Institute of Formal Methods in Computer Science

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Multi-criteria Bicycle Routing

Florian Barth

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Stefan Funke
Supervisor:	Prof. Dr. Stefan Funke
Commenced:	November 15, 2017
Completed:	April 13, 2018

Abstract

This thesis considers the problem of finding alternative routes by weighting multiple metrics. This approach has the benefit that it always yields a Pareto optimal path. These routes have to be then checked against a similarity measure to ensure that they are different enough for a user to consider them helpful. The Splitting Algorithm designed in this thesis uses a geometric intuition to explore the possible weightings and find good alternative routes heuristically. To achieve fast query times although many routes need to be calculated and compared, the algorithm builds upon the multi-criteria version of the contraction hierarchies speed-up technique. In an implementation for cyclists with three metrics, the algorithm found up to thirteen routes that share less than 50% of their length and four or five routes in half the cases.

Kurzfassung

Diese Arbeit beschäftigt sich mit dem Finden von Alternativrouten durch das Gewichten mehrerer Metriken. Dies hat den Vorteil, dass immer Pareto-optimale Routen gefunden werden. Die Routen werden dann durch ein Ähnlichkeitmaß geprüft, um sicherzustellen, dass sie unterschiedlich genug sind, um für einen Benutzer hilfreich zu sein. Der Splitting Algorithmus, der in dieser Arbeit entworfen wurde, nutzt eine geometrische Anschauung, um auf heuristische Weise die möglichen Gewichtungen zu explorieren und gute Alternativrouten zu finden. Um kurze Laufzeiten zu erreichen, obwohl viele Routen berechnet und verglichen werden müssen, baut der Algorithmus auf der multi-kriteriellen Variante der Contraction Hierarchy speed-up Technik auf. In einer Implementierung für Fahrradrouten mit drei Metriken fand der Algorithmus bis zu dreizehn Routen, die weniger als 50% ihrer Distanz gemein hatten und vier oder fünf Routen in der Hälfte aller Fälle.

Contents

List of Figures

List of Tables

List of Abbreviations

CSP Constrained Shortest Path. 14

LP linear program. 19

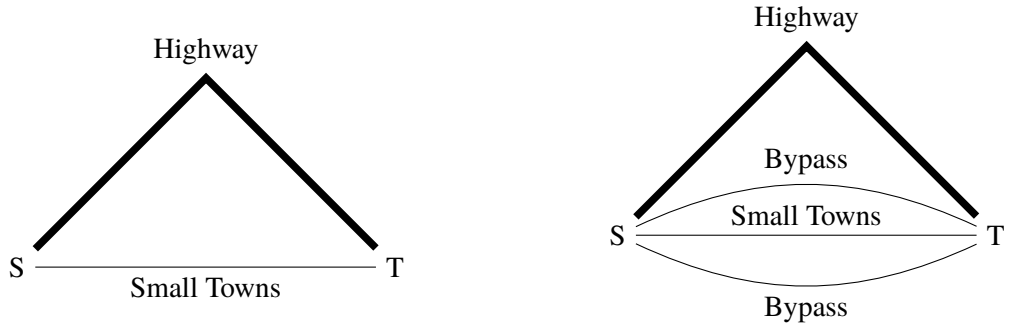
MARS Maximum Alternative Route Set. 23

MSAR Minimally Similar Alternative Routes. 23

1 Introduction

Finding the shortest route between two points in a road network is a straightforward task. Dijkstra's algorithm does this by exploring from the start point taking every outgoing edge and labeling the nodes it finds on the way with the best possible distance with which it reached them. It keeps a min-heap of all reached nodes which sorts the nodes by distance. The algorithm then pops the node with the current minimal distance to explore further until it either finds the target node or explores the graph entirely. This can be done in the very same way for the fastest or most fuel-efficient path. [Dij59]

But sometimes that does not lead to the paths a user might want to use. In the example presented in Figure 1.1a, the fastest route takes a considerable detour by using the highway and in the end gains little time over the shortest route. The shortest route on the other hands leads through the streets of small towns, which can be unnerving and slow to navigate. Therefore, it might be more in the interest of the user to make different trade-offs. Maybe the user would have instead taken a bypass road which avoids the streets of small towns but is not an as large detour as the highway route. Such alternatives are indicated in Figure 1.1b. Having alternative routes that do not only optimize one metric, can, therefore, be beneficial. It gives the user more choice to make the trade-offs suitable for him/her. Furthermore, looking for routes that optimize certain trade-offs, may give rise to using metrics that would not work well in a single metric case. Minimizing the use of non-bicycle paths or height ascent for bicycles in isolation can lead to large and inconvenient detours. For example going all the way around a hill, when there is a short route with small height ascent at the side of the hill. Mixing those metrics with the shortest distance to some degree, on the other hand, may lead to interesting combinations.



(a) The shortest and the fastest route between S and T.

(b) 1.1a extended with alternative routes that make different trade-offs.

Figure 1.1: Comparison between routes optimal by one metric and routes that make trade-offs between those metrics.

1.1 Related Work

In [DW09], the authors search for Pareto optimal routes between two nodes s and t by using an adapted version of the SHARC speed-up technique. In a multi-criteria setting, a path is called Pareto optimal if no other path has less or equal costs in every metric. The problem of finding all Pareto optimal routes is NP-hard in general. Therefore, they limit the search by using a main metric in which they restrict how much the minimal cost can be overrun. They achieve speed-ups which range from two to four orders of magnitude in comparison to modified Dijkstra algorithm for two to four metrics.

Abraham et al. [ADGW13] use bidirectional Dijkstra searches to find alternatives to the shortest path. The candidates they consider are called "via paths". A via path P_v from s to t consists of the shortest paths s–v and v–t. For a via path to be an admissible alternative route, it has to satisfy three conditions. An alternative route must not share too much of its length with the shortest path. Furthermore, it needs to be "locally optimal" which means that subpaths of P that are below a certain length need to be shortest paths to avoid detours in the path. The third condition is that alternative routes need to have "uniformly bounded stretch". This condition means that each subpath is only allowed to be $1 + \epsilon$ times longer than the shortest path between its end vertices.

The approach of Bader et al. [BDGS11] is to build an alternative graph (AG) which encodes alternative routes between s and t. Every edge of this alternative graph corresponds to a path in the original graph. An AG is constructed by using existing methods, like the via paths from [ADGW13] or Pareto optimal routes form [DW09] to find alternative paths and add these paths to the AG. Similar to the criteria for admissible alternative routes used by Abraham et al. [ADGW13] three criteria are checked. They serve similar purposes as in the other approach. They limit the maximal length of alternative routes, discourage sharing and limit the complexity of the AG. These criteria are different from the former in that they have to apply to the AG as a whole and not to individual paths.

The Constrained Shortest Path (CSP) Problem provides another way to think about alternative routes. It looks for the shortest route according to one metric that does not exceed the given constraint in another metric. Finding a CSP is NP-hard [MZ00]. Storandt [Sto12] modifies the contraction hierarchies technique and optionally combines it with arc-flags to allow fast queries for CSPs in large route networks. The paper uses a label setting and a dynamic programming approach and reports significant speed-ups compared to the native counterparts.

1.2 Contribution

This thesis provides the following on the topic of finding alternative routes:

- Two new problem definitions are which that formalize the notion of finding alternative routes. The first problem definition focuses on having two very different routes, while the second concentrates on having a maximally large set of diverse routes.
- A heuristic algorithm is designed which works on top of the multi-criteria contraction hierarchy approach of [FLS17]. It uses the fast queries provided by multi-criteria contraction hierarchies to search for routes in the configuration space that meet the criteria of the given

problem definitions. At its core lies the splitting of the configuration space into smaller areas until this is no longer beneficial. A randomized algorithm is designed and used to verify the results.

- The Splitting Algorithm is experimentally compared against the randomized algorithm. In the experiments which use three metrics that are relevant for cyclists, the splitting algorithm generally finds more routes than the randomized alternative. Furthermore, it provides criteria when to stop the search.

1.3 Outline

In Chapter 2 the foundation of this thesis is described. Especially, Contraction Hierarchies and the multi-criteria adaption of the technique are explained. Also, similarity measures are formally defined to be used in the problem definitions and algorithms. Chapter 3 gives a detailed view of the problems the thesis is trying to solve and how this is approached. The parts of the Splitting Algorithm are set out together with the properties of the problem which make each piece necessary. In addition to that, information belonging to the implementation which was used to test the algorithm is given. The results of testing the algorithm against a randomized variant based on the same foundation are laid out in Chapter 4. Finally, in Chapter 5 the results are discussed, and ideas for possible future work are developed.

2 Preliminaries

2.1 Personalized or Multi-Criteria Route Planning

Typical route planning algorithms optimize for one metric at a time. They compute either the fastest, shortest or optimal according to some other criterion route.

Problem: Shortest Path Given a Graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$ and two vertices $s, t \in V$, find a path $p = (s, v_2, \ldots, v_n, t)$ from s to t such that $\sum_{v \in p} c(v)$ is minimal.

This problem is for example solved by Dijkstra's algorithm [Dij59]. The problem and the algorithm can be extended to work with multiple metrics simultaneously. This extension means that the cost function now maps edges to a vector $c : E \to \mathbb{R}^{+d}$ in which every component contains the cost of one metric. The goal of the problem changes too. It can either become a search for Pareto optimal routes where none of the resulting routes is dominated by any other in all metrics [Mar84] or a search for an optimal route according to a weighting of the metrics. Such a weighting can be achieved by supplying a configuration vector $\alpha = (\alpha_1, \ldots, \alpha_d)$ at query time. Let $p_{st,\alpha}$ denote the optimal path from s to t using the configuration α. The following conditions must hold for the components of the configuration vector:

$$\forall \alpha_i \in \alpha : 0 \leq \alpha_i \leq 1$$
$$\sum_i \alpha_1 = 1$$

This thesis is going to use the following problem definition for multi-criteria route planning:

Problem: Personalized Route Planning Given a Graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^{+d}$, two vertices $s, t \in V$ and a configuration $\alpha = (\alpha_1, \ldots, \alpha_d)$, find a path $p = (s, v_2, \ldots, v_n, t)$ from s to t such that $\sum_{v \in p} \alpha^T \cdot c(v)$ is minimal. [FLS17]

Note that the routes found by the Personalized Route Planning Problem are a subset of the Pareto optimal routes. If there exists a route p' which dominates a route p found by a personalized route planning algorithm, the weighted cost of p' would be smaller than that of p:

$$\sum_{v \in p'} \alpha^T \cdot c(v) \leq \sum_{v \in p} \alpha^T \cdot c(v)$$

By definition, the personalized route planning algorithm would find p' instead of p. Therefore, all routes found by a personalized route planning algorithm are Pareto optimal.

On the other hand, not all Pareto optimal routes can be found by such an algorithm. Figure 2.1 shows three edges which are all Pareto optimal because none of the other edges have lower costs in both metrics. However, while there exist configurations that make a and c optimal personalized routes, there does not exist such a configuration for route b.

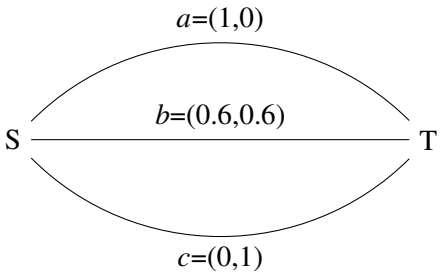


Figure 2.1: Graph with an edge (b) which is Pareto optimal but not optimal for any personalized route planning configuration.

2.2 Contraction Hierarchy

Contraction Hierarchy is a route planning technique which facilitates preprocessing to improve query times in contrast to the Dijkstra algorithm significantly. In the preprocessing phase, the nodes of the graph are contracted. To contract a node, it and its adjacent edges are removed from the graph, and shortcut edges are inserted if necessary to maintain shortest path distances. Shortcut edges replace two edges adjacent to the node that is contracted if and only if they form the only shortest path between their other adjacent nodes. That is, if the vertex u is currently being contracted and is adjacent to the vertices v and w, a shortcut $s = (v, w)$ with costs $c(s) = c((v, u)) + c((u, w))$ will be inserted if and only if v–u–w is the only shortest path from v to w. An example shortcut insertion is depicted in Figure 2.2. The order of the node contractions determines the hierarchy of the nodes. Nodes that are contracted later are of a higher level. The node order can be determined in different ways. It is possible to choose at random or choose nodes next that optimize for a specific criterion. For example the “Edge Difference” which is the difference between the number of new shortcut edges and the number of deleted edges when contracting the node. As a node contraction only affects edges incident to one node, nodes that are not adjacent to each other can be contracted at the same time/step. [Gei08]

The query to determine shortest paths consists of a modified bidirectional Dijkstra algorithm. The forward search considers only edges that lead to nodes of a higher level while the backward search only considers edges, coming from nodes of a higher level. The search space of the forward search is called the “upward graph”, while the search space of the backward search is called “downward graph”. The searches eventually meet in the node with the highest level on the shortest path. Through the hierarchy in the contraction order, the search space (the nodes visited for one query) needed is greatly reduced. This reduction in search space increases the performance significantly in comparison to a standard unidirectional Dijkstra search. After finding the shortest path in the augmented graph, the shortest path in the original graph can be obtained by recursively unpacking the used shortcuts. [GSSD08]

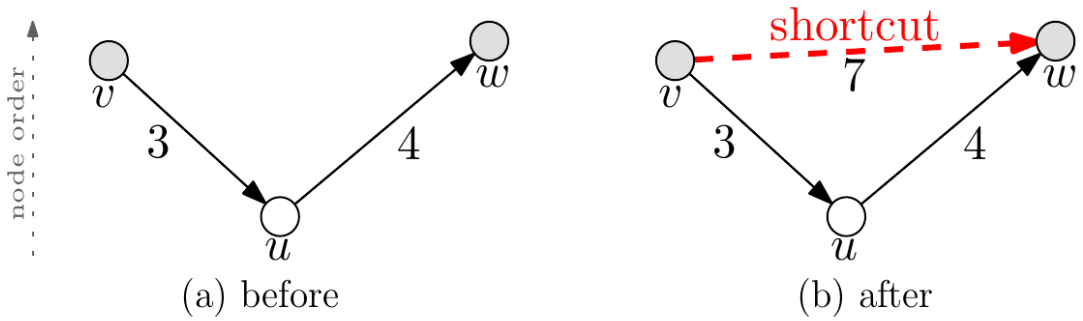


Figure 2.2: Adding a shortcut when contracting u. [Gei08]

2.3 Multi-Criteria Contraction Hierarchy

When contraction hierarchy is applied to a personalized route planning graph, the preprocessing phase gets more complex. To determine if a shortcut needs to be inserted for the path p^* between u and w, a configuration α must be found for which p^* becomes the only optimal path from u to w. If such an α cannot be found, the insertion of the shortcut is not necessary. An example with two metrics can be found in Figure 2.3. When the node B is contracted, there is no need to create a shortcut from A to C because there is no valid $\alpha = (\alpha_1, \alpha_2)$ for which:

$$\alpha_1 \cdot (3 + 5) + \alpha_2 \cdot (4 + 3) \leq \alpha_1 \cdot (2 + 3) + \alpha_2 \cdot (4 + 2)$$
$$8\alpha_1 + 7\alpha_2 \not\leq 5\alpha_1 + 6\alpha_2$$

For the path from A to D on the other hand, the configuration $\alpha = (1, 0)$ makes the path A—B—D the only shortest path from A to D:

$$\alpha_1 \cdot (3 + 4) + \alpha_2 \cdot (4 + 2) \leq \alpha_1 \cdot 8 + \alpha_2 \cdot 2$$
$$7\alpha_1 + 6\alpha_2 \leq 8\alpha_1 + 2\alpha_2$$
$$7 \cdot 1 + 6 \cdot 0 \leq 8 \cdot 1 + 2 \cdot 0$$
$$7 \leq 8$$

One way of finding such α is to create a linear program (LP) which computes a configuration for which the path p^* from u to w is optimal. To achieve this, the LP needs to contain constraints that make all solutions where p^* is not optimal infeasible. If there are d metrics and the set P contains all paths from u to w, the constraints of the LP are the following:

$$\forall p \in P: \qquad \alpha^T \cdot c(p^*) - \alpha^T \cdot c(p) \leq 0 \tag{2.1}$$

$$\sum_{i=1}^{d} \alpha_i = 1 \tag{2.2}$$

$$i = 1, \ldots, d: \qquad \alpha_i \geq 0 \tag{2.3}$$

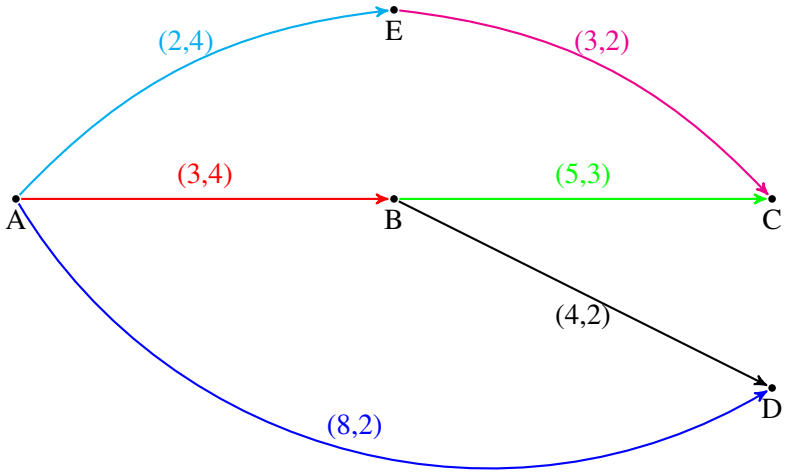


Figure 2.3: Example graph for node contraction in multi-criteria contraction hierarchy.

The constraints of form 2.1 ensure that the path cost of p^* is less than the cost of any other path from u to w while the constraints of form 2.2 and 2.3 ensure that the conditions for a valid configuration are met. As such an LP can have exponentially many constraints, it is not practical to use it like this. The LP can, however, be efficiently used and constructed incrementally. Each time the partially constructed LP is solved, a configuration is obtained. By running Dijkstra with this configuration, it is possible to check if the entire LP would be violated. If the Dijkstra run results in p^*, a shortcut needs to be inserted. If another path p is obtained, this path belongs to the set P and is used to add a new constraint of form 2.1 to the LP. If the LP is infeasible, no shortcut needs to be inserted as p^* can never be the optimal path from u to w. [FLS17]

When trying to contract the path A—B—D from Figure 2.3, the configuration $\alpha = (0.5, 0.5)$ might be obtained. Running Dijkstra with this α returns the path A—D, which leads to a new constraint of form 2.1, which in turn leads the LP to find the correct configuration.

2.4 Comparing Alternative Routes

Just because two routes are different, they do not have to be suitable alternative routes. Often, the second shortest path differs from the shortest path only by one or two edges. It is therefore vital to have one or more similarity measures which given two routes determine how similar or different the routes are. Let P be the set of all paths p from s to t and a similarity measure be a function $\delta : P \times P \rightarrow [0, 1]$ where $\delta(p_1, p_2) = 1$ means that $p_1 = p_2$ and $\delta(p_1, p_2) = 0$ that the paths are maximally dissimilar under δ. Good candidates for implementing similarity measures are for example sharing between the paths or a geometric distance like the fréchet distance. Using sharing as similarity measure conforms to the notion that alternative paths should use mostly different roads. It is also very cheap to compute.

The fréchet distance, on the other hand, can intuitively be described as measuring the shortest leash necessary such that a dog and his owner could walk on the different routes. As it allows both owner and dog to move at any speed (without going backward), it needs at least quadratic time in the number of edges to calculate [Bri14]. It is useful to differentiate between alternative routes that use parallel roads and ones that use a genuinely different way to the destination.

In this thesis, the ratio of the length that is shared by both paths to the length of the longer path is used as similarity measure. The sharing similarity measure was chosen because it is faster to compute and is easily scaled between 0 and 1.

3 Triangle Splitting

This chapter describes the approach used to generate alternative routes in this thesis. Section 3.1 provides a high-level overview of the approach. Two problem definitions are formulated in Section 3.2 which offer different goals when searching for alternative routes. In Section 3.3 the Splitting Algorithm is presented and divided into parts. Each part is then explained in its own section. To test the effectiveness of the Splitting Algorithm, a randomized alternative algorithm is described in Section 3.4. The final section of this chapter is about the implementation done for this thesis.

3.1 General Idea

Through the application of multi-criteria route planning, different routes can be generated for every node combination. Each of these routes is optimal respective to the configuration used to create the route. The idea is to explore the configuration space in order to find configurations that produce sufficiently different routes. As one query for alternative routes consists of many route finding queries, multi-criteria contraction hierarchy is used to keep query times small. The discovered routes are then compared by a similarity measure to decide which areas of the configuration space should be explored further.

3.2 Problem definition

There are different ways to view the generation of alternative routes. A formal problem definition is given for two distinct viewpoints of the problem. The first is about finding two alternatives which differ as much as possible, while the second is concerned with finding as many sufficiently different routes as possible between two nodes in a graph.

Problem: Minimally Similar Alternative Routes (MSAR) Given a Graph $G = (V, E)$, two nodes $s, t \in V$, and a similarity measure δ, find two configurations $\alpha = (\alpha_1, \ldots, \alpha_d)$ and $\alpha' = (\alpha'_1, \ldots, \alpha'_d)$ that minimize $\delta(p_{st,\alpha}, p_{st,\alpha'})$

Problem: Maximum Alternative Route Set (MARS) Given a Graph $G = (V, E)$, two nodes $s, t \in V$, a similarity measure δ and a threshold h, find a set of configurations $A \ni \alpha = (\alpha_1, \ldots, \alpha_d)$ such that $\forall \alpha, \alpha' \in A : \delta(p_{st\alpha}, p_{st,\alpha'}) < h$ and $|A|$ is maximal.

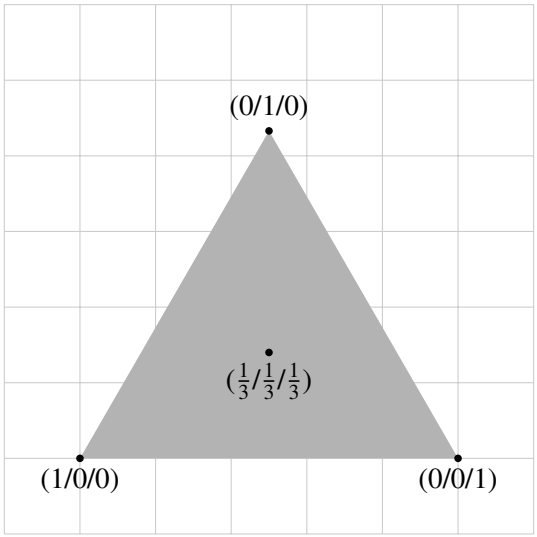


Figure 3.1: Configuration space for three metrics

3.3 The Splitting Algorithm

The goal of the Splitting Algorithm is to find α which create sufficiently different routes between s and t. When using three metrics, the space of the possible configurations α can be viewed as an equilateral triangle. Each point in this triangle represents one value of α. Figure 3.1 shows a configuration space for three metrics and marks the configurations which consider only one of the metrics (corner points) as well as the configuration which considers all metrics equally (centroid).

One approach to find candidate configurations is to split the triangle into three triangles by its centroid. Each of the resulting triangles will then have the centroid of the original triangles and two of its corners as corners. The split and the resulting triangles are depicted in Figure 3.2. This split can be applied recursively to all resulting triangles until a stopping criterion is reached.

The approach described here translates cleanly into higher dimensions. If d metrics are used, the initial shape will have d equilateral edges and in the first step be deconstructed into d triangles. From this point on the algorithm can continue in the way described in the following sections.

3.3.1 Stopping Criterion: Similarity Measure

While the splitting of the triangles continues, the area of the triangles gets smaller, and their corner points get closer together. As configurations that are close to each other have a higher chance of generating routes that are more similar to each other, it makes sense to stop splitting if the routes generated by all corners exceed a certain similarity threshold to the route generated by the centroid of the triangle. Let C be the set of Corners in the triangle, δ be a similarity measure, and h be a threshold for δ. Stop if $\forall c \in C : \delta(p_{st,\text{center}(C)}, p_{st,c}) \geq h$. The similarity measure used in this thesis is the ratio of the shared length between the routes to the length of the longer routes. This stopping criterion might not work for the triangles that contain two corners which lie on one of the edges of the original triangle. There are at least three of those triangles in every iteration. If the "outer triangles" of subsequent iterations are compared their "inner" corner moves toward the

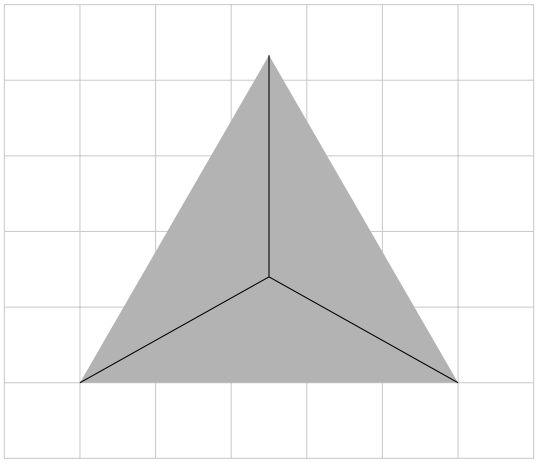


Figure 3.2: Triangle split into three parts

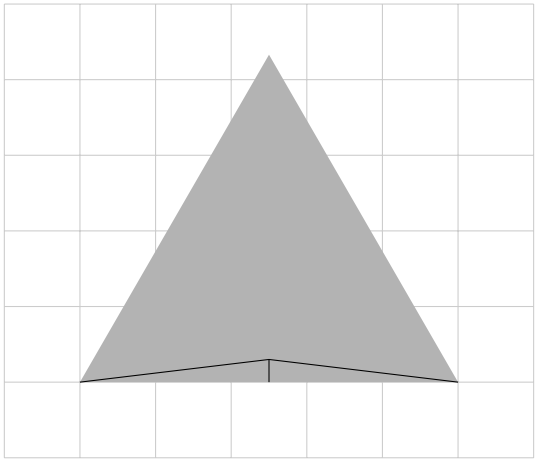


Figure 3.3: Outer triangle split in half

center of the outer edge. Therefore there is a lower bound to the distance between the centroid of those triangles and two of its corners. This can lead to situations where there always exists a corner with a better similarity measure than the threshold h. To prevent an infinite loop the triangle can be split in half at the outer edge instead of the center of the triangle. To decide whether such an "outer split", that is shown in Figure 3.3 needs to be applied, both the routes for the centroid and the one for the center of the outer edge are calculated. If the minimum similarity of the route corresponding to the center of the outer edge is bigger than the minimum similarity of the route corresponding to the centroid, the outer split should be applied. The reason for this is that splits should increase the similarity in the resulting triangles as much as possible.

3.3.2 Stopping Criterion: Number of Splits

The number of routes generated by triangle splitting can be quite large depending on the threshold used by the similarity measure criterion. As more routes are generated, the number of routes that are similar to other routes increases. A user might not need or want dozens of marginally different

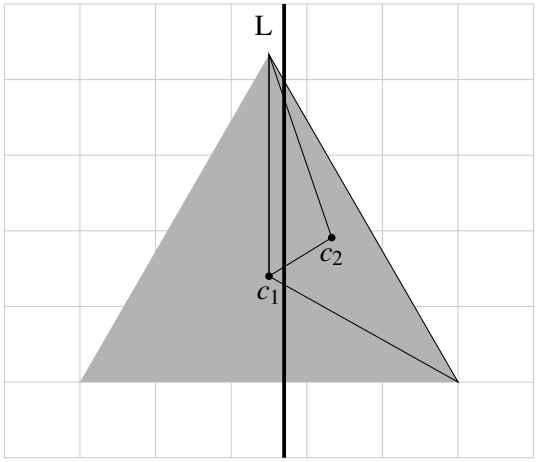


Figure 3.4: Triangle intersected by a line in configuration space that separates configurations that correspond to one of two dissimilar routes.

routes but a selection of a few significantly different ones. Therefore the triangle splitting can be stopped after a set amount of splits is reached. With only a fixed number of splits available, it is essential just to use the most promising splits. This can be achieved by using a priority queue which sorts the triangles by average similarity measure between the two original corners and the new one (centroid of original triangle).

3.3.3 Stopping Criterion: Repeating Routes

The discrete nature of road graphs can lead to situations where splitting a triangle neither reveals new routes nor increases the minimal similarity inside the triangle. Figure 3.4 shows an example of such a situation where c_1 is the centroid from the original triangle while c_2 is the centroid of one of the resulting triangles from a split. Assume that for the configurations contained in one triangle there exist only two routes q and r from s to t and that there is a line L in the configuration space which has the property that all configurations α left of L generate r and α to the right of L generate q. Assume further that q and r are less similar than the threshold h: $\delta(q, r) < h$. As L intersects the triangle, at least one of its corners is on a different side of L than the centroid. A split of this triangle creates at least one triangle which is intersected by L. To prevent the algorithm from further and further refining triangles it can be checked whether a previously not encountered route belongs to the configuration of the centroid. If this is not the case, the refinement of this triangle can be stopped. As this might prevent useful splits from happening, the stop can be deferred until n refinements which are ancestors of each other, yield no new route.

3.3.4 Stopping Criterion: Refinement Depth

The refinement depth of a triangle is the number of ancestors it has. The initial configurations space has refinement depth zero, and its children have a refinement depth of one. With each split, one triangle vanishes and three triangles with $\frac{1}{3}$ of the area are created (or two triangles with $\frac{1}{2}$ in case of an outer split). As the refinement depth increases, the triangle area and the likelihood of

finding new routes decreases. Furthermore, skinny triangles with one tiny angle get more frequently. Skinny triangles in the inner of the triangle are bad candidates for splits for the same reason detailed in Section 3.3.1 for skinny outer triangles. Therefore, triangles with a refinement depth greater than some d should not be split further.

3.3.5 Selecting Routes

The triangle splitting algorithm generates a large number of configurations and routes. By definition of the similarity measure stopping criterion, a certain threshold of similarity must be exceeded already to stop refining in an area of the configuration space. For that reason, the right routes need to be selected to fulfill the criteria of the problems defined in Section 3.2. In case of the MSAR problem, this means finding the two configurations with the lowest similarity measure.

$$\min_{\alpha,\alpha'} \delta(p_{st,\alpha}, p_{st,\alpha'})$$

For the MARS problem, on the other hand, one needs to find a set where no two routes have similarity measure greater than given threshold h. To ensure this, iterate over each not yet excluded configuration α and exclude all configurations α' for which $\delta(p_{st,\alpha}, p_{st,\alpha'}) \geq h$.

3.4 Random Alternative Routes

The algorithm presented in Section 3.3 uses a structured approach to find configurations which produce suitable routes. It is compared to a randomized alternative, to measure the effectiveness of the algorithm,. The randomized algorithm creates configurations and routes uniformly at random. Depending on the problem to solve, the algorithm either returns the two that defer most (MSAR) or all that mutually differ to at least a certain extent (MARS). For the results of the algorithms to be comparable, they should do a similar amount of work. This is achieved by using the actual amount of routes created by the Splitting Algorithm as the number of configurations that should be generated by the randomized algorithm.

3.5 Implementation

The algorithms described above were implemented for three metrics for cyclists: distance, elevation gain and road suitability for cyclists. The data for distance and the road suitability where extracted from OpenStreetMap [18b]. Distances were calculated with the haversine formula [Rob57] in meters between the two node coordinates of every edge. It calculates the beeline distance between two points on earth's surface. The road (un-)suitability was assigned by using the OpenStreetMap "highway" tags for road type in general and the explicit "bicycle" and "cycleway" tags [18a]. The elevation gain in meters was extracted from the Shuttle Radar Topography Mission dataset [FRC+07]. Data was extracted for different regions of Germany.

The C++ implementation can be found at [Bar18a]. It needs a C++17 compatible compiler (g++ 7 or clang 5 or greater) and the following libraries have to be installed:

- Coin-or CLP [HHSF17]
- Boost serialization [Ram04]
- Boost program_options [Pru]
- Boost filesystem [Daw15]

The implementation has the following features: read graphs in a text or binary format, contract graphs, save graphs in binary format, start a web interface for routing and finding of alternative routes. Graphs in the needed text format can be created with another tool built for this thesis which can be found at [Bar18b].

3.5.1 Scaling of Metrics

A good distribution of the interesting configurations over the configuration space is preferable for the user and the algorithm. If a metric has considerably smaller value range than the others, it will only make an impact on the found routes when its part of the configuration vector is relatively high. This would force the interesting configurations closer together in one corner of the configuration space. Therefore the values for road suitability range from 0.5 to 2 in steps of 0.25 and are multiplied by the length of the corresponding edge. This puts the road suitability metric in the same range as distance and fits the notion that riding a bike on unsuitable roads for a considerable distance is worse than for a short distance. The mapping of the value of the highway tag to multipliers is shown in Table 3.1. The height metric was scaled to decimeters for the same reason.

Highway Tag	Length Multiplier
cycleway	0.50
footway	0.75
path	0.75
pedestrian	0.75
platform	0.75
track	0.75
service	0.75
living_street	0.75
traffic_island	1.00
residential	1.00
unclassified	1.00
bridleway	1.25
road	1.25
tertiary_link	1.25
tertiary	1.25
secondary_link	1.50
secondary	1.50
primary_link	1.75
primary	1.75
other tags	2.00

Table 3.1: Multipliers used for different road types

4 Results

The experiments were run on the graph extracted out of the OpenStreetMap data of Baden-Württemberg. The graph has 9.6 million nodes and 17.5 million edges. It was contracted to 98.5% which increased the number of edges to about 85 million edges.

4.1 Experiment Setup

There are four parameters introduced in Chapter 3:

- Similarity Threshold
- Number of Splits
- Maximum Refinement Depth
- Number of Repeating Routes in descendant triangles

For each of these parameters, different values need to be checked for their effect on the algorithm. All tested values are listed in Table 4.1. For each value combination listed in the table, 40 randomly chosen node combinations are used to run the Splitting Algorithm and the randomized alternative described in Section 3.4. The values were chosen to represent different use cases for the algorithm. The chosen similarity thresholds range from 30% to 70% to encompass most reasonable similarities that users might want. The other three parameters all have a lower end between one and up to five to represent parameters one would choose to limit query time measured in the number of totally calculated routes and a higher end with 20 and 40 to see how query times and resulting set size increase.

The Splitting Algorithm is given the parameters below, and the following values are recorded: The set size $|A|$ found, the minimum similarity between any two routes, and how many routes were generated totally. The randomized algorithm is then run with the same threshold parameter and the number of totally generated routes. The same values are reported for it.

4.2 Quality of Algorithm Output

For the MSAR problem, it is important to find maximally different routes. Figure 4.1 contains a violin plot which shows the distribution of the lowest obtained sharing ratio between two paths, separated by algorithms. For both algorithms, values from the whole value range 0.0 to 1.0 are present. This is to be expected, as the source and target nodes define the best possible values that can be achieved. It is possible that only one optimal route for all α exists between such two nodes, forcing the lowest possible sharing value to be 1.0. The output of both algorithms is more

Similarity Threshold	# Splits	Max. Refinement Depth	Repeating Routes
30	1	1	1
40	2	2	2
50	3	3	3
60	4	4	20
70	5	5	40
	20	20	
	40	40	

Table 4.1: Values tested for the parameters of the Splitting Algorithm

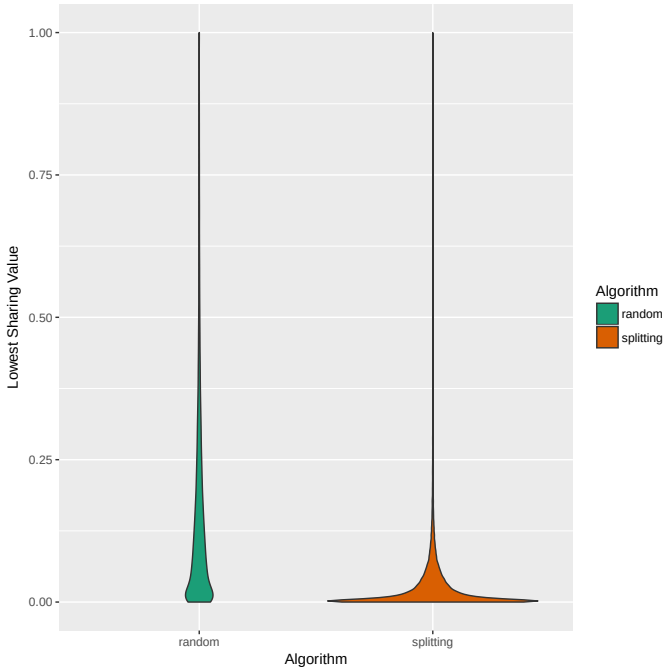


Figure 4.1: Violin plot showing the distribution of the lowest sharing ratios obtained by the splitting and randomized algorithm.

concentrated on the lower end of the spectrum. The concentration of the results of the randomized algorithm slowly decreases as the sharing value increases from 0.0 to 0.37. At this point, the graph is so thin that values above can be considered outliers. The Splitting Algorithm, on the other hand, has a strong concentration at the very bottom of the graph and it declines rapidly until a value of about 0.12 is reached. This marks the point beyond which points can be considered outliers.

The graph shows that both algorithms tend to produce routes with low sharing ratio rather than high ones. In contrast to the randomized algorithm, the Splitting Algorithm has a significantly higher likelihood to generate routes that share less than 10% of their distance. It is, therefore, better suited to solve the MSAR problem than the randomized algorithm.

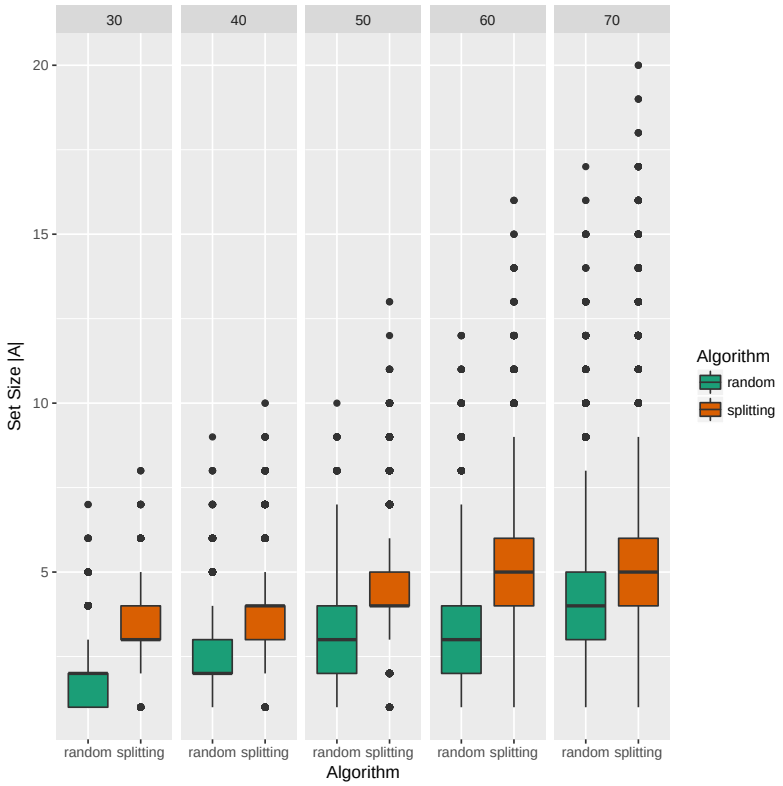


Figure 4.2: Boxplots showing the distribution of the set sizes $|A|$ obtained with the different thresholds (indicated at the top) by the splitting and randomized algorithms.

Finding many different routes is the objective of the MARS problem. Figure 4.2 contains boxplots that visualize the distribution of the set sizes $|A|$ obtained when running each of the algorithms with different similarity thresholds. For both algorithms, the maximum number of routes that can be found increases as the threshold increases. Also, the lower and upper quartile, which encompass the middle half of the values, increase. Three of the five boxes of the Splitting Algorithm contain only two set sizes, while the last contains three. Similarly, two of the boxes of the randomized algorithm contain two set sizes and the other three contain three. Note that for the thresholds from 30% to 60% the boxes of the Splitting Algorithm are above the boxes of the randomized one. Likewise, the maximum outlier of the Splitting Algorithm is higher than the one of the randomized algorithm for every threshold. Only at the highest threshold of the experiments, the boxes overlap. Also increasing the threshold beyond 60% does not increase the expected set size for the Splitting Algorithm anymore, but does increase the maximum achievable set size.

The results depicted in Figure 4.2 indicate that the Splitting Algorithm produces significantly more routes than the randomized algorithm for the similarity thresholds 30%, 40%, 50% and 60%. This holds true for the expected case as well as for the maximum outliers that may be found. Furthermore, one can expect to get three to four or four to five routes in most cases depending on the threshold. At the same time, a higher threshold enables set sizes up to ten, thirteen or sixteen to be obtained for 40%, 50%, and 60% respectively.

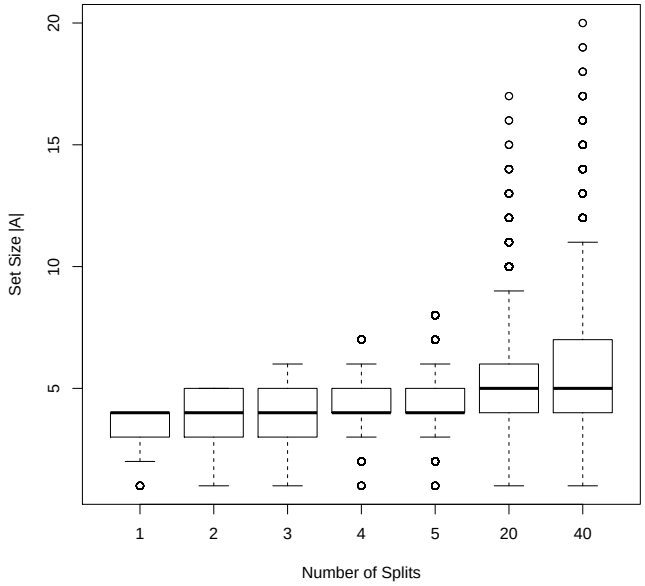


Figure 4.3: Boxplot showing the distribution of set sizes $|A|$ obtained with the Splitting Algorithm using different maximum numbers of splits.

4.3 Parameters

It is essential to inspect the impact of the parameters on the output of the algorithm. The investigation will focus on the effects that the parameters have on the MARS problem. The plots presented in Figure 4.2 already showed that a higher threshold at least up to 60% leads to more routes being found between two nodes.

4.3.1 Maximum Number of Splits

The next parameter is the maximum number of allowed splits. The distribution of set sizes $|A|$ under different amounts of splits are displayed in Figure 4.3. While the median set size remains at four between one and five splits and only increases to five for 20 and 40 splits, the maximum set size obtainable rises with the number of splits.

This rise in the number of obtainable routes does not come without a cost. In Figure 4.4 one can see a graph of the total number of routes calculated, given a certain maximum number of splits. With only one split available the algorithm will calculate four routes most of the time. When started with two or three splits, the number of routes calculated rises to six and eight respectively. When the parameter is increased even further, the number of calculated routes start to vary more. Five splits lead to between nine and twelve routes. When given 20 or 40 splits the algorithm calculates any number of routes up to 41 or 66. As the number of route calculations determines the runtime of the algorithm, this means more allowed splits increase the variance in expected runtimes a lot, while raising the expected set size only slightly.

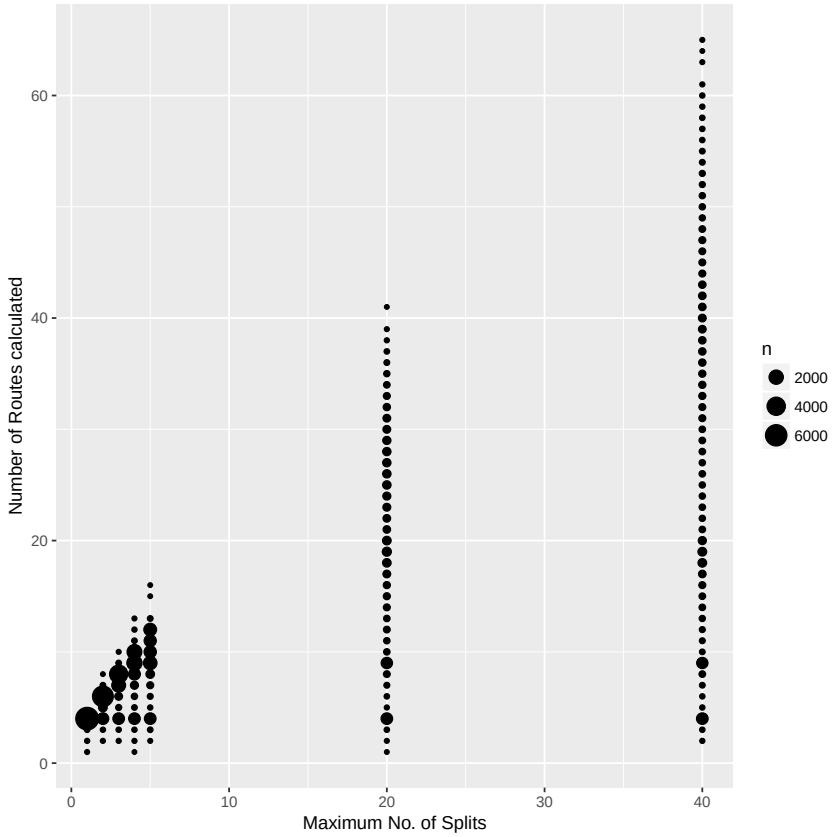


Figure 4.4: Graph showing the distribution of totally generated routes with the Splitting Algorithm using different maximum numbers of splits. Bigger dots mark a higher number of occurrences for a route count.

The data from the Figures 4.3 and 4.4 suggest that a value of four allowed splits makes a reasonable trade-off for the given metrics. On the one hand, it has a similar output regarding set size as five maximum splits. On the other side, the cost concerning the number of calculated routes lies closer to those of the queries with three splits. Increasing the number of splits to really high values leads to long runtimes and is only valuable if a high number of routes shall be found. For an end user application, it is probably not sensible.

4.3.2 Maximum Refinement Depth

Figure 4.5 displays one dot plot of the number of routes needed to calculated to achieve a specific set size per maximum refinement depth. Naturally, a refinement depth of one limits the possible set size to four routes belonging to the corners of the original triangle and their centroid. When the maximum refinement depth increases the set size and the number of routes that are calculated increase, too. The highest set size of 20 is encountered only for a refinement depth of four and five. The highest route count of 65 appears for a refinement depth of five and forty. A value of three reaches its maximum values when it calculates 20 routes to return 13 different ones, while a value of four already leads to 46 routes of which 20 are different.

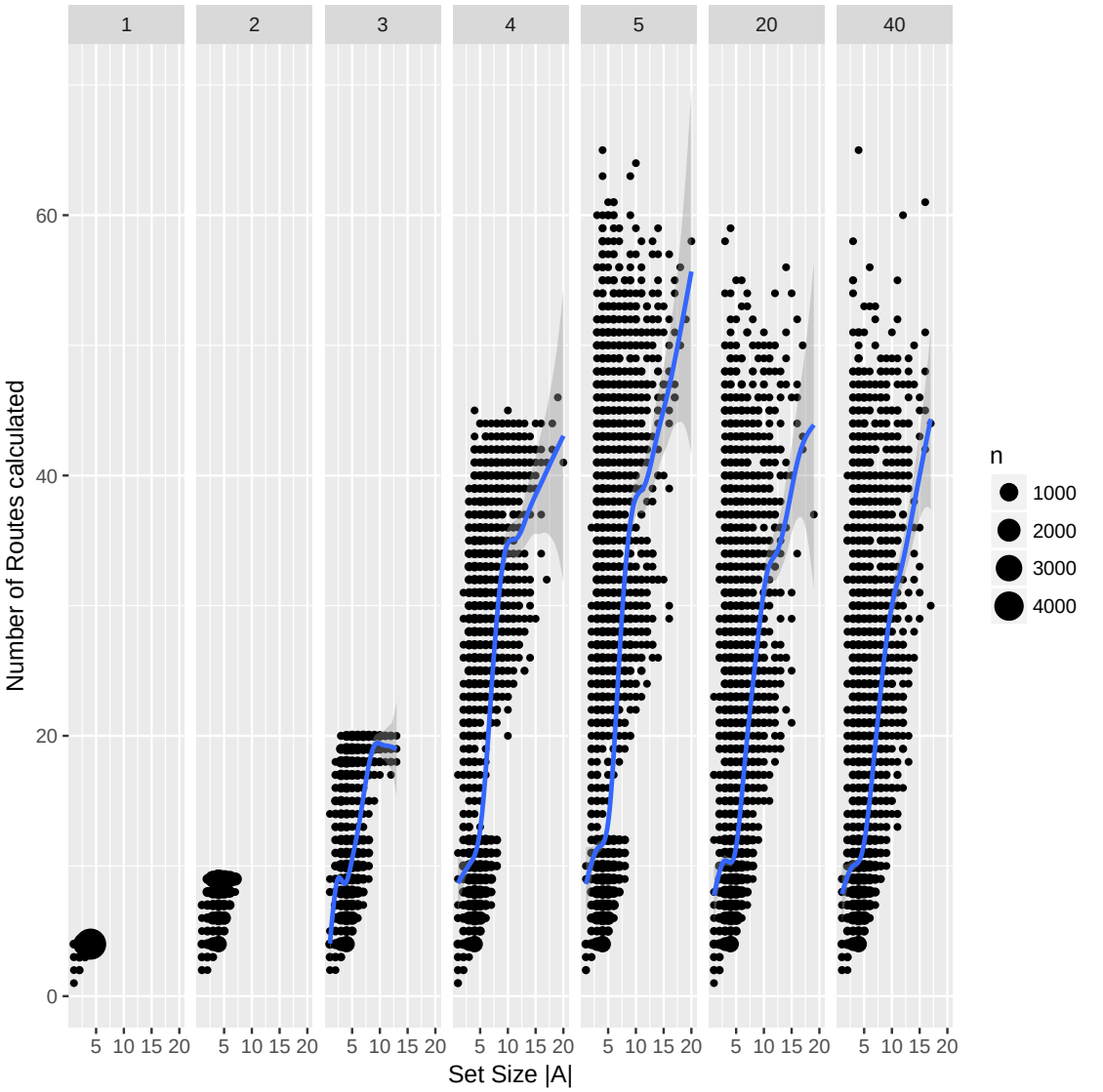


Figure 4.5: Dot plot of set sizes $|A|$ and number of calculated routes per different amounts of maximum refinement depth (indicated at the top of the plots). The blue line shows the trend of data with its confidence interval displayed in grey.

This stopping criterion was designed to decrease the number of unnecessary calculated routes in favor of hopefully more useful ones. That this goal was achieved can be seen in the data because the highest refinement depth values do not create higher set sizes than the lower ones. For the given graph, three seems to be an excellent choice for the maximum refinement depth. It limits the runtime to 20 route calculations and still obtains up to 13 different routes. If higher set sizes are desirable, a value of four can be chosen. This doubles the runtime for some queries but increases the maximum set size to 20.

4.3.3 Maximum Number of Repeating Routes

The stopping criterion “number of repeating routes” was designed to stop the Splitting Algorithm from refining triangles over and over, that do not provide new routes. In Figure 4.6 violin plots are presented that show the distribution of the number of found routes depending on the maximum number of repeating routes. There is one plot for every allowed splits value. As one would expect, the number of routes increases with the number of splits. The distributions in each of the violin plots are remarkably similar.

The similar distributions indicate that it doesn’t matter how many repeating routes are awaited to break the refinement of a triangle. The situations which were the reason for this stopping criterion either do not occur often enough in the graph used for testing or are already taken care of by the other stopping criteria.

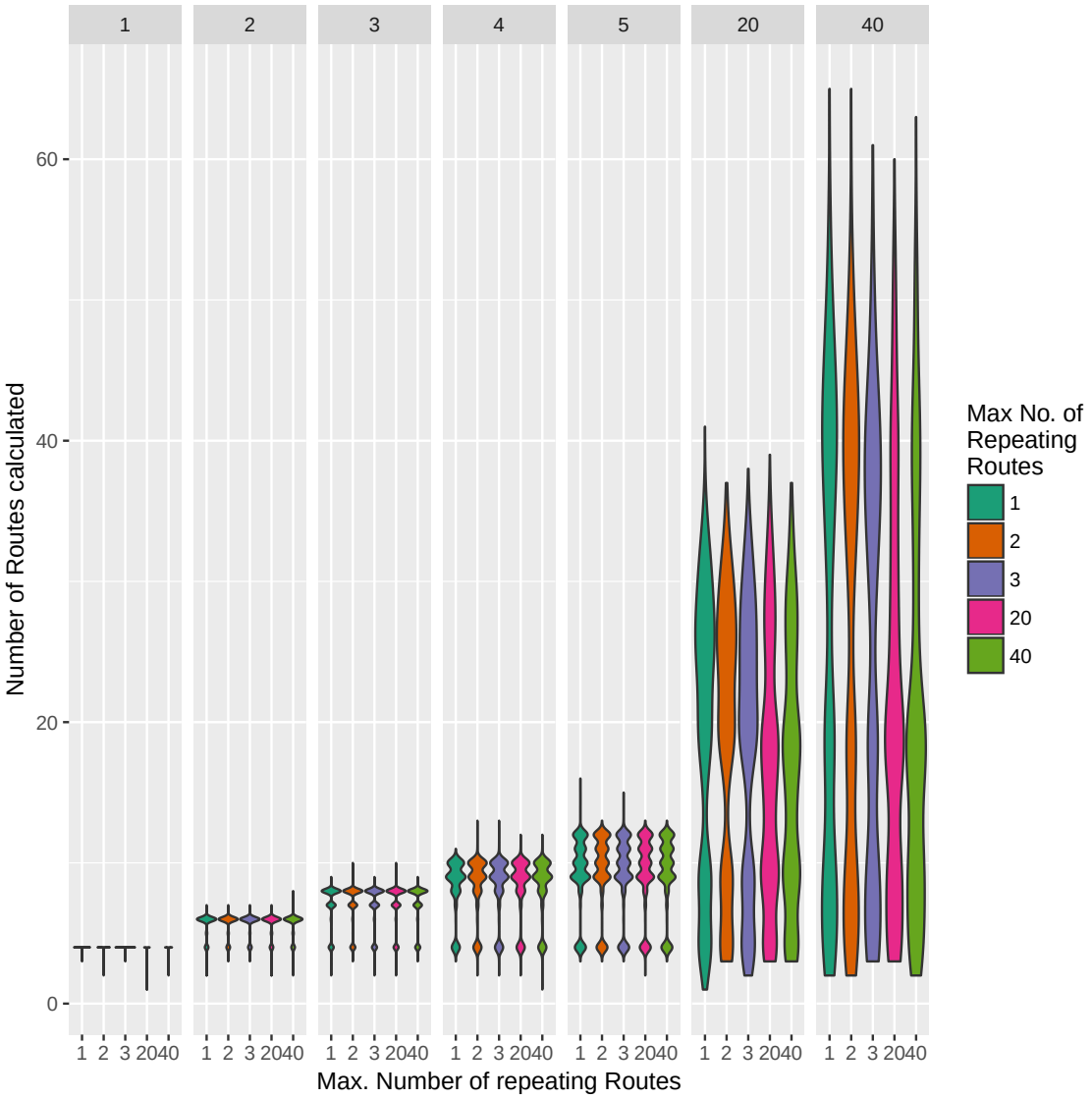


Figure 4.6: Violin plot of totally calculated routes per different amounts of maximum splits (indicated at the top of the diagrams) and the maximum number of repeated routes in a refinement tree (color coded)

5 Conclusion

In this thesis, a heuristic algorithm for finding alternative routes is presented. Personalized route planning with contraction hierarchies is used as the foundation for fast queries. With this basis, the Splitting Algorithm is designed to search the configuration space heuristically to find good alternative routes. Every route found this way is optimal considering a trade-off between the used metrics and is therefore also Pareto optimal. The Splitting Algorithm is experimentally proven to find more results than a randomized search on the configuration space. Furthermore, the parameters defined for the Splitting Algorithm are empirically investigated and evaluated.

5.1 Future Work

In Section 3.3 it was mentioned, that the Splitting Algorithm can also be applied to higher dimensions. It would be interesting to see if the algorithm performs differently with more metrics. The configuration space that can be explored increases when more metrics are added, but at the same time, more metrics might provide more alternative routes between any two nodes.

So far, the approach used relies on heuristics to find areas in the configuration space that could yield good alternative route. Another possible method would be to try to enumerate all routes for which an α exist. As the union of the upward graph from s and the downward graph from t contains all those paths, it might be a good starting point for designing an algorithm that looks for routes corresponding to an α.

Bibliography

[18a] *Bicycle - OpenStreetMap Wiki*. Jan. 2018. URL: https://wiki.openstreetmap.org/wiki/Bicycle (visited on 03/29/2018) (cit. on p. 27).

[18b] *OpenStreetMap*. en-US. Mar. 2018. URL: https://www.openstreetmap.org/ (visited on 03/29/2018) (cit. on p. 27).

[ADGW13] I. Abraham, D. Delling, A. V. Goldberg, R. F. Werneck. "Alternative routes in road networks". en. In: *Journal of Experimental Algorithmics* 18 (Dec. 2013), pp. 1.1–1.17. ISSN: 10846654. DOI: 10.1145/2444016.2444019. URL: http://dl.acm.org/citation.cfm?doid=2444016.2444019 (visited on 01/26/2018) (cit. on p. 14).

[Bar18a] F. Barth. *Mutli-Criteria Bicycle Routing*. original-date: 2017-10-10T14:56:06Z. Apr. 2018. URL: https://github.com/Lesstat/cycle-routing (visited on 04/07/2018) (cit. on p. 27).

[Bar18b] F. Barth. *Pbfextractor*. original-date: 2017-10-10T14:56:06Z. Apr. 2018. URL: https://github.com/Lesstat/pbfextractor (visited on 04/07/2018) (cit. on p. 28).

[BDGS11] R. Bader, J. Dees, R. Geisberger, P. Sanders. "Alternative Route Graphs in Road Networks". In: *Theory and Practice of Algorithms in (Computer) Systems*. Ed. by A. Marchetti-Spaccamela, M. Segal. Vol. 6595. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 21–32. ISBN: 978-3-642-19753-6. DOI: 10.1007/978-3-642-19754-3_5. URL: http://link.springer.com/10.1007/978-3-642-19754-3_5 (visited on 01/17/2018) (cit. on p. 14).

[Bri14] K. Bringmann. "Why Walking the Dog Takes Time: Frechet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails". In: *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. Oct. 2014, pp. 661–670. DOI: 10.1109/FOCS.2014.76 (cit. on p. 21).

[Daw15] B. Dawes. *Filesystem*. Jan. 2015. URL: https://www.boost.org/doc/libs/1_58_0/libs/filesystem/doc/index.htm (visited on 04/07/2018) (cit. on p. 28).

[Dij59] E. W. Dijkstra. "A note on two problems in connexion with graphs". en. In: *Numerische Mathematik* 1.1 (Dec. 1959), pp. 269–271. ISSN: 0029-599X, 0945-3245. DOI: 10.1007/BF01386390. URL: https://link.springer.com/article/10.1007/BF01386390 (visited on 03/26/2018) (cit. on pp. 13, 17).

[DW09] D. Delling, D. Wagner. "Pareto Paths with SHARC". In: *Experimental Algorithms*. Ed. by J. Vahrenhold. Vol. 5526. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 125–136. ISBN: 978-3-642-02010-0. DOI: 10.1007/978-3-642-02011-7_13. URL: http://link.springer.com/10.1007/978-3-642-02011-7_13 (visited on 03/30/2018) (cit. on p. 14).

[FLS17] S. Funke, S. Laue, S. Storandt. *Personal Routes with High-Dimensional Costs and Dynamic Approximation Guarantees*. en. Tech. rep. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 2017. DOI: 10.4230/LIPIcs.SEA.2017.18 (cit. on pp. 14, 17, 20).

[FRC+07] Farr Tom G., Rosen Paul A., Caro Edward, Crippen Robert, Duren Riley, Hensley Scott, Kobrick Michael, Paller Mimi, Rodriguez Ernesto, Roth Ladislav, Seal David, Shaffer Scott, Shimada Joanne, Umland Jeffrey, Werner Marian, Oskin Michael, Burbank Douglas, Alsdorf Douglas. "The Shuttle Radar Topography Mission". In: *Reviews of Geophysics* 45.2 (May 2007). ISSN: 8755-1209. DOI: 10.1029/2005RG000183. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2005RG000183 (visited on 03/29/2018) (cit. on p. 27).

[Gei08] R. Geisberger. *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks*. July 2008. (Visited on 03/19/2018) (cit. on pp. 18, 19).

[GSSD08] R. Geisberger, P. Sanders, D. Schultes, D. Delling. "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks". en. In: *Experimental Algorithms*. Ed. by C. C. McGeoch. Vol. 5038. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 319–333. ISBN: 978-3-540-68552-4. DOI: 10.1007/978-3-540-68552-4_24. URL: http://link.springer.com/10.1007/978-3-540-68552-4_24 (visited on 03/19/2018) (cit. on p. 18).

[HHSF17] J. Hall, L. Hafer, M. Saltzmann, J. Forrest. *COIN-OR Linear Programming Solver*. Jan. 2017. URL: https://projects.coin-or.org/Clp (visited on 04/07/2018) (cit. on p. 28).

[Mar84] E. Q. V. Martins. "On a multicriteria shortest path problem". In: *European Journal of Operational Research* 16.2 (May 1984), pp. 236–245. ISSN: 0377-2217. DOI: 10.1016/0377-2217(84)90077-8. URL: http://www.sciencedirect.com/science/article/pii/0377221784900778 (visited on 03/26/2018) (cit. on p. 17).

[MZ00] K. Mehlhorn, M. Ziegelmann. "Resource Constrained Shortest Paths". In: *Algorithms - ESA 2000*. Ed. by G. Goos, J. Hartmanis, J. van Leeuwen, M. S. Paterson. Vol. 1879. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 326–337. ISBN: 978-3-540-41004-1. DOI: 10.1007/3-540-45253-2_30. URL: http://link.springer.com/10.1007/3-540-45253-2_30 (visited on 03/31/2018) (cit. on p. 14).

[Pru] V. Prus. *Chapter 21. Boost.Program_options - 1.58.0*. URL: https://www.boost.org/doc/libs/1_58_0/doc/html/program_options.html (visited on 04/07/2018) (cit. on p. 28).

[Ram04] R. Ramsey. *Serialization*. Nov. 2004. URL: https://www.boost.org/doc/libs/1_58_0/libs/serialization/doc/index.html (visited on 04/07/2018) (cit. on p. 28).

[Rob57] C. C. Robusto. "The Cosine-Haversine Formula". In: *The American Mathematical Monthly* 64.1 (1957), pp. 38–40. ISSN: 0002-9890. DOI: 10.2307/2309088. URL: http://www.jstor.org/stable/2309088 (visited on 03/29/2018) (cit. on p. 27).

[Sto12] S. Storandt. "Route Planning for Bicycles-Exact Constrained Shortest Paths Made Practical via Contraction Hierarchy." In: *ICAPS*. Vol. 4. 2012, p. 46 (cit. on p. 14).

All links were last followed on March 17, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature