cover of the previous layer
3. Create overlay graph for the vertex cover.

## My Solution: Custom implement a vertex cover heuristic (LR-deg).

**LR-deg**: Initialize Vertex Cover VC to an empty Set. For each v ∈ V (v picked in increasing order of degree), add Neighbor(v) to VC if v not already in VC.

## Real time Querying: Funke's algorithm

A fast bidirectional dijkstra using access points (H. Bast et al, 2007)

# Formulas used in the algorithm

| group | entity | key | value | r_time | r_slowdown | r_surface | r_traffic |
|---|---|---|---|---|---|---|---|
| crossing | node | crossing | island | 1 | 20 | -1 | -1 |
| crossing | node | crossing | traffic_signals | 1 | 30 | -1 | -1 |
| crossing | node | crossing | uncontrolled | 1 | 15 | -1 | -1 |
| crossing | node | crossing | unmarked | 1 | 20 | -1 | -1 |
| crossing | node | crossing | yes | 1 | 15 | -1 | -1 |
| crossing | node | crossing | zebra | 1 | 15 | -1 | -1 |
| crossing | node | highway | crossing | 1 | 15 | -1 | 2 |
| crossing | node | highway | traffic_signals | 1 | 30 | -1 | 3 |
| dismount | way | bicycle | dismount | 0.4 | 0 | 2 | -1 |
| dismount | way | footway | crossing | 0.4 | 0 | 2 | -1 |
| dismount | way | footway | sidewalk | 0.4 | 0 | -1 | 0.5 |
| dismount | way | highway | footway | 0.4 | 0 | -1 | 0.5 |
| dismount | way | highway | footway;path | 0.4 | 0 | -1 | 0.5 |
| dismount | way | highway | pedestrian | 0.4 | 0 | -1 | 0.5 |
| for_bicycles | relation | route | bicycle | 1 | 0 | -1 | 0.9 |
| for_bicycles | way | bicycle | designated | 1 | 0 | -1 | 0.2 |
| for_bicycles | way | bicycle | permissive | 1 | 0 | -1 | -1 |
| for_bicycles | way | bicycle | yes | 1 | 0 | -1 | -1 |
| for_bicycles | way | cycleway | lane | 1 | 0 | -1 | 0.6 |
| for_bicycles | way | cycleway | share_busway | 1 | 0 | -1 | 0.7 |
| for_bicycles | way | cycleway | shared_lane | 1 | 0 | -1 | 0.8 |
| for_bicycles | way | cycleway | track | 1 | 0 | -1 | 0.4 |
| for_bicycles | way | cycleway:left | lane | 1 | 0 | -1 | 0.6 |
| for_bicycles | way | cycleway:left | share_busway | 1 | 0 | -1 | 0.7 |
| for_bicycles | way | cycleway:left | shared_lane | 1 | 0 | -1 | 0.8 |
| for_bicycles | way | cycleway:right | lane | 1 | 0 | -1 | 0.6 |
| for_bicycles | way | cycleway:right | share_busway | 1 | 0 | -1 | 0.7 |
| for_bicycles | way | cycleway:right | shared_lane | 1 | 0 | -1 | 0.8 |
| for_bicycles | way | highway | cycleway | 1 | 0 | -1 | 0.2 |
| motor_roads | way | highway | living_street | 1 | 0 | -1 | 0.5 |
| motor_roads | way | highway | primary | 1 | 0 | -1 | 10 |
| motor_roads | way | highway | primary_link | 1 | 0 | -1 | 10 |
| motor_roads | way | highway | residential | 1 | 0 | -1 | -1 |
| motor_roads | way | highway | secondary | 1 | 0 | -1 | 6 |
| motor_roads | way | highway | secondary_link | 1 | 0 | -1 | 6 |
| motor_roads | way | highway | service | 1 | 0 | -1 | -1 |
| motor_roads | way | highway | tertiary | 1 | 0 | -1 | 2 |
| motor_roads | way | highway | tertiary_link | 1 | 0 | -1 | 2 |
| obstacles | way | highway | steps | 0.1 | 0 | 10 | -1 |
| obstacles | node | highway | elevator | 1 | 75 | 7 | -1 |
| obstacles | node | highway | steps | 1 | 25 | 10 | -1 |
| offroad | way | highway | bridleway | 0.7 | 0 | 2 | -1 |
| offroad | way | access | agricultural | 0.8 | 0 | 2 | -1 |
| offroad | way | access | forestry | 0.8 | 0 | 2 | -1 |
| offroad | way | highway | path | 0.7 | 0 | 2 | 0.5 |
| offroad | way | highway | track | 0.8 | 0 | 2 | 0.5 |
| surface | way | smoothness | bad | 0.7 | 0 | 3 | -1 |
| surface | way | smoothness | excellent | 1 | 0 | 0.5 | -1 |
| surface | way | smoothness | horrible | 0.5 | 0 | 2 | -1 |
| surface | way | smoothness | intermediate | 0.8 | 0 | 1 | -1 |
| surface | way | smoothness | very_bad | 0.6 | 0 | 4 | -1 |
| surface | way | surface | cobblestone | 0.7 | 0 | 5 | -1 |
| surface | way | surface | compacted | 0.9 | 0 | 1.5 | -1 |
| surface | way | surface | dirt | 0.7 | 0 | 3 | -1 |
| surface | way | surface | grass | 0.65 | 0 | 5 | -1 |
| surface | way | surface | gravel | 0.5 | 0 | 5 | -1 |
| surface | way | surface | ground | 0.6 | 0 | 4 | -1 |
| surface | way | surface | mud | 0.4 | 0 | 5 | -1 |
| surface | way | surface | paving_stones | 0.75 | 0 | 1.5 | -1 |
| surface | way | surface | sand | 0.6 | 0 | 4 | -1 |
| surface | way | surface | setts | 0.8 | 0 | 2 | -1 |
| surface | way | surface | unpaved | 0.75 | 0 | 4 | -1 |
| surface | way | surface | wood | 0.65 | 0 | 4 | -1 |

Travel time value:

$$c_1(u,v) = \begin{cases} \frac{l(u,v)+a_l \cdot a(u,v) \cdot l(u,v)}{r_1(u,v)} + q(u,v) * s & \text{if } a_l > 0, \\ \frac{l(u,v)}{s_d(u,v,s_{dmax}) \cdot r_1(u,v)} + q(u,v) * s & \text{otherwise,} \end{cases}$$

where l(u, v) - edge distance, a(u, v) - edge slope, r1 - r_time coefficient, q(u, v) - r_slowdown value, s - speed (m/s), sd - downhill speed multiplier

$$s_d(u,v,s_{dmax}) := \begin{cases} s_d max & \text{if } d'(u,v) > d'_c, \\ \frac{(s_{dmax}-1)d'(u,v)}{d'_c} + 1 & \text{otherwise,} \end{cases}$$

where sdmax - maximum downhill speed multiplier, d'(u, v) - edge slope, d'c - critical d' value when speed equals sdmax

Comfort value:

$$c_2(u,v) = l(u,v) \cdot \max\{(r_s(u,v), r_t(u,v))\}$$

where rs - r_surface, rt - r_traffic

Flatness value:

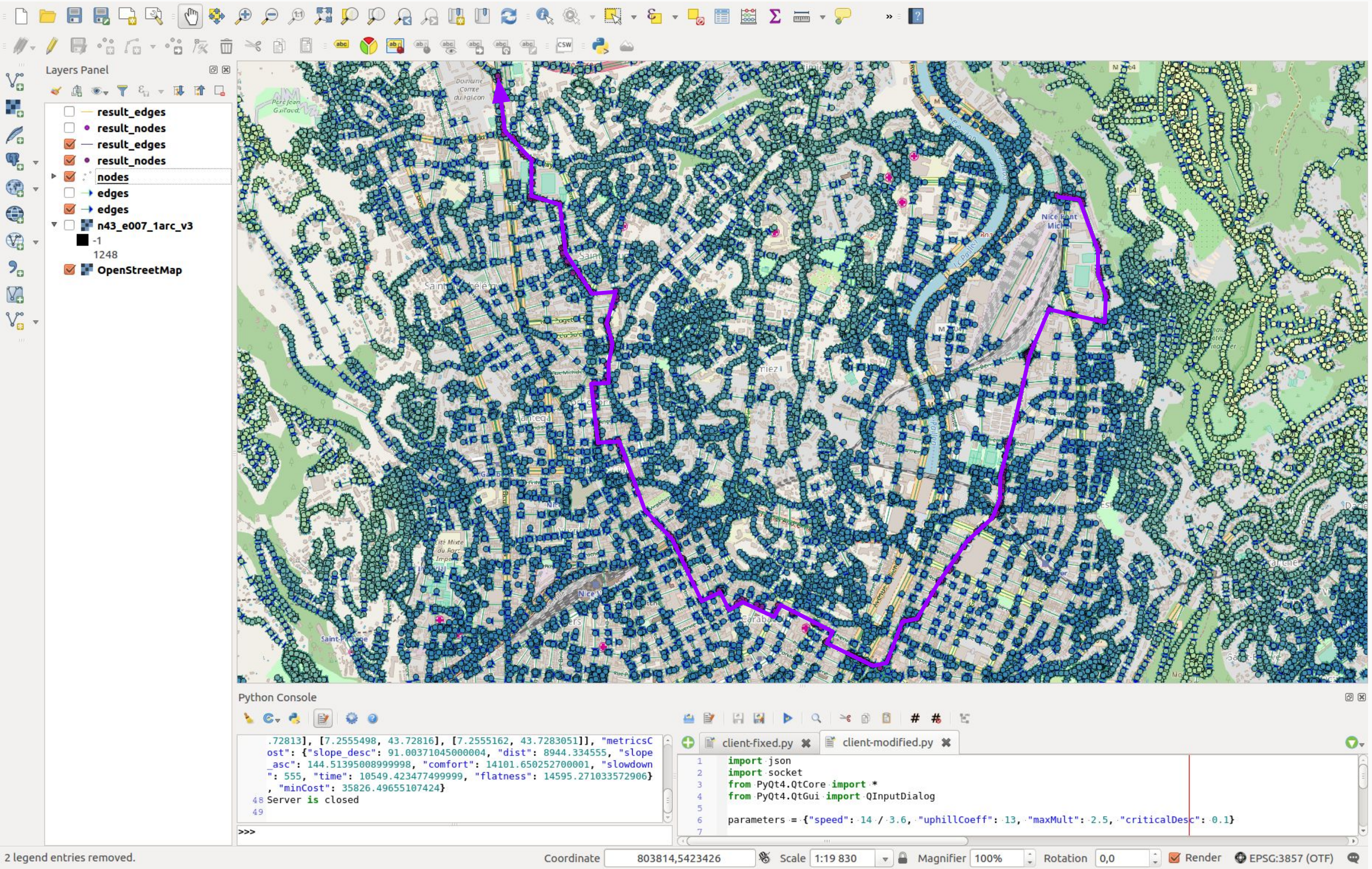
$$c_3(u,v) = \begin{cases} l(u,v) \cdot (10128.074 \cdot a(u,v)^3 - 140.785 \cdot a(u,v)^2 + 6.693 * a(u,v) + 1) & \text{if } a(u,v) > 0, \\ l(u,v) & \text{otherwise.} \end{cases}$$

# Initial version of the algorithm performance

Analysis of the initial algorithm’s performance on the map of Nice

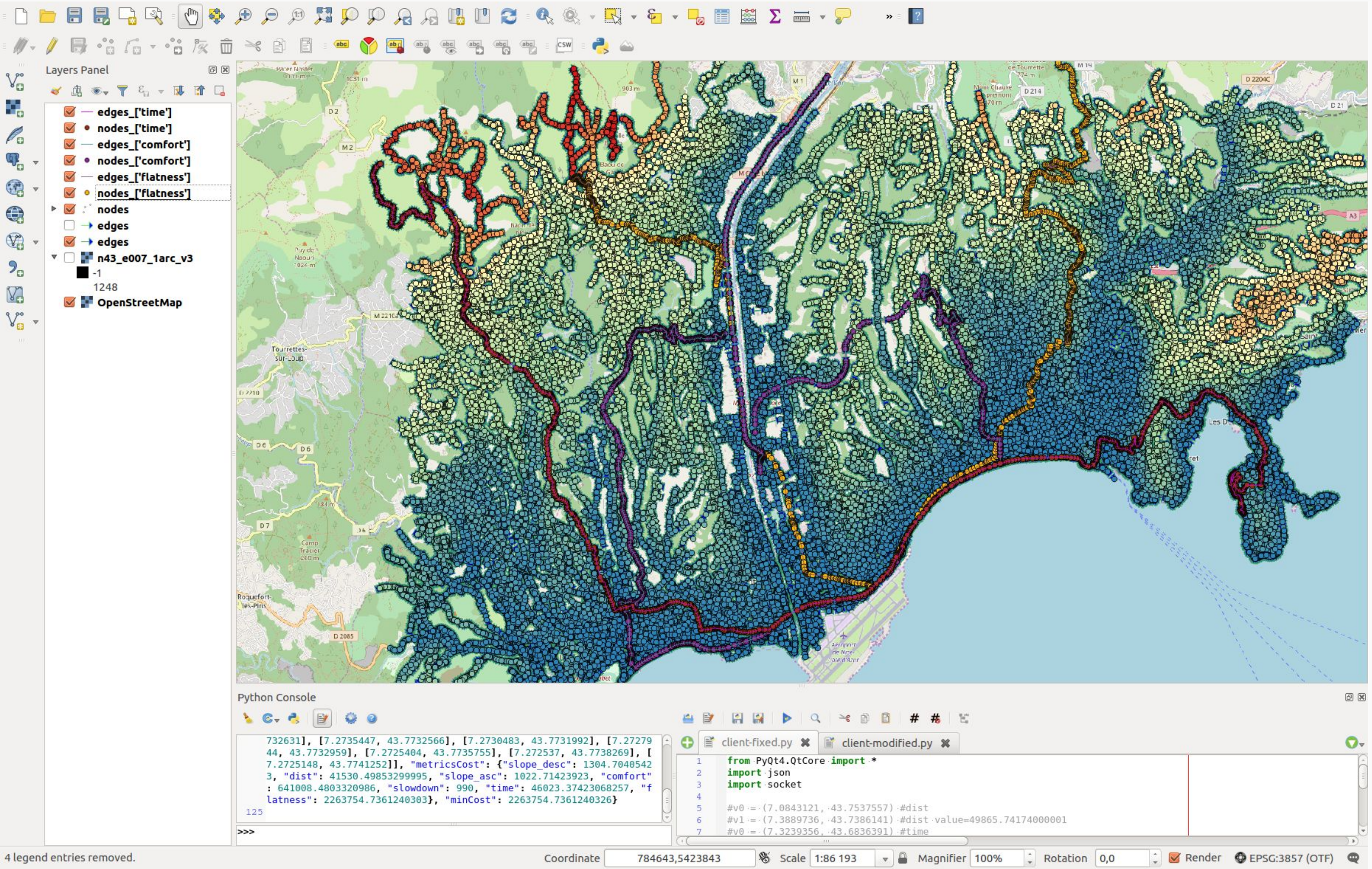
| Overlay layer | k | Construction time (s) | # vertices | # edges | D avg | D max | Dijkstra (ms) | Search (ms) | Speed up |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 7.67 | 100768 | 200155 | 3.97 | 56 | 873 | 1898 | 0.46 |
| 1 | 2 | 23.1 | 55552 | 119675 | 4.31 | 146 | 873 | 975 | 0.9 |
| 2 | 4 | 32.6 | 30646 | 79887 | 5.21 | 322 | 873 | 561 | 1.56 |
| 3 | 8 | 38.7 | 17169 | 71627 | 8.34 | 1034 | 873 | 458 | 1.91 |
| 4 | 16 | 49.2 | 9923 | 114459 | 23.07 | 4263 | 873 | 358 | 2.44 |
| 5 | 32 | 138 | 6070 | 365236 | 120.34 | 28338 | 873 | 466 | 1.87 |

# Example of optimal path




- The optimal path avoids:
  - hills;
  - major roads;

# Diameters of the graph




The paths which are:

- the most time-consuming;
- the least comfortable;
- the most hilly.