Genetic Algorithm for Bicycle Route Search

Branislav Ševc
456263@mail.muni.cz

4.8.2020

Abstract

This document summarizes the algorithm used for multiple-solution search for cycling routes based on user-provided parameters, including the evaluation methodology and results.

Contents

1 Introduction

The purpose of the algorithm is to recommend suitable bicycle trip/training routes for given parameters. An interest in this may come from professional and semi-professional cyclists, who may want to plan their training sessions with careful choice of elevation profile. The search is performed in an unsupervised manner, i.e. without existing route database. Genetic algorithm (GA) is used to optimize a population of routes with respect to appropriate cost function. The main reason of choosing GA is to produce not one, but multiple solutions the user can choose from. Diversity of solutions is therefore equally important goal.

2 Input

The input of the algorithm takes form of a map data and user-defined search parameters.

2.1 Map

The map data come in modified OSM JSON format. The structure is rid of unneeded tags and nodes are completed with extra `ele` attribute for elevation.

Elevation data are not part of the OSM project, thus have to be acquired from different, usually local sources. We have chosen to do so using Mapy API from Seznam.

The other step is the actual preprocessing, as the OSM's segmentation of ways is not suitable for the algorithm. Preprocessing consists of two phases:

1. splitting each way on each junction node,
2. concatenating each reasonable consecutive ways (sharing a node), if degree of this node is two.

2.2 Parameters

The search parameters can be classified into cost-defining parameters, constraining parameters, and hyperparameters.

The cost-defining parameters are

- distance – the desired route distance in km,
- elevation profile – grade function – partial function ($km \mapsto \%$).

The constraining parameters are

- transit points – a list of WGS84 coordinates that need to be traversed by the route in the given order.

The hyperparameters are namely

- iteration count,
- input population size,
- output population size,
- way type weights for stochastic way selection.

3 Measures

3.1 Cost

3.1.1 General Formula

The cost function is designed to be as intuitive as possible. It answers the question:

"how many times is this solution/value worse than the desired one?".

From this interpretation, we get the following general formula, with some edge cases included (x – desired value, $\tilde{x}$ – actual value):

$$c(x, \tilde{x}) = \begin{cases} max\{\tilde{x}, 1\} & \text{for } x = 0, \\ max\{x, 1\} & \text{for } \tilde{x} = 0, \\ max\{\frac{\tilde{x}}{x}, \frac{x}{\tilde{x}}\} & \text{otherwise.} \end{cases}$$

The following axioms hold:

$$\begin{aligned} c(x, \tilde{x}) &\geq 1 \\ c(x, \tilde{x}) &= c(\tilde{x}, x) \\ c(x, x) &= 1 \\ c(x, x \cdot k) &= max\{k, \frac{1}{k}\}, \text{ for } x, k > 0 \end{aligned}$$

If $x = 0$ and $\tilde{x} \geq 1$, we obtain $c(x, \tilde{x}) = \tilde{x} = \tilde{x} - x$ and vice-versa, i.e. the absolute difference instead of proportional. The case of $x = 0$, $\tilde{x} < 1$ and vice-versa is considered to be extreme. In practice, we do not look for routes with length of less than one meter. Similarly, we do not differentiate between a route with the cumulative elevation profile of zero or one meters.

3.1.2 Distance Cost

The cost function for distance is obtained by direct substitution into the general formula (d – desired distance, $\tilde{d}$ – actual distance):

$$c_d = c(d, \tilde{d})$$

3.1.3 Elevation Profile Cost

To obtain cost of a route's elevation profile, a bit of transformation needs to be done. Let $g(s)$ denote the desired and $\tilde{g}(s)$ the actual **slope**, where s is a sample point (distance from the start of the route) and S is the set of all sample points (zero-based, uniformly distributed).

First, we define the difference of elevation profiles (grade functions) as the sum of absolute deviations (SAD):

$$\begin{aligned} d_g &= \sum_{s \in S} |\tilde{g}(s + t) - g(s)| \\ t &= \operatorname*{argmin}_{t \in T} w(t) \cdot \sum_{s \in S} |\tilde{g}(s + t) - g(s)| \end{aligned}$$

The value t allows a certain offset while trying to find the minimum difference. T is the set of allowed offsets. For values outside the function domain, we put $|\tilde{g}(s+t)-g(s)| = 0$. The difference is weighted by $w(t)$, which is currently a constant function equal to 1.

Next, this difference is integrated to obtain the difference in elevation from the desired elevation profile d_e. Since sampling step Δs is constant, this is corresponds to multiplication. Likewise, e, the total desired elevation, is calculated for the defined portion of the desired grade function.

$$d_e = d_g \cdot \Delta s$$
$$e = \sum_{s \in S} |g(s)| \cdot \Delta s$$

Finally, the grade cost is calculated by the following substitution into the general formula:

$$c_g = c(e, \tilde{e});\ \tilde{e} = e + d_e$$

3.1.4 Scalarization

From the above two formulas, we get a vector $\vec{c} = (c_d, c_g)$, which we convert to a scalar with respect to the intuitive meaning of cost. That is, we multiply individual sub-costs to obtain the total cost $c_{total}(c_d, c_g) = c_d^{w_d} \cdot\ c_g^{w_g}$, where w_d and w_g are weights to make one sub-cost more influential than the other. So far, we put $w_d, w_g = 1$.

The following axioms hold for this scalarization:

$$c_d \geq 1 \wedge w_d \geq 0 \wedge c_g \geq 1 \wedge w_g \geq 0 \implies c_{total}(c_d, c_g) \geq 1$$
$$c_{total}(c_d \cdot k, c_g \cdot l) = c_{total}(c_d, c_g) \cdot kl \text{ for } k, l \geq 1$$

Thus, we get a total cost function, which expresses relative inferiority of a solution as a multiplication of defects of individual traits. For example, for a route, which is two times longer than desired and for which, the grade is twice as much for each sample in its first half (which is compared to the desired grade function), we obtain $c_{total} = 2 \cdot 2 = 4$.

3.2 Similarity

The similarity metric is used in deterministic crowding to select which way a pair of offsprings replaces their parents. It is chosen to be the Jaccard similarity index, a.k.a. intersection over union. The intersection and union sets in this context consist of individual directed ways. The size of each set is calculated as the total distance of the ways it contains.

$$s(s_1, s_2) = \frac{||s_1 \cap s_2||}{||s_1 \cup s_2||}, \text{ where } ||W|| = \sum_{w \in W} ||w||$$

4 Algorithm

4.1 Solution Encoding

Direct encoding is employed; genotype and phenotype of a solution is the same structure.

In favor of simplicity and performance, the algorithm transforms the problem domain into finite graph search. The decision points are junction nodes, meaning that the algorithm cannot decide in the middle of a way to turn around. Its decisions are simply not that fine-grained.

A solution comes out as a route (sequence of ways, where each way shares the end node with the start node of the next way). More precisely, this route is split into consecutive sub-routes by the transit nodes. It allows the implementation to operate on these parts independently (and in parallel).

4.2 Initialization

The purpose of the initialization phase is to generate a diverse population of valid and meaningful solutions. The most significant constraint for this are the transit points, which are at the same time algorithm's guides. First, however, these points, which come in arbitrary WGS84 coordinates, are snapped to the map. The closest junction node is found for each transit point, with which the algorithm can now work. The nodes are connected using randomized search algorithms mentioned in Section 4.7. To obtain a diverse set of solutions, the weights of ways in the map decrease with their use (niching). This phase can be also parallelized by using differently seeded RNGs for each thread. Doing so produces somewhat less diverse, but still usable output.

4.3 Selection for Reproduction

This phase selects a subset of current population suitable for reproduction (currently a half). The algorithm used is Stochastic Universal Sampling (SUS). Picked solutions are randomly formed into pairs with uniform pairing probability distribution.

4.4 Recombination

Recombination is performed using Deterministic Crowding (DC) approach. Crossover is performed on copies of two paired solutions (parents) to obtain two children. Each child undergoes separate mutation. Afterwards, an integrated tournament is held between the children and the parents to select the two solutions replacing the original pair.

We have made additional measures in order to avoid producing two identical solutions. In such case, the second solution to be produced is replaced with the fitter solution of the remaining two. If this is not possible (all are identical), the solution is invalidated and does not participate in the next population. This strategy not only conserves diversity, but enforces it. Following operators are designed to preserve consistency of solutions.

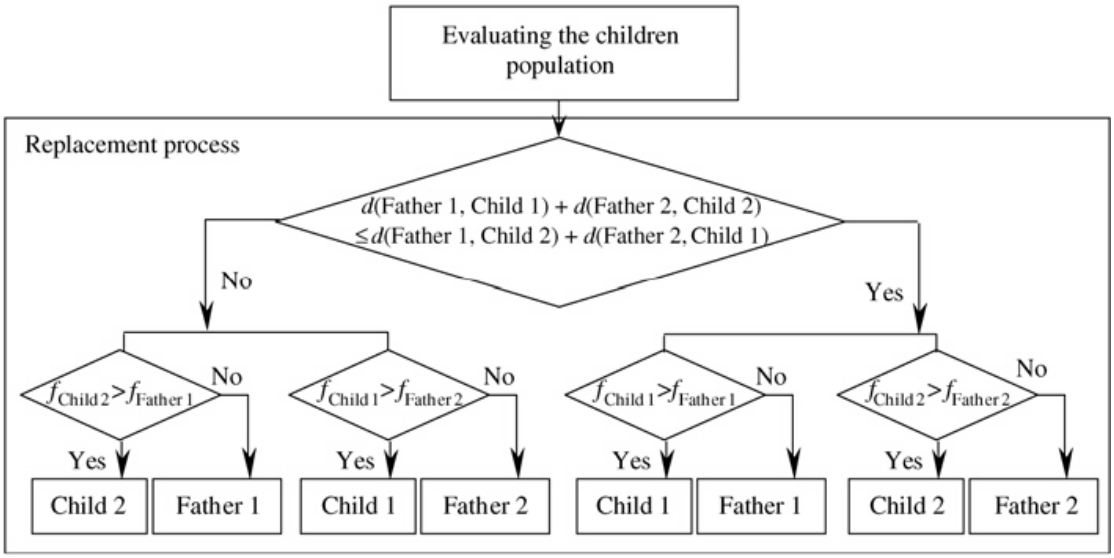

Figure 1: Deterministic crowding algorithm

The figure shows a basic replacement process with DC. d and f are distance and fitness functions, respectively.

4.4.1 Crossover

One-point crossover is employed for each sub-route. A sub-route is chosen for mutation with another using uniform sampling without replacement. If two solutions' sub-routes contain at least one crossing point, a crossing point is chosen with uniform probability.

4.4.2 Mutation

This operator modifies the input solution in the following universal manner:

1. select a part to remove;
2. generate two random routes from the ending nodes, one forward, one in backward direction;
3. connect the other two ends of the random routes in the forward direction.

4.5 Selection for the Next Generation

Successive generation is formed by truncation selection. Least-fit solutions are discarded to form a new (usually smaller) subset.

4.6 Internal Parameters

The algorithm is accompanied by additional internal heuristics and constants:

- distance cost factor – w_d – set to 1;
- grade cost factor – w_g – set to 1;
- route crossover probability – $\frac{configurable\_factor}{route\_count}$;
- route mutation probability – $\frac{configurable\_factor}{route\_count}$;
- way initialization reuse cost factor – multiplier for an internal metric – set to 1;
- way mutation reuse cost factor – multiplier for an internal metric – set to 1;
- reproducing count – half of the current population size;
- duplicate solution removal – ensure unique solutions in each population – enabled;
- missing solution filling – copy most fit solutions from the previous population to increase its size after duplicate removal – enabled.

4.7 Stochastic Algorithms

Stochastic algorithms are used throughout the program (initialization and mutation phases) to explore the search space of possible solutions. Such an algorithm outputs a route between two nodes. Currently, there are two main search strategies, depending whether the input nodes are identical or not.

4.7.1 Random Route Search (two nodes)

Two distinct nodes are connected using the A* algorithm. To limit the search space and produce more human-like routes, the algorithm prefers ways of higher-weighted types and lower-angle turns. During mutation, weights of previously used ways are decreased to promote exploration of new solutions (similar to niching used in initialization).

4.7.2 Random Route Generation (one node)

This strategy is used only during initialization, when two consecutive transit nodes are identical. The strategy is expected to produce a loop starting and ending at the given node. An additional parameter – expected loop length is required. This value is calculated as the distance left to reach the desired distance, divided by the number of loops, with negative values clamped to zero. The algorithm first generates random outgoing and incoming routes from this point. The generation of each route is stopped after reaching $\frac{1}{3}$ of the expected loop length. The other ends of these routes are then connected together using the previous algorithm. Thus, the loop comprises of three route segments, with total distance from around $\frac{2}{3}$ to $\frac{4}{3}$ of the requested distance.

5 Evaluation

In the following benchmarks, we consider 100 iterations, 1000 initial solutions and 10 final solutions. We set `mutation_probability_factor` to 0, effectively disabling any mutations. It was discovered that the current implementation produces unlikely desired routes with many random artifacts and frequent, short detours.

The outputs are viewable in a custom web application provided with the data. It is able to display aggregated route population with metadata (i.e. costs, total distance and elevation change) and supports range filtering based on total cost.

5.1 Machine Evaluation

The algorithm can produce large amounts of data that is challenging to evaluate manually. Therefore, we have resorted to use primarily evaluation via an automated benchmark. This benchmark consists of the following steps:

1. generate a random, but "meaningful" route on the map;
2. deduce the user-provided parameters for this route;
3. run the algorithm with the parameters;
4. output the similarity index between the generated route and the most similar output route.

The idea behind the benchmark is to expect the generated route (or a similar one) among the output route set, measuring recall of the search. The test is repeated to reduce variance and stabilize the results, both mean and median.

Currently, we generate routes with lengths 25, 50 and 100 km in the South Moravian Region. For each case, we deduce a different number of transit points: 2, 3, 5, 10 and 50. These are chosen uniformly from all junction nodes of the route. Secondly, we choose the defined portion for elevation profile creation (0, 25, 50, 75 and 100%). The complete elevation profile is calculated for the generated route (by sampling with 1 m step), then it is divided into approximately 2 km long intervals. The profile value is kept (approximately) in the corresponding percentage of intervals, values in other intervals are undefined (reset to NaN). Maximum misalignment of elevation profile is set to 0.

We run each benchmark case 50 times. Overall, we get $3 \cdot 5 \cdot 5 = 75$ benchmark cases and $75 \cdot 50 = 3750$ algorithm runs. We present the results in series of bar charts, showing the dependence of similarities on `point_count` and `defined_portion` parameters. The similarities are aggregated as an average of maximum similarities of each run's output population. The corresponding files can be found in the `evaluation` directory.

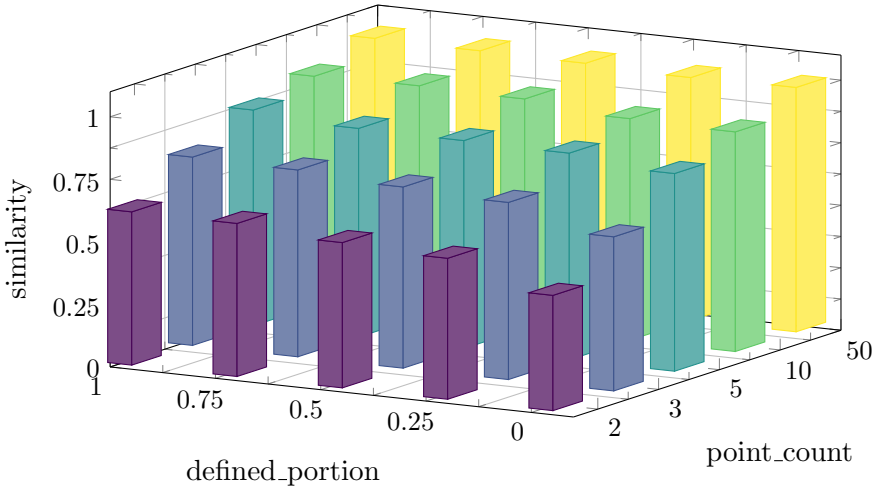

Figure 2: 25 km route search

The figure shows the result for generated routes of length approximately 25 km. Clearly, the number of transit points has a great descriptive power, thus the effect on the maximum similarity average dominates that of defined_portion. However, we are pleased with the fact, that the effect of defined_portion is still visible, mostly with lower point_count. For example, the difference between defined_portion = 0 and defined_portion = 1 at point_count = 2 is around 15%. At higher point_count, similarities are monotonically increasing with increasing defined_portion. There are only few exceptions, as the values seem to be influenced by a random noise from the generation phase and by floating-point errors of elevation profile deduction.

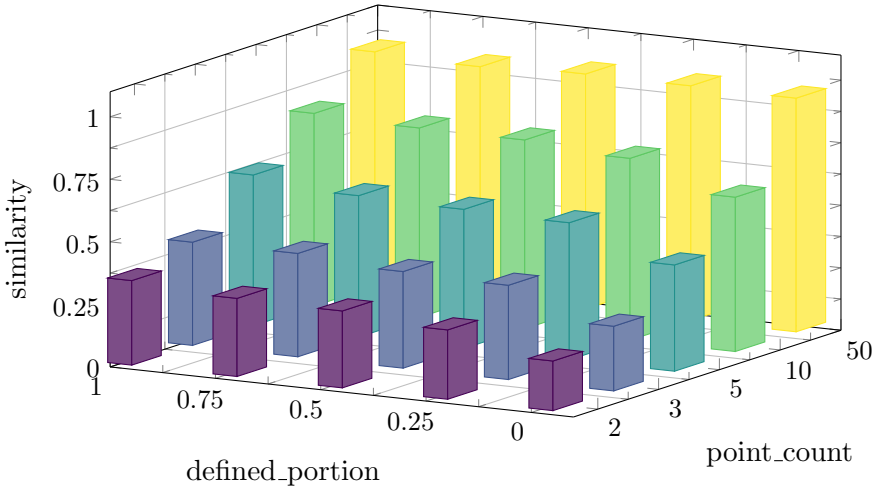

Figure 3: 50 km route search

This graph shows similar trend to the previous one, although the impact of point_count is even more prevalent. The effect of defined_portion is still nearly monotonic.

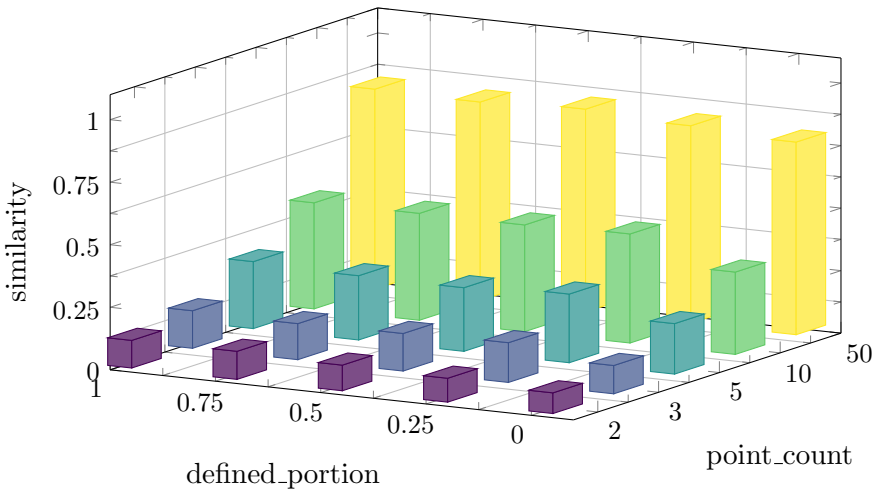

Figure 4: 100 km route search

The last set of test cases continues with the trend. There is still an overall positive effect of defined_portion. A larger number of transit points would be desired to describe the routes sufficiently, so that we get similarities above 95%, but that is far from a real use case.

5.2 Human Evaluation

For a limited number of searches, we have used our own knowledge of the local area to evaluate the algorithm's performance intuitively. We consider three areas of South Moravian, Olomouc and Moravian-Silesian regions with different characteristics. For each area, we try a number of different searches and judge the quality of output. As far as the desired elevation profile goes, we will describe a certain number of hills, by specifying both ascent and descent intervals with constant magnitude of 5 or -5%. Maximum misalignment of the grade function is set to 5 km to allow slight labeling errors. Our workflow is to try a search with the least transit points (i.e. 2) first, then to add key points to guide another search in case of unsatisfactory output.

The corresponding files can be found in the `human_evaluation` directory.

5.2.1 Moravský kras

First, we will test the algorithm on our favorite training area near Brno. This area has a characteristic relief with relatively uniform hills dissected by a few valleys. Hills start approximately at 250 m.a.s.l. and peak at around 500 m.a.s.l. The tested routes have several uniformly distributed hills that are similar in profile. The first and the last transit points are the same, meaning that we are looking for a ride "in a loop".

61 km, 3 hills

We can immediately say that the algorithm will not provide desired results with less than two intermediate points. The points are needed, as there are two detours – points, where we use to turn around. The algorithm currently limits its search space by not allowing spontaneous change of direction in the middle of a sub-route, so we need to make them a sub-route delimiting points/nodes.

We have tried 2-5 transit points. With no transit points, the initialization process (4.7.2) is quite different and not yet at the top of its form. It produces seemingly randomly convoluted routes, with which the algorithm cannot cope. Certainly, the fact that the points are in the geographical center of a city (dense way network), makes this effect worse. This trend continues in all such setups, so we will not mention it further.

With one transit point placed at the 2nd peak, we are able to obtain a population consisting (correctly) only of routes with 3 similar hills. With one more point placed at the other detour's end, we obtain two solutions that are very close to the desired route. They differ only in last, ca. 7 km long segment, which is even similar to the desired one. However, the segment is included in number of other solutions. We attribute this defect to insufficient search through available combinations. An additional transit point solves this quickly and the algorithm produces a nearly-perfect route. In comparison, 8 transit points are required to find this route with Mapy.cz, whereas we need only 5 (including start and end points).

71 km, 4 hills

This route contains only one detour. With an additional transit point placed at the detour, the initial population is limited in direction to this point and the rest of the required area remains unexplored for the rest of the iterations. Thus, we should improve our search heuristic, which is currently guided by geographical distance to this point. With one more transit point, we were not able to get satisfying output. With 5 carefully placed transit points, we get variants of the expected solution. Only routing through the city is slightly different from our taste. In comparison, Mapy.cz require around twice as much transit points.

76 km, 5 hills

This route is largely similar to the previous one, only that it contains one extra hill at the start. It is slightly more complex, therefore we test with 5 and more transit points. The output is satisfactory even with 5 points, unchanged from the previous case. An unexpected, but fitting solution was found ranking in the middle of the output population. Clearly, Mapy.cz would need more guiding transit points than previously.

Overall, the solutions in cases with more than two transit points were always reasonable. This was supported by their cost function, which was often lower than the cost of the output we aimed for. This also tells us that the input parameters may be hard to fill in appropriately, not to cause serious misfit. Our test cases have discovered some flaws in the initialization and reproduction phases.

5.2.2 Pálava

Next, we look at a different type of terrain. Pálava Protected Landscape Area is included among the most popular trip areas from Brno, perhaps because of no hills being in the way. Pálava itself is a small hilly area surrounded by plains. In the following cases, we will not expect any particular output, we will merely judge the quality of all output solutions.

110 km, Mikulov, no hills

First, we try to find a flat route to Mikulov (town). For this, we put the grade function to constant 0. This will cause the edge-case of the grade cost function to be used. With just two transit points, the output is sorrowful. Apparently, grade cost minimization outweighs the distance, because grade cost reaches much higher numbers now. With transit point in Mikulov center, the output is acceptable. 7 out of 10 outputs are almost identical same-return routes with only minor differences in Mikulov. The other three solutions are more diverse and contain a few digressions from the routes to Mikulov. Judging by the initial population, this case would profit from a higher `way_initialization_reuse_cost_factor`.

110 km, Mikulov, one hill

We have modified the previous search, so that it contained a small hill at 32 km from the start. The algorithm tried, but the result was limited because of the same lack of

diversity. Effectively, the diversity of the output was even lower, because only small number of explored routes was deemed fit.

110 km, Mikulov, Pavlov, no hills

We return to the first case and put and additional point in Pavlov (village) after the one Mikulov. The grade function remains to be set to 0. As expected, all routes avoided the hilly terrain that lays between the two municipalities. The detour was always chosen the same way, from the north. Other parts of routes remained relatively diverse.

110 km, Mikulov, Pavlov, one hill

As the last case in Pálava, we have changed our previous requirement of no hills to on hill between Mikulov and Pavlov. The result was admissible, with two possible routes between the two municipalities. Likewise, the solutions were overall relatively diverse.

5.2.3 Jeseníky

Lastly, it would be pitiful not to include quite the opposite to Pálava, mountains. The terrain is more coarse-grained, and route density is lower. This is perhaps the easiest area among the tested ones. All our routes start and end in Šumperk (town). We have used 3 transit points in all cases except the last one, where we have used 4. Placing transit points on important places (e.g. Praděd) where there is only one access route (it is a detour) severely depreciates the demonstrative purpose of these tests, but, unfortunately, our algorithm does not have enough search power yet. Due to the elevation profile cost, in most cases, there were not enough options for a more diverse output.

62 km, Dlouhé Stráně, 1 hill

We were able to successfully plan a trip through Dlouhé Stráně (mountain). There are two ways to get to the top, and they are different in profile. We have specified both ascent and descent gradient, and this produced only routes, which used the first way for the ascent and the other for descent. Perhaps, it would be better to have more slack in selection, so that we would obtain other ascent-descent combinations also. Selection could be based on a threshold, which would be a multiple of the lowest cost among solutions. If we wanted to preserve low output count (e.g. 10), this would trade-off the diversity of best solutions.

115 km, Praděd, 3 hills

The rest of the tests include passes through the most prominent mountain of the area. We start with the shortest route, which has approximately 115 km. There is one more hill on the way there (and back). The algorithm correctly found a set of shortest routes, at diversity's expense.

125 km, Praděd, 3 hills

This test differs only by a few kilometers in desired distance from the previous one. It is expected from the algorithm to choose a bit longer (not direct) path at the start and

the end, or prolong the route in other parts. All routes were above 120 km long and no route passed through the shorter section from the previous case.

125 km, Praděd, 5 hills

This time, we aimed for a trip around Praděd, which includes 5 major hills. We have described grade of all hills, kept three transit points and the algorithm coped well. Out of curiosity, we have tried the search without defining the grade function and it turned out to be unsatisfactory. All solutions avoided the mountain passes to the north of Praděd.

144 km, Dlouhé Stráně, Praděd, 6 hills

Finally, we have included one more prominent hill, Dlouhé Stráně to the start of the previous plan. This was achieved by placing a transit point on the top. Presumably, a transit point at the end of the descent would be sufficient, too. The output was not hampered in any way, compared to the previous case.

In this section, we have tested the algorithm intuitively, judging the quality of output under different conditions. The results cannot be quantified, so it is best to review them one-by-one. One can quickly get a sense for diversity and "meaningfulness", although the latter may vary from person to person. Rather than stating the ratio of satisfactory routes of each output set (which is not easy), we have tried to explain the reason behind algorithm's behavior.

6 Conclusion

We have implemented basic route-generating GA with direct encoding and advanced diversity control. Major flaws remain in the implemented search methods, which worsen the performance of initialization phase and mutation operator. Search is overly biased towards connecting consecutive transit points, and if these points are identical (or lie much closer to each other than is the expected connection length), it is not satisfactory. The search should also be able explore dead ends and to be potentially guided by the grade function, at least in the initialization phase. Perhaps the two search methods could be united into one universal one.

Concerning user experience, the number of algorithm's hyperparameters was tried to be kept as low as possible, with meaningful defaults. Input of the algorithm is currently provided as a JSON file, and this is meant to be filled in by a web-based frontend. Regardless, it turned out, that the grade cost function is sensitive to minor inconsistencies. A substantial improvement in this direction would be to internally split the grade function into several intervals and perform matching with different offset for each of them.

A algorithm/README.md

A.1 Licensing

- to be decided

A.2 Build Requirements

The project requires C++17 standard support. Furthermore, it requires the following libraries to be installed on the system:

- Boost - version 1.68.0 or higher (with modules `program_options` and `system`)
- OpenMP
- bzip2
- zlib
- Expat

These prerequisities are contained within a custom-made docker image - https://gitlab.fi.muni.cz/xsevc/a project-docker. A built docker image can be found at https://hub.docker.com/r/sevc/ai-project.

A.3 Submodules

Other dependencies are included as Git submodules:

- ExprTk - for convenient grade function user input
- JSON for Modern C++ - for reading of Overpass map and input configuration files
- libfort - for table printouts in log messages

Use the `--recursive` flag while issuing `git clone` or call `git submodule update --init --recursive` separately to fetch them.

A.4 Building

First, create and switch to a directory for an out-of-source build:

```
mkdir build && cd build
```

Now, generate the build system files via CMake, e.g.:

```
cmake .. -G "Unix Makefiles"
```

Finally, run:

```
make -j <BUILD_PROCESS_COUNT>
```

A.5 Usage

The algorithm is run using the following command:

```
ai_project_evolve -c <CONFIG_FILE>
```

<CONFIG_FILE> is a path to the input configuration JSON file, which may look the following way:

```
{
  "map_path": "../maps/cr_regions/jihomoravsky.json",
  "parameters": {
    "points": [
      { "lat": 49.2072808, "lon": 16.6019825 },
      { "lat": 49.3351631, "lon": 16.7180686 },
      { "lat": 49.2072808, "lon": 16.6019825 }
    ],
    "distance": 60,
    "grade": {
      "function": "0",
      "sampling_step": 0.01,
      "max_misalignment": 1
    },
    "way_type_preferences": {
      "trunk_link": 0.5,
      "primary_link": 0.5,
      "residential": 0.5,
      "cycleway": 0.5,
      "service": 0.25,
      "track": 0.5
    }
  },
  "output_path": "../outputs/kras",
  "output_verbosity": ["log_time", "final"],
  "iteration_count": 100,
  "initial_count": 1000,
  "final_count": 10,
  "distance_cost_factor": 1;
  "grade_cost_factor": 1;
  "crossover_probability_factor": 1;
  "mutation_probability_factor": 1;
  "way_initialization_reuse_cost_factor": 1;
  "way_mutation_reuse_cost_factor": 1;
  "reproducing_ratio": 0.5;
  "remove_duplicates": true,
```

```
  "fill_missing": true,
  "thread_count": 1,
  "deterministic": true
}
```

Explanation:

- `map_path` - path to the **preprocessed** OSM JSON map file (relative to the config file) (Wiki)
- the preprocessed OSM JSON file is created using:
  1. `script/download_elevation/download_elevation.html` – to add non-standard `"ele"` values to nodes (from Mapy API, allowing ~10 nodes/s)
  2. `ai_project_preprocess` - to "normalize" the ways in the followind steps:
  3. split ways on junctions
  4. concatenate ways on non-junctions
- `parameters` **- the user input parameters of the desired route**
- `points` **- a list of WGS84 transit points to which all routes will be constrained**
  - each point is snapped to the map by finding the closest junction node
- `distance` **- the desired route distance [km]**
- `grade`
  - `function` **- the (partial) grade function (km to %) in ExprTk format**
  - `sampling_step` - the interval at which the above function will be sampled [km]
  - `max_misalignment` - the maximum allowed misalignment of the grade function and the route [km]
- `way_type_preferences` - weights of way types in non-deterministic heuristics (initialization, mutation); not used by the cost function
  - the available types are `unclassified`, `primary`, `secondary`, `tertiary`, `trunk_link`, `primary_link`, `secondary_link`, `tertiary_link`, `residential`, `living_street`, `cycleway`, `service`, `track` (Wiki)
  - if not mentioned, the value defaults to `1`
- `output_path` - the path of the root output folder (relative to the config file), containing:
- `iteration_*` folders, containing:
  - `place_*.gpx` files - solutions sorted by cost in ascending order

  - `info.csv` - statistics table about the solutions, i.e. total cost, distance cost, grade cost, distance, elevation and corresponding medians, means and standard deviations

- `initial` - symlink to `iteration_0`
- `final` - symlink to `iteration_<iteration_count>`
- `place_*.gpx` - symlink to `iteration<iteration_count>/place*.gpx`
- `info.csv` - symlink to `iteration<iteration_count>/info.csv`
- `output_verbosity` - flags to filter the introspection output:
- `none` - no output
- `all` - combination of:
  - `save_all` - combination of
    - `save_solutions` - combination of:
      - `save_initial_solutions`
      - `save_intermediate_solutions`
      - `save_final_solutions`
    - `save_info` - combination of:
      - `save_initial_info`
      - `save_intermediate_info`
      - `save_final_info`
  - `log_all` - combination of:
    - `log_info` - combination of:
      - `log_initial_info`
      - `log_intermediate_info`
      - `log_final_info`
    - `log_phases` - log which phase is currently in progress (initialization, recombination, etc.)
    - `log_iterations` - log which iteration is currently in progress
    - `log_time` - log the time taken for the whole algorithm

- `initial_info` - combination of `save_initial_info` and `log_initial_info`,
- `intermediate_info` - combination of `save_intermediate_info` and `log_intermediate_info`
- `final_info` - combination of `save_final_info` and `log_final_info`
- `info` - combination of `final_info` and `intermediate_info`
- `initial` - combination of `save_initial_solutions` and `initial_info`
- `intermediate` - combination of `save_intermediate_solutions` and `intermediate_info`
- `final` - combination of `save_final_solutions` and `final_info`
- `iteration_count` **- the number of iterations to perform (**`0` **- initialization only)**
- `initial_count` - size of the initial population
- `final_count` - the number of final outputs

- `thread_count` - the number of threads to use for parallelizable tasks, i.e. recombination, cost function computation and output serialization
- `*_factor` - various exponential and multiplicative factors - additional hyperparameters
- `reproducing_ratio` - the ratio of solutions in a given population that will reproduce
- `remove_duplicates` - enables/disables duplicate solution removal after crossover and mutation to boost diversity
- `fill_missing` - enables/disables supplying the current population with solutions discarded from the previous population (before crossover and mutation) in order to prevent premature convergence
- `deterministic` - a boolean flag whether to guarantee that for the same input, the output will be the same each time

A.6 Evaluation

Evaluation is done by repeatedly running the algorithm using the following command:

```
ai_project_evaluate -c <CONFIG_FILE>
```

<CONFIG_FILE> is a path to the input configuration JSON file, which is very similar to the config file for the evolution itself:

```
{
  "map_path": "../../maps/cr_regions/jihomoravsky.json",
  "parameters": {
    "point_count": 2,
    "distance": 50,
    "grade": {
      "defined_portion": 0.5,
      "distance_per_interval": 2,
      "sampling_step": 0.001,
      "max_misalignment": 0
    },
    "way_type_preferences": {
      "trunk_link": 0.1,
      "primary_link": 0.1,
      "residential": 0.1,
      "cycleway": 0.1,
      "service": 0.05,
      "track": 0.25
    }
  },
  "output_path": "../outputs/test",
```

```
  "iteration_count": 100,
  "initial_count": 1000,
  "final_count": 10,
  "distance_cost_factor": 1;
  "grade_cost_factor": 1;
  "crossover_probability_factor": 1;
  "mutation_probability_factor": 1;
  "way_initialization_reuse_cost_factor": 1;
  "way_mutation_reuse_cost_factor": 1;
  "reproducing_ratio": 0.5;
  "remove_duplicates": true,
  "fill_missing": true,
  "run_count": 10,
  "thread_count": 12,
  "deterministic": true
}
```

Explanation:

- `parameters` - mixed parameters for generation, as well as evaluation
- `point_count` - the number of transit points that will be deduced (min. `2`)
- `distance` - the **approximate** distance of the tested route
- `grade`
  - `defined_portion` - the **approximate** portion of grade samples, which are defined
  - `distance_per_interval` - the mean distance of intervals, which are used to deduce the grade input
  - intervals are randomly cleared to NaN, until the desired `defined_portion` is reached (crossed from above)
- `output_path` - the path of the root output folder (relative to the config file), containing:
- `case_*` - output folder for each case, additionally containing:
  - `expected.gpx` - the route from which the input parameters for this case were generated
  - `similarity.csv` - the results of measuring similarity between the final solutions and the expected route (including mean, median and standard deviation)
- `similarity.csv` - aggregation of the best similarity results from each case
- `run_count` - the number of cases to run

The rest of the parameters is inherited from the evolution and each of these parameters is passed as-is.

B visualizer/README.md

B.1 Usage

Run `npm install` in this directory and open `index.html` in your browser.

Click the green import button in the top-right corner and select a set of `.gpx` files to open. Files with other extensions are automatically filtered out. Afterwards, a set of routes and a "timeline" is displayed. The timeline serves as a range filter of total cost. User can hover over a route in a map and a timeline entry. This highlights the route on the map and displays additional metadata in a side pane. Elevation profile graph is unfortunately not implemented.

Importing other files the second time may not work properly; it is recommended to reload the whole page.

C Screenshots

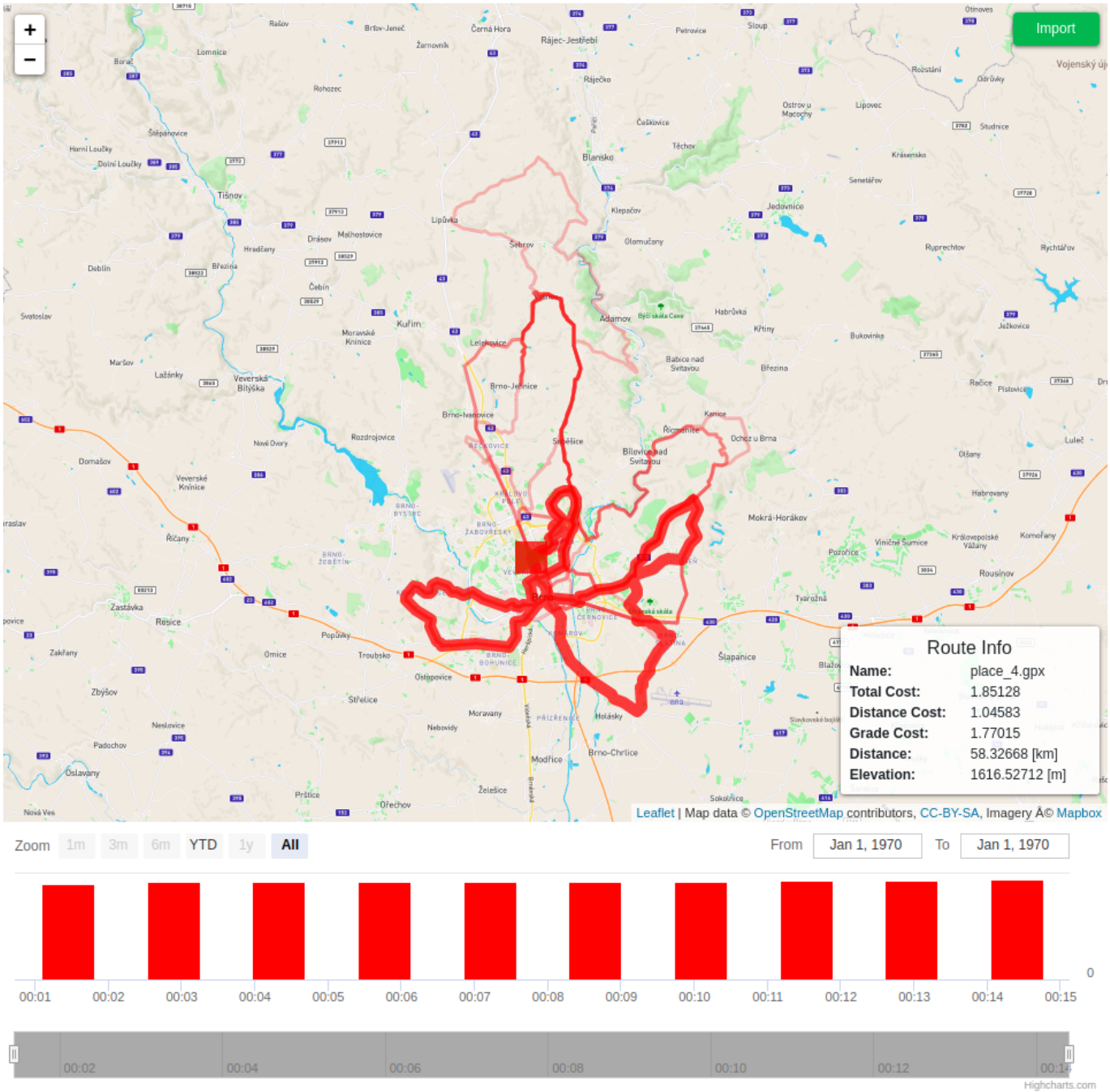

Figure 5: Moravský kras – 61 km, 3 hills, 2 points

The results of the first evaluation case. A route is hovered over and an info panel is shown.

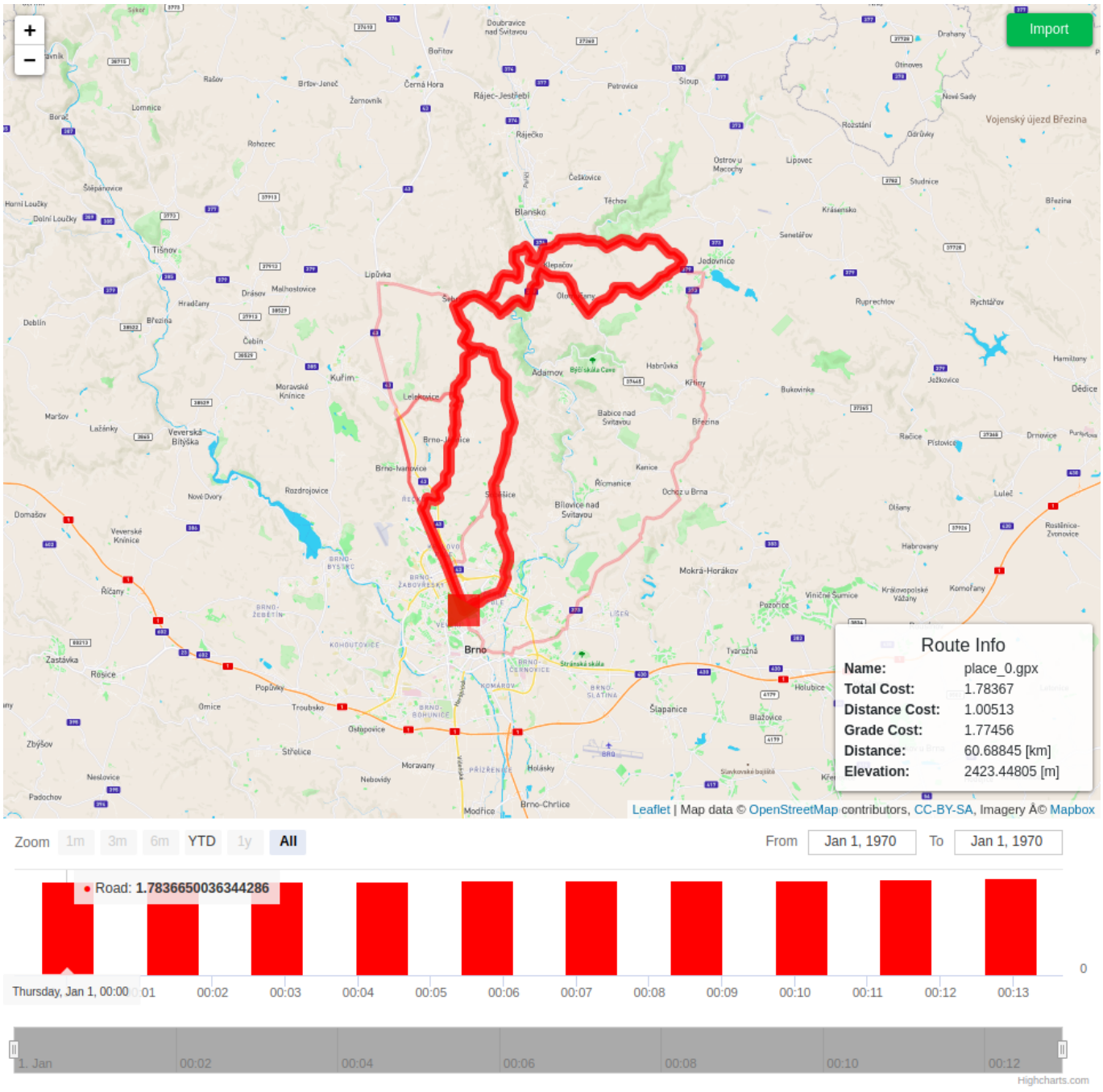

Figure 6: Moravský kras – 61 km, 3 hills, 3 points

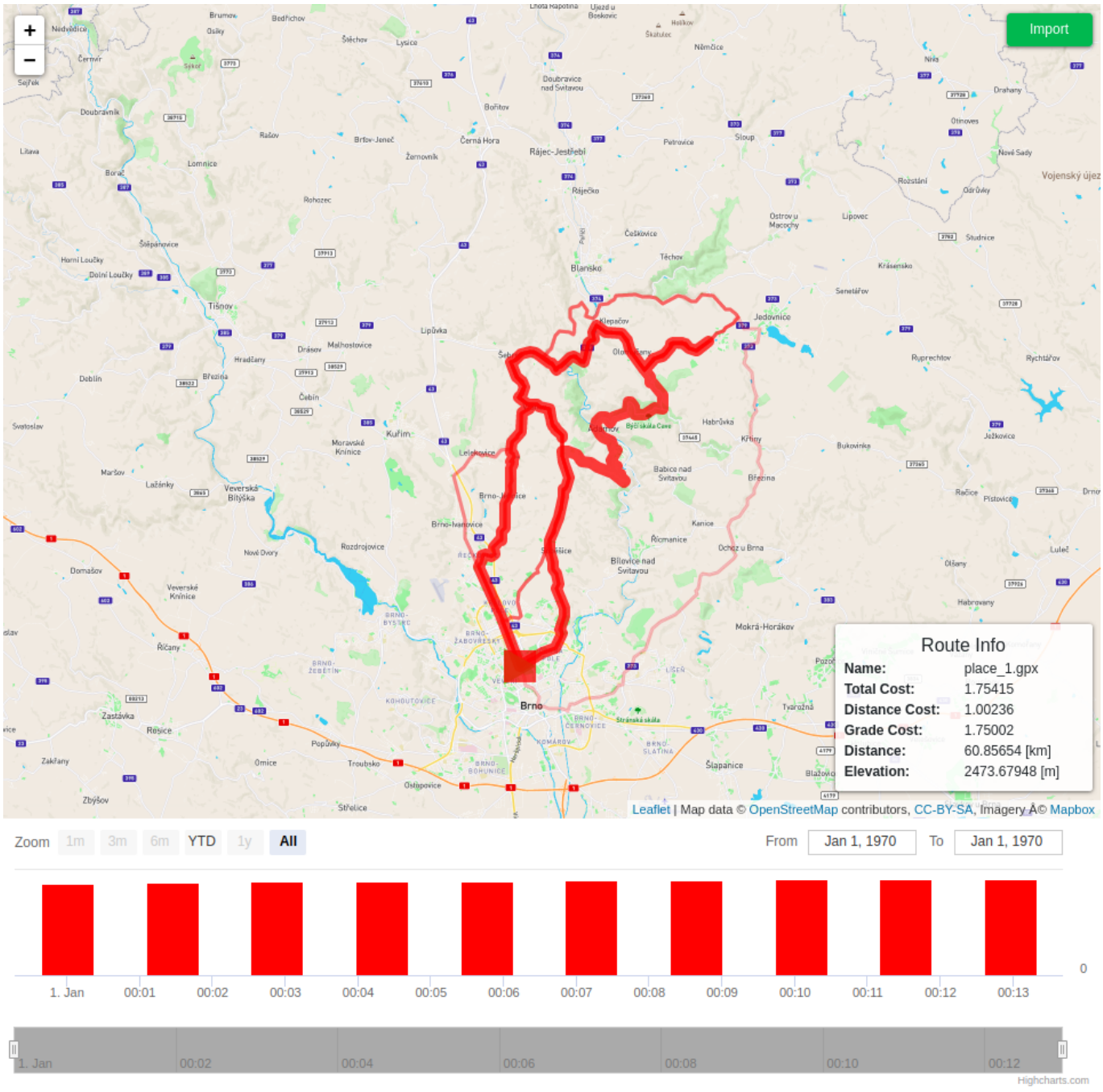

Figure 7: Moravský kras – 61 km, 3 hills, 4 points

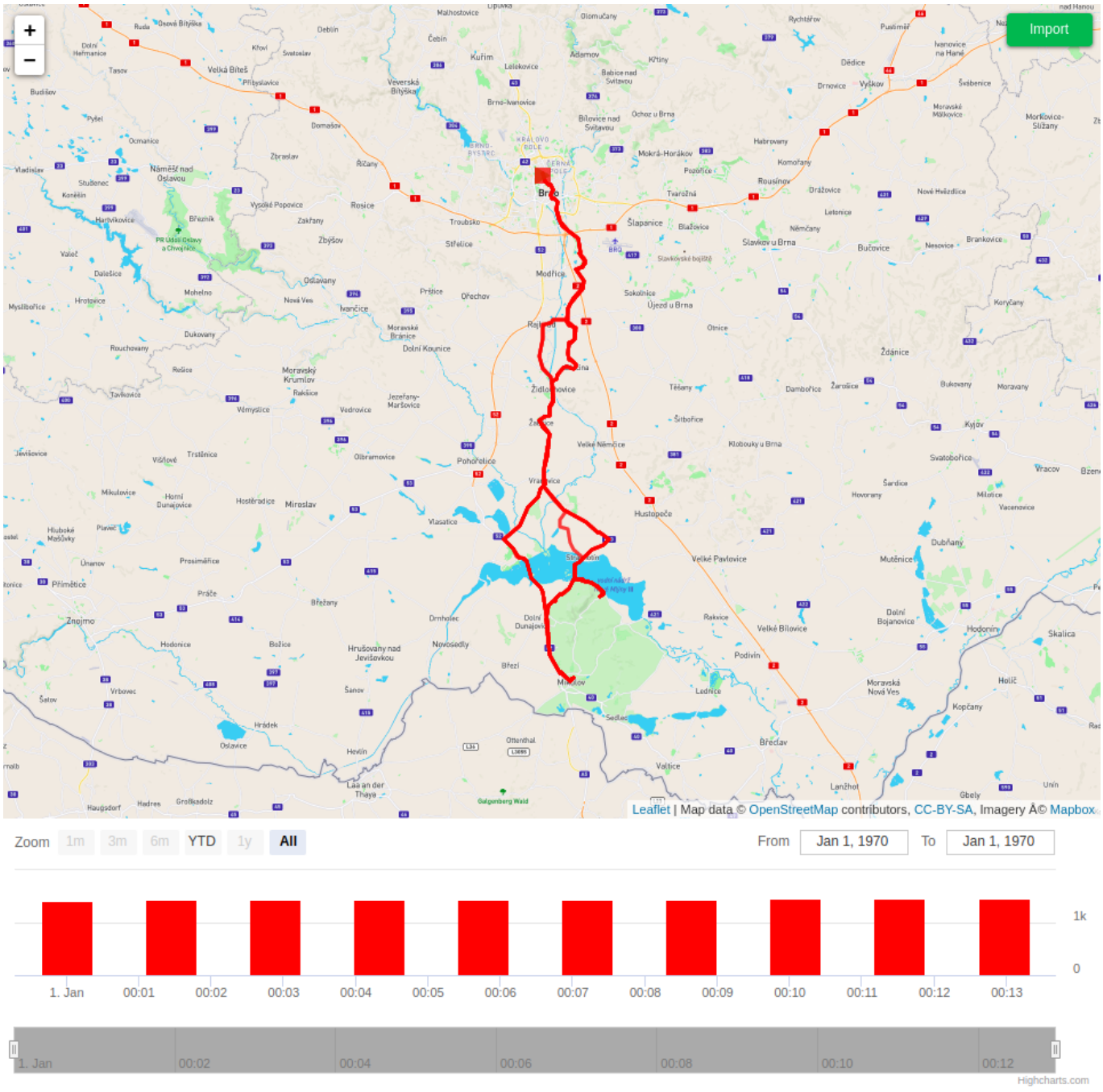

Figure 8: Pálava – 110 km, Mikulov, Pavlov, 0 hills, 4 points

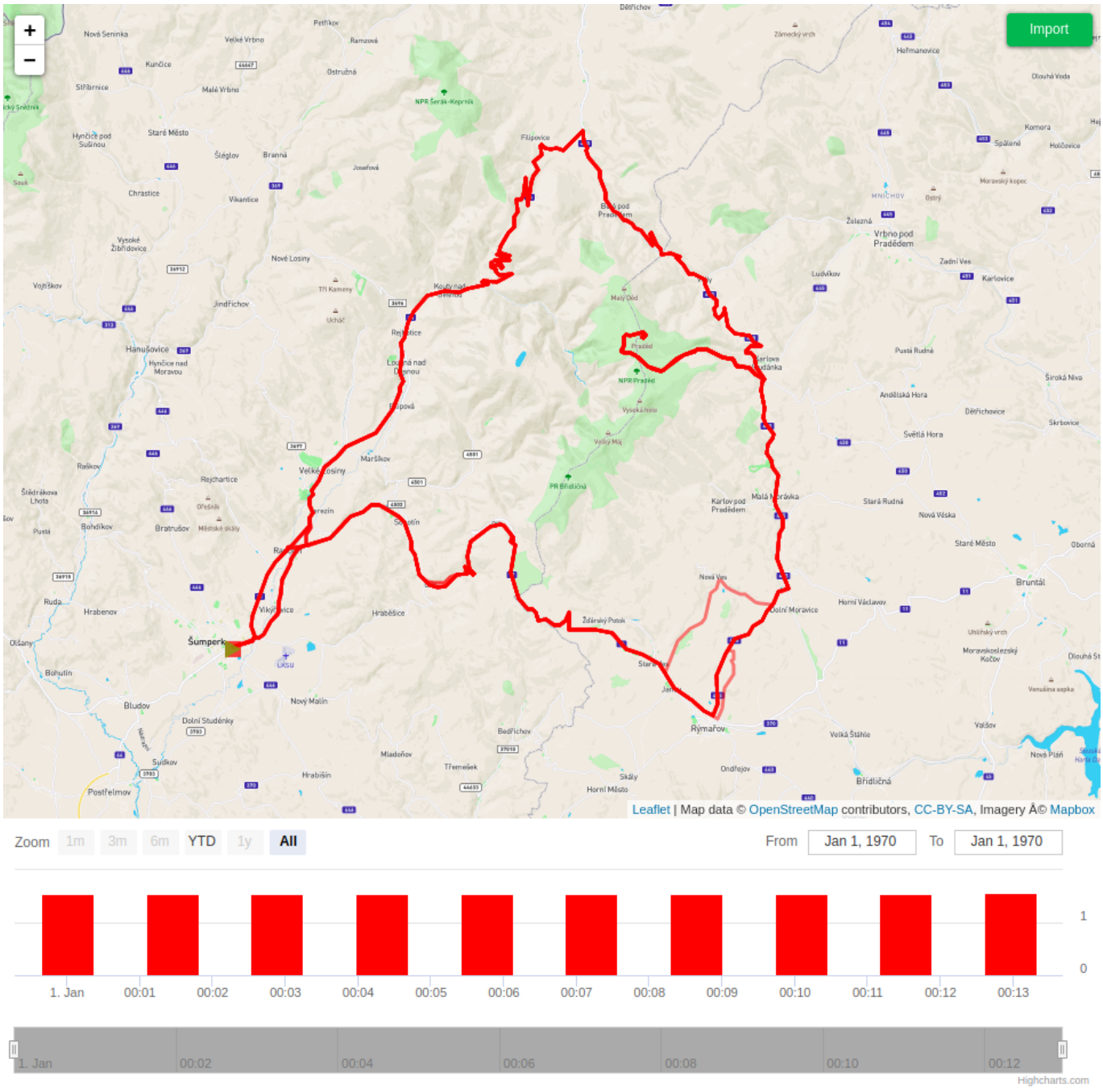

Figure 9: Jeseníky – 125 km, Praděd, 5 hills, 3 points

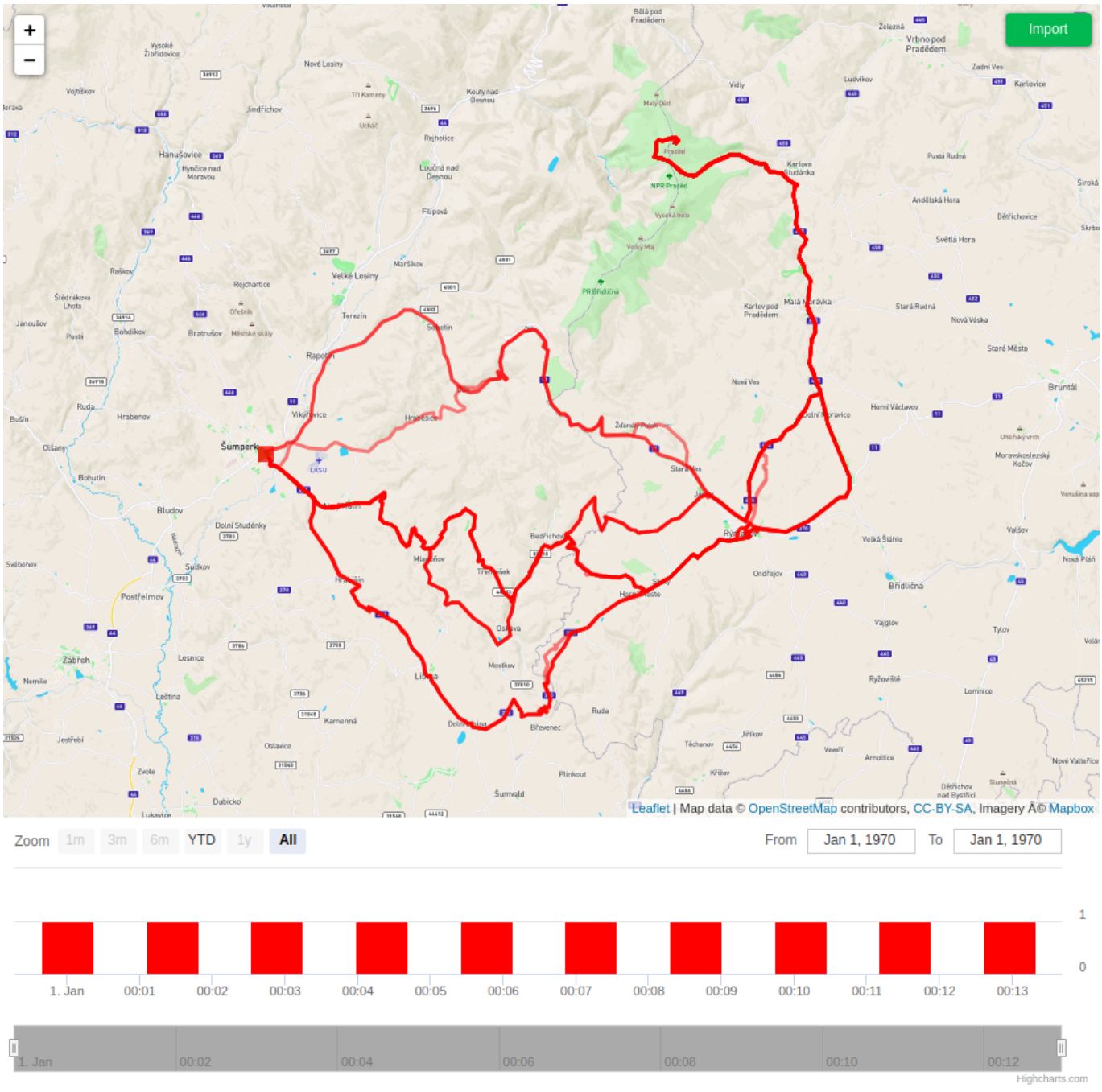

Figure 10: Jeseníky – 125 km, Praděd, 5 hills, 3 points, no grade function

Glossary

angle Measure of elevation change defined as $arctan(\frac{\Delta e}{\Delta d})$.

directed way A way instance with defined direction.

grade Measure of elevation change defined as $\frac{\Delta e}{\Delta d} \cdot 100$.

junction node A node that is used by more than one way, or it is used by exactly one way, either at its start or an end.

node A point defined by WGS84 coordinates and elevation.

route A sequence of linked directed ways.

slope Measure of elevation change defined as $\frac{\Delta e}{\Delta d}$.

way A sequence of connected nodes, either one-way or with no predetermined direction.