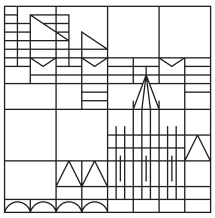
Improved Algorithms for Rendezvous Routing based on the Penalty Approach

Master's Thesis

presented by
Daniel Muffler

at the

Universität Konstanz



Faculty of Sciences
Department of Computer and Information Science

1. Evaluated by	Prof. Dr. Sabine Storandt
2. Evaluated by	Prof. Dr. Michael Grossniklaus

Konstanz, 2021

Daniel Muffler

Improved Algorithms for Rendezvous Routing based on the Penalty Approach

Abstract

With the rise of digitization and traffic, efficient route planning gains in importance. Additionally, vehicles like drones are used for transportation. This requires more enhanced and efficient routing algorithms. Apart from computing shortest paths, the problem of alternative route planning plays an important role for route planning. For the alternative route planning, other paths than the shortest paths should be found and alternative routes should meet further criteria. The Penalty algorithm as an algorithm for alternative route planning makes use of penalizing edges and aims to retrieve high-quality alternative routes.

This work first defines a special case of alternative route planning, namely Rendezvous Routing. As this work initiates the research for Rendezvous Routing, a time complexity analysis is conducted and results in a weakly NP-hardness. Then, the Penalty approach is evaluated whether it suits the requirements for Rendezvous Routing. Improved variants of the Penalty approach for Rendezvous Routing are developed including a parameter discussion concerning quality of results and running time.

The evaluation and the suitability study indicates that the Penalty approach can be adapted for Rendezvous Routing and the developed algorithm(s) perform well concerning running time and quality of an approximation for Rendezvous Routing.

Zusammenfassung

Mit zunehmender Digitalisierung und steigendem Verkehrsaufkommen gewinnt eine effiziente Routenplanung immer mehr an Bedeutung. Zusätzlich werden Fahrzeuge und Luftfahrzeuge wie Drohnen vermehrt für den Transport eingesetzt. Dies erfordert verbesserte und effizientere Routing-Algorithmen. Neben der Berechnung kürzester Wege spielt bei der Routenplanung auch das Problem der Alternativroutenplanung eine wichtige Rolle. Für die alternative Routenplanung sollen andere Wege als die kürzesten Wege gefunden werden und weitere Kriterien für Alternativrouten erfüllt werden. Der Penalty-Algorithmus als Algorithmus zur Alternativroutenplanung nutzt die Bestrafung von Kanten (Gewichtserhöhung) und zielt darauf ab, qualitativ hochwertige Alternativrouten zu finden.
Diese Arbeit definiert zunächst einen Spezialfall der alternativen Routenplanung, nämlich das Rendezvous-Routing. Da diese Arbeit den Grundstein für die Forschung für das Rendezvous-Routing setzt, wird eine Zeitkomplexitätsanalyse durchgeführt, die zu einer schwachen NP-Härte des Rendezvous-Routings führt. Anschließend wird der Penalty-Ansatz daraufhin untersucht, ob er den Anforderungen des Rendezvous-Routing entspricht. Verbesserte Varianten des Penalty-Ansatzes für das Rendezvous-Routing werden entwickelt, einschließlich einer Parameterdiskussion bezüglich der Qualität der Ergebnisse und der Laufzeit.
Die Evaluierung und die Eignungsstudie zeigen, dass der Penalty-Ansatz für das Rendezvous-Routing angepasst werden kann und die entwickelten Algorithmen hinsichtlich der Laufzeit und der Qualität der Approximation für das Rendezvous-Routing gut abschneiden.

Acknowledgements

First, I would like to thank Professor Sabine Storandt for her great accompaniment of this work and very helpful feedback. Next, my thanks go to Professor Michael Grossniklaus for his examination of this work.
Finally, I would like to thank my family and friends for their general support and feedback on the thesis.

Table of Contents

CHAPTER 1
Introduction

In a highly digitized world as we have today, digital route planning gains more and more importance. As an example, Google Maps, as one of the most famous route planners, has more than one billion users per month[1]. In contrast to the well-known problem of computing a shortest path between a start and a target, the finding of alternative routes does not necessarily aim to compute the k-shortest paths but also tries to optimize other route parameters like overlapping of routes. There exist various approximate and optimized algorithms in alternative route planning, computing the k-shortest paths in a graph, which can be optimized for additional parameters and adjusting edge weights dynamically [21, 6, 10, 5]. The alternative routes can be restricted to meet some criteria, like maximum overlap with the shortest path, maximum length of the alternative routes, and more.
In order to make the computation of alternative routes more efficient and more dynamic, *alternative graphs (AGs)* are used as an intermediate step. An AG results from multiple alternative routes represented in an alternative graph. From the alternative graph, a subset of routes can be selected as a solution to the alternative route planning problem [21].
However, this work's focus lies on a special case of alternative route planning called *Rendezvous Routing (RR)*. The RR problem is related to the problem of alternative route planning: For RR, the solution only consists of two routes whereas, in general, solutions for alternative route planning problems can contain multiple alternative routes.
RR defines further constraints on alternative routes. Thus, various applications in traffic routing and flight planning exist. An intuitive use case for RR is the transport of money with two cars: in the first car, the money is transported. The second car transports the key for the safe in the first car. Now, an optimal transport route looks like this: the routes of the two cars do not overlap and both cars still arrive at the destination at the same time with the same departure time without driving long detours compared to the shortest route. The same situation holds for transports of critical goods or emergencies.
Taking an alternative graph representation, the idea is to select a pair of candidate routes as a solution for RR. Therefore, possibly any existing algorithm from alternative route planning could be used and adapted for RR. Nevertheless, the *Penalty approach* is a highly customizable algorithm for alternative route planning, as it requires various input parameters that define constraints on the alternative

[1] https://cloud.google.com/blog/products/maps-platform/9-things-know-about-googles-maps-data-beyond-map

routes. The Penalty algorithm is a heuristic algorithm and aims to find alternative routes meeting predefined criteria that are reflected in the input parameters of the algorithm. This work provides an algorithmic evaluation of the Penalty approach proposed by Kobitzsch et al. applied on RR, which will be introduced later [18]. In addition, the time complexity of RR and the suitability of the Penalty algorithm for RR are examined separately.

1.1 Problem Setting

A graph $G = (V, E)$ in the context of this work is a directed and weighted graph. The weights of the edges are defined as $\omega : E \to \mathbb{R}$ and are *non-negative*. A path $\mathcal{P}_{s,t}$ from a source node s to a target node t is a sequence of vertices defined as follows: $\mathcal{P}_{s,t} = \langle v_0, v1, \ldots, v_l \rangle$, where $v_i \in V$ and $(v_i, v_{i+1}) \in E$. The length $\mathcal{L}(\mathcal{P}_{s,t})$ is the combined weights of the edges of the path, so $\mathcal{L}(\mathcal{P'}_{s,t}) = \sum_{j=0}^{l-1} \omega(v_j, v_{j+1})$. The shortest path between s and t is the path where the length is minimal over all possible paths. The distance between s and t is defined as the length of the shortest path [3, 18].

Taking the RR problem, more constraints on alternative routes are added. In the RR, the goal is to find two paths from source to target which do not overlap and are roughly of equal length. More formally, two paths $\mathcal{P}_{s,t}$ and $\mathcal{P'}_{s,t}$ which share the same source s and the same target t have to meet the following optimization problem for any path $\mathcal{P}$ and any path $\mathcal{P}'$ which should solve the RR problem for a given source s and target t:

$$\min(|\mathcal{L}(\mathcal{P}_{s,t}) - \mathcal{L}(\mathcal{P'}_{s,t})|) \tag{1}$$

Furthermore, sharing is not allowed (except for source node and target node):

$$\mathcal{L}(\mathcal{P}_{s,t} \cap \mathcal{P'}_{s,t}) = 2 \tag{2}$$

Lastly, the two paths which solve the RR problem have to be of minimum length concerning other pairs of paths. For any pair $\mathcal{L}(\mathcal{P''}_{s,t})$ and $\mathcal{L}(\mathcal{P'''}_{s,t})$ which meet Condition (1) and Condition (2), another optimization has to be considered:

$$\mathcal{L}(\mathcal{P}_{s,t}) + \mathcal{L}(\mathcal{P'}_{s,t}) \leq \mathcal{L}(\mathcal{P''}_{s,t}) + \mathcal{L}(\mathcal{P'''}_{s,t}) \tag{3}$$

In addition, the complexity of algorithmic problems for route planning may change whether or not repeating nodes in paths are allowed. For the RR problem, only loopless paths are allowed for any path $\mathcal{P}_{s,t}$:

$$|\{v|v \in \mathcal{P}_{s,t}\}| = \mathcal{L}(\mathcal{P}_{s,t}) \tag{4}$$

Basically, the definition of the RR problem contains two optimizations, Condition (1) and Condition (3) while also fulfilling Condition (2) and Condition (4). Together, Condition (1), Condition (2), Condition (3), and Condition (4) define the RR problem.

1.2 Research Objectives

As the RR problem is a fairly new problem, this work is dedicated to cover basic theoretical and practical research. Furthermore, the Penalty approach is one of the central algorithms for this work. The idea of the Penalty algorithm is to penalize edges laying e.g. on the shortest path. With the higher edge weights, a further run of the Dijkstra algorithm on the penalized graph leads to possible alternative routes. Additionally, constraints for the alternative routes are established and possible alternative routes are checked against these constraints. This work evaluates if the Penalty approach fits the requirements for the RR problem and and how well it performs concerning the RR problem. This substantiates the following questions:

1. *What is the theoretical complexity of the RR problem?*
2. *Does the Penalty approach suit the requirements for the RR problem?*
3. *How well does the Penalty algorithm perform in approaching the RR problem?*

Answering these questions results in a scientific contribution for basic research for the RR problem and an evaluation of the Penalty approach including adjustments of the algorithm. Adjusting the Penalty approach leads to various algorithmic variants. Furthermore, the analysis and evaluation of the Penalty approach emphasizes strength and weaknesses of the algorithm in general and demonstrates obstacles for solving or approximating the RR problem. Additionally, further algorithmic ideas for RR are presented.

Thus, the following research tasks can be derived:

Task 1: *Complexity analysis of the RR problem*

Task 2: *Suitability study of the Penalty approach for RR*

Task 3: *Implementation and experiments for evaluating the Penalty approach*

Task 4: *Visualization and evaluation of the results from the experiments*

Task 5: *Deriving further research fields for RR*

1.3 Outline

This work is divided into three different parts: analysis of RR, theoretic discussion of the Penalty algorithm including a (practical) suitability study, and the experimental evaluation of the Penalty algorithm applied on RR.
Chapter 2 summarizes related work, especially concerning the alternative route planning problem. Next, Chapter 3 explains the methodology used for the evaluation and discussion of the Penalty approach. In Chapter 4, the time complexity of RR is analyzed and proven. This is followed by a suitability study in Chapter 5 and Chapter 6. In the suitability study, practical experiments are conducted in Chapter 6 with a first implemented version of the Penalty approach. The goal of these first experiments is to evaluate the practical suitability of the Penalty algorithm for RR. Afterwards, improved variants of the Penalty algorithm are introduced in Chapter 7 and experimentally evaluated in Chapter 8. This experimental evaluation and the implemented algorithms are based on the experiments of the suitability study. Finally, Chapter 9 concludes this work and gives an overview over future work.

CHAPTER 2

Related Work on Alternative Route Planning

The following contains short summaries of references concerning the alternative route planning problem in general and demonstrates the aspect-richness of the research field.

Barth et al. [4] try to tackle the alternative route planning problem including multi-dimensional weight functions for edges. Especially the obstacle of self-driving cars and route selection dominates their work. The main idea of the approach is to enumerate possible routes from a source to a target by considering the multi-dimensional costs/weights. In addition, the individual cost components can be adapted concerning their importance by adding a request-individual importance function. In order to get more diverse alternative routes, the authors suggest to introduce additional metrics (apart from travel time etc.) such that the alternative routes provide sufficient diversity. To achieve the diversity constraint and sufficient performance, the authors make use of a convex hull computation. Afterwards, the path selection can be computed more efficiently. The experimental results of the authors show that processing of a whole country is possible within a few seconds and preserving quality of the alternative routes at the same time [4].

Stroobant et al. [23] explore another problem in the world of alternative route planning. They constraint the routes not only to a maximum length (upper bound) but also to a lower bound. The problem is shown to be strongly NP-hard if the aim is to find the shortest route considering the lower bound. Two algorithms are provided that solve the problem correctly. As the running time is not applicable for longer routes, a heuristic approach is developed by the authors. The heuristic makes extensive use of penalization of edges by taking an intermediary point to calculate a bicycle tour. In summary, the experimental results show that the quality of the tours seems to be good and the running times are reasonable for small real-world applications [23].

The work of Bader et al. [3] is motivated by the still existing obstacle of finding alternative routes that match the idea of human beings. Similar to the penalty approach, an alternative graph is used for better route extractions. The alternative graph can be computed by an approach like Penalty, Plateau, k-shortest path algorithm, or a combination of them. In order to extract the routes of the alternative graph, heuristics can be used. The authors suggest two new heuristics, Global and Local Thinout. The idea is to remove useless edges globally (the whole path) or

locally (for an edge). Furthermore, attributes of such an alternative graphs are defined (totalDistance, averageDistance, and decisionEdges). These attributes make it possible to remove useless edges with the Global and Local Thinout. Thus, the result is an optimized alternative graph where routes can be extracted. The experimental results and a user study show that the Penalty approach and the Plateau approach perform well for alternative routes and match the ideas of human beings better than other approaches [3].

Feng et al. [11] extend the Plateau method for alternative route planning with the concept of pheromones in traffic. First, the Plateau approach grows two trees from a source to a target: a forward and a backward tree. Taking the overlap of both leads to a list of nodes of a Plateau. Every node of the Plateau can now be included in a path from the source to the target. Various methods and improvements exist for selecting good Plateau nodes. Introducing the concept of pheromone in traffic as the authors in [2] did, could lead to better alternative routes for road maps. Furthermore, pheromone in the context of traffic can be seen as the density of traffic. The pheromone metric influences the computation of an alternative graph and the alternative routes such that the plateaus list is sorted in increasing order concerning the amount of pheromone. Additionally, the nodes in the plateau should lay on the main road network. The main road network is defined as the network including the main traffic routes. The experimental results indicate that the proposed algorithm and the heuristics seem to be better concerning quality of results compared to other approaches [11].

CHAPTER 3
Methodology

This section introduces the Penalty approach of Kobitzsch et al. and further work on alternative route planning namely a promising alternative algorithm proposed by Chondrogiannis et al. [6]. As there exist many work on alternative route planning, this section focuses on relevant work for the evaluation of the Penalty approach. Afterwards, the methodology used for this work is explained.

3.1 Penalty Approach

The following summarizes the proposed algorithm of Kobitzsch et al. [18]. The Penalty algorithm will be evaluated in the upcoming sections.
The work of Kobitzsch et al. proposes an approach for alternative route planning on (directed) graphs, based on the work of Bader et al. [3]. As various methods for alternative route planning exist, the main contributions of the approach of Kobitzsch et al. are the almost real-time processing of an entire graph using a dynamic level selection technique, and the improvements of the original penalty algorithm.
First, the idea of the penalty approach of Bader et al. is to penalize arcs of a shortest path as long as alternative routes can be found. The iterated computation of the shortest path between a source and a target is performed on the alternative graph which results from the penalization of arcs. While finding alternative routes, each possible alternative route is checked against three criteria: limited sharing, uniformly bounded stretch, and local optimality[1].
Furthermore, small hops in the alternative route compared to the shortest path are avoided by penalizing not only edges on the current found path but also edges connected to nodes of the current found path. Each penalization of different types of arcs is parametrized and can thus be adapted to get better results. Another improvement introduced by Kobitzsch et al. is the overall converging criterion. It is defined by the length of the current found path and the length of the shortest path using an additional input parameter. Further changes to the penalization are motivated by fulfilling the local optimality criterion and optimizing running time. The authors propose values for the input parameters mainly based on experimental

[1] Limited sharing implies a restricted overlap of the possible alternative route and the shortest path. The criterion of a uniformly bounded stretch defines the maximal length between pairs of nodes of the alternative path and thus restricts the length of alternative paths. Sub-paths of predefined maximum lengths of the alternative paths have to be optimal, which is reflected in the local optimality criterion.

results.
One of the most important contributions of the paper, besides the improvements for the algorithm itself, is to apply a dynamic level selection technique. Therefore, the original graph is hierarchically decomposed to get a multi-level graph. Finding the shortest path on the graph (including penalties) can be performed using a dynamically selected search level. Additionally, updates of arcs in the penalization steps are performed on different levels leading to a balancing problem between update costs and runtime costs for the shortest path queries. The authors suggest experimentally evaluated hierarchy levels depending on the length of the shortest path.
In summary, the penalty approach as introduced by Kobitzsch et al. in combination with the dynamic level selection technique seems to be sufficient in quality and performance for real-time dynamic route finding applications (based on the experimental results of the authors).

3.2 k-Shortest Paths with Limited Overlap

Chondrogiannis et al. introduce the *k-Shortest Paths with Limited Overlap (kSPwLO)* problem as a closely related definition to RR [6]. The outcomes of the paper are summarized and explained in a more detailed way. First, the authors define a similarity measure for alternative paths by calculating a similarity ratio taking the overlap of two paths divided by the length of the shorter path into account. The parameter k refers to the number of alternative paths that have to be found by the algorithm. Additionally, the proof for the kSPwLO problem to be weakly NP-hard is provided. This proof is also used in this work to show that the RR problem can be reduced to the kSPwLO problem (see Chapter 4). In their work, Chondrogiannis et al. provide various algorithms for exactly solving kSPwLO and heuristic algorithms. For the heuristic algorithms, the definition and criteria of kSPwLO are adapted such as alternative paths do not have to be as short as possible. Consequently, the heuristic approaches do not give any guarantee concerning the lengths of the alternative paths.
Even though the experiments are conducted using advanced implementations and hardware, the running times of the heuristic approaches highly depend on the input parameter k. Furthermore, the higher the dissimilarity, the higher the running time. Nevertheless, the authors provide well-performing heuristic algorithms for the kSPwLO problem. Figure 1 gives an overview over datasets used by the authors to measure the performance of the algorithms.
In order to extract some indications for the later evaluation in this work, Figure 2 presents the running times of two heuristic approach for different datasets. It is quite intuitive that for increasing parameter k the running times get also higher as the finding of alternative paths is an iterative process.

Road networks	Nodes	Edges	Structure
Rome	3353	8870	City-center
Oldenburg	6105	14,058	City-center
San Joaquin	18,263	47,594	Grid-based
Tianjin	31,002	86,584	Ring-based
Porto Alegre	63,751	187,364	Grid-based
Beijing	74,383	222,778	Ring-based
Milan	187,537	525,296	Ring-based
Chicago	386,533	1,121,620	Grid-based
Colorado	435,666	1,057,066	State
Florida	1,070,376	2,712,798	State

Figure 1 Datasets used by Chondrogiannis et al. to estimate the performance of their algorithms. Table taken from [6].

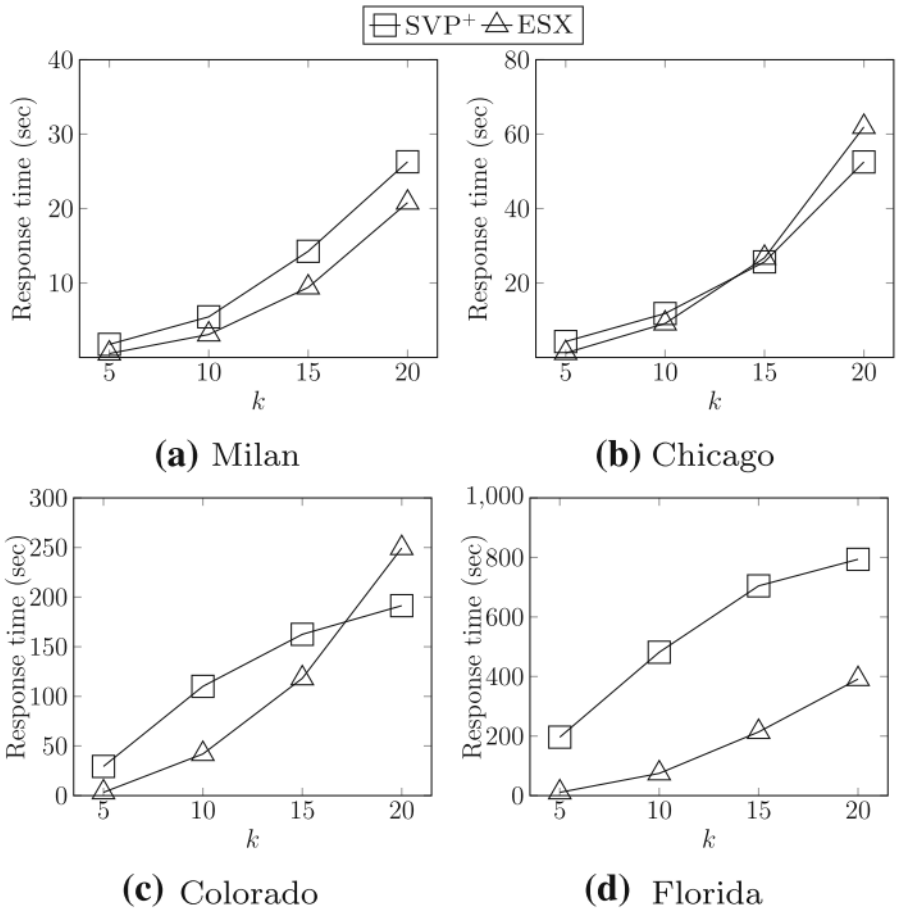


Figure 2 Running times of two heuristic approaches in dependence of parameter k. Figure taken from [6].

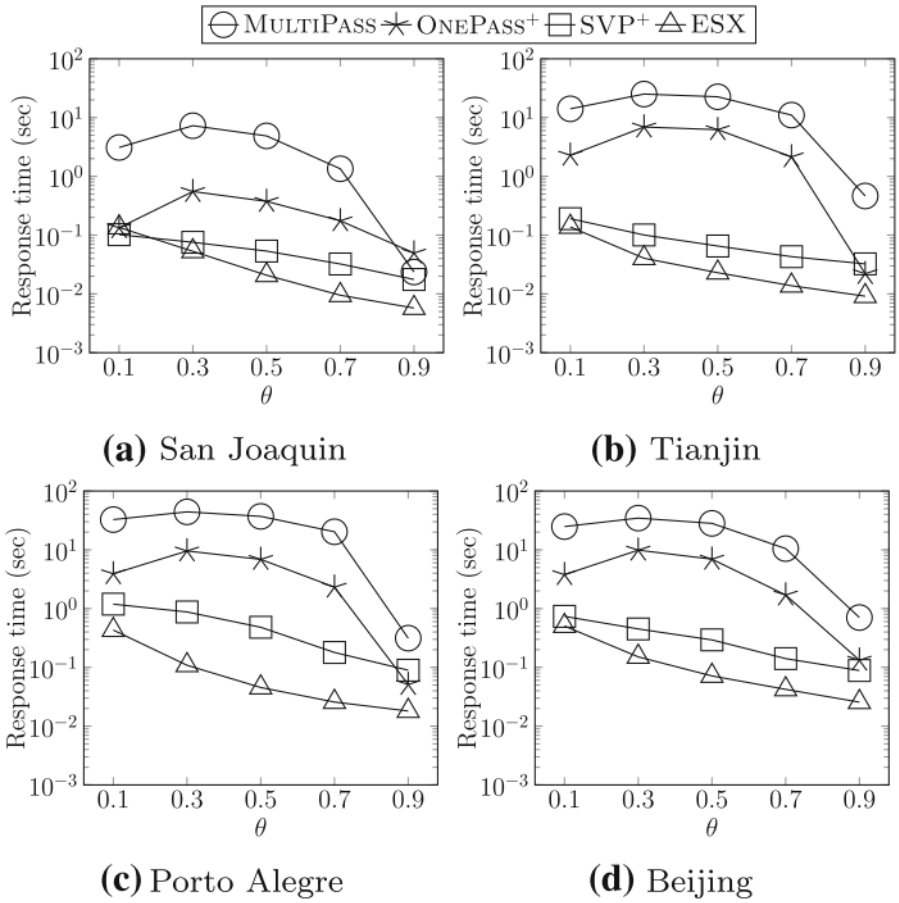


Figure 3 Running times of two heuristic approaches in dependence of parameter θ. Figure taken from [6].

In addition, Figure 3 shows the running times in relation to changes of the similarity θ of alternative paths. In general, it can be seen that the more similar the alternative routes can be the lower the running times. The observations from Chondrogiannis et al. are used in the evaluation section to provide a comparison of different approaches.

3.3 Procedure

In order to extensively evaluate the RR problem and possible approximations, the methodological approach in this work includes the following steps:

1. Proof of weak NP-completeness of Rendezvous Routing
2. Theoretical discussion of Penalty approach including running time and quality of alternative routes
3. Practical evaluation and discussion of the Penalty approach including running time and quality of alternative routes

4. Improved Penalty algorithm for Rendezvous Routing: Optimized version of Penalty algorithm for running time and quality evaluation

Step 2 and 3 can be summarized in a suitability study in order to evaluate whether the Penalty approach meets the requirements for RR. The last step takes the results from the previous steps into account, followed by an extensive evaluation of the implemented improved algorithm. Furthermore, speed-up techniques like a bidirectional Dijkstra variant and Contraction Hierarchies are applied.
The evaluation of the Penalty approach includes measuring the route quality. Therefore, a pair of routes is compared by their dissimilarity, the length difference of the routes and the overlap length. In addition, the length of the larger path in comparison to the shortest path between a source and a target completes the quality measurement of a pair of routes.
In order to gain better insights about the running time, the running time of sub parts of the algorithm are measured separately. This enables an enhanced analysis of possible points for improvements.
The algorithms are implemented and evaluated in Java 11. In addition, roughly 50 Gigabyte of RAM are given to the application for the experiments operating on an AMD Ryzen 7 3700X processor. The graph for the metrics is a real-world road map consisting of about 100 000 nodes.

CHAPTER 4
Time Complexity

Chondrogiannis et al. provide a proof for the weakly NP-hardness of kSPwLO (see Section 3.2). The following sketches the idea of the proof. Furthermore, the proof for RR to be weakly NP-hard is introduced.

Chondrogiannis et al. reduce the *subset sum problem* to kSPwLO. For the subset sum problem, index-based natural numbers are given: $a_1, \ldots, a_m \in \mathbb{N}, S \in \mathbb{N}$. The problem asks for an index set $I \subseteq \{1, \ldots, m\}$ such that $\sum_{i \in I} a_i = S$.

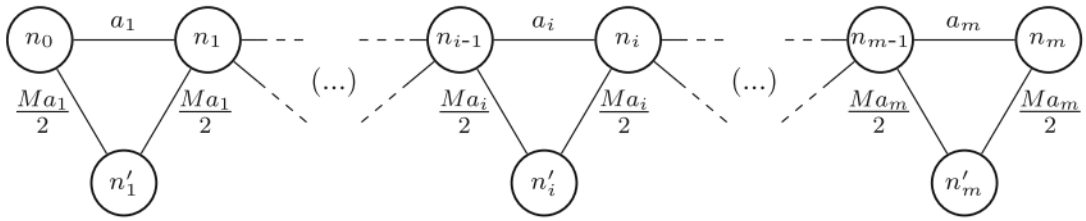


Figure 4 Fixed instance for subset sum problem represented in a graph. Figure taken from [6].

The idea of the proof is to fix an instance of the subset sum problem and transform it in a graph as shown in Figure 4. The kSPwLO aims to find two paths for that instance with a limited overlap of $\frac{S}{A}$. Furthermore, $M > A = \sum_{i=1}^{m} a_i$. So M must be greater than A, otherwise the overlap for the kSPwLO problem could not be satisfied. In order to conduct the reduction, there has to be an index set $I \subseteq \{1, \ldots, m\}$ with $\sum_{i \in I} a_i = S$ if and only if $\mathcal{L}(\mathcal{P}_2 = A \cdot M - S \cdot (M - 1)$ where P_2 is the second path found by kSPwLO. It can be clearly seen by Figure 4 that such a path has to exist as the shortest path and P_2 can be edge-disjoint ($\langle n_0, n'_1, \ldots, n'_m, n_m \rangle$).

In addition to the constraint that the paths of RR have to be completely dissimilar (Condition (2)), Condition (1), and Condition (3) add more constraints on the lengths of the paths for RR compared to kSPwLO. As P_2 can be chosen to be completely edge-disjoint, the dissimilarity criterion does not change the proof for the weakly NP-hardness of RR. For Condition (1), the condition is already included in the definition of kSPwLO as the alternative paths have to be of minimum length. Additionally, Condition (3) (minimum difference of pair of paths) does not change the proof above neither. In order to find the optimal solution for Condition (3), it is

necessary to enumerate all possible alternative paths and find the pair that minimizes the length difference of all pair of paths. Therefore, the weakly NP-hardness holds. As Condition (3) could be a conflicting criterion with Condition (1), it requires to e.g. define a prioritization and/or length limit here. Nevertheless, this optimization criterion can only be fulfilled if all paths are enumerated and all pairs of paths are compared to each other. Therefore, this criterion does not change the proven time complexity. Lastly, Condition (4) (no repeating nodes in paths) is indirectly included in the definition of kSPwLO as we assume that all edge weights are non-negative. This leads to acyclic shortest paths.
Thus, all (additional) criteria do not change the proof and therefore the time complexity. This shows that the RR problem is also weakly NP-hard. As exact algorithms are not efficient for the RR problem and bounded approximation algorithms are not known, heuristic algorithms may be designed for real-world use cases.

CHAPTER 5
Theoretical Suitability

In Chapter 4, we have shown that the time complexity for the RR problem does not allow efficient exact algorithms. Next, the suitability of the Penalty approach for RR is evaluated. The first part approaches the Penalty algorithm from a theoretical perspective while the second part conducts a practical suitability study. In addition, the methodology for the suitability study is briefly explained.

The following study examines the Penalty approach generally concerning possible use cases. In addition, places for improvements in the Penalty algorithm are identified for later use.

5.1 Methodology

As a first step, the theoretical discussion of the Penalty algorithm is based on a running time discussion and a quality discussion. As the Penalty approach is a heuristic to a good solution, bad running times as well as bad quality of the routes are possible. The quality of alternative routes can be measured by how good a route performs according to the input parameters and other alternative routes. Furthermore, the comparison to the shortest route can also be a measurement of quality of routes.

Secondly, the elaboration of use cases is based on the theoretical discussion in the first part. Use cases are extracted from related work and evaluated for the Penalty algorithm. In particular, "good" and "bad" use cases are extracted and provided that show the strengths and weaknesses of the Penalty approach. Bad use cases are the ones where the running time of the Penalty algorithm is bad or the the quality of the routes is low. If a use case fits both criteria, running time and quality, it is considered to be a feasible application.

5.2 Algorithm Discussion

This section first gives more formal definitions for alternative route planning and the Penalty approach, focusing on the algorithm itself and the improvements suggested

by Kobitzsch et al. This is followed by a closer running time and quality discussion. Furthermore, examples demonstrate the running time and quality of the Penalty algorithm.

5.2.1 Background

In order to specify the criteria for evaluating routes in the Penalty approach, the definitions of limited sharing, uniformly bounded stretch and local optimality are introduced. There, $\mathcal{D}(a, b)$ is denoted as the length of the shortest path between a and b. Taking a shortest path $\mathcal{P}_{s,t}$ between s and t, the shortest path $\mathcal{P}_{s,v,t}$ where $v \in V \backslash \mathcal{P}_{s,t}$ is a so-called *Via-Node Alternative Route*. As such via-node alternatives can be arbitrary bad, the aforementioned definitions are introduced. If all these criteria are fulfilled for the parameters $\gamma, \epsilon, \alpha, v$, the alternative route is called *viable*. The formal definitions can be summarized as follows: [1, 18]

1. $\mathcal{L}(\mathcal{P}_{s,t} \cap \mathcal{P}_{s,v,t}) \leq \gamma \cdot \mathcal{D}(s, t)$
2. $\forall a, b \in \mathcal{P}_{s,v,t}, \mathcal{D}_{\mathcal{P}_{s,v,t}}(s, a) < \mathcal{D}_{\mathcal{P}_{s,v,t}}(s, b) :$
   $\mathcal{D}_{\mathcal{P}_{s,v,t}}(a, b) \leq (1 + \epsilon) \cdot \mathcal{D}(a, b)$
3. $\forall a, b \in \mathcal{P}_{s,v,t}, \mathcal{D}_{\mathcal{P}_{s,v,t}}(s, a) < \mathcal{D}_{\mathcal{P}_{s,v,t}}(s, b),$
   $\mathcal{D}_{\mathcal{P}_{s,v,t}}(a, b) \leq \alpha \cdot \mathcal{D}(s, t) : \mathcal{D}_{\mathcal{P}_{s,v,t}}(a, b) = \mathcal{D}(a, b)$

Kobitzsch et al. suppose a slightly improved and adapted version of the Penalty algorithm proposed by Bader et al. [18, 3]. The algorithm is summarized in Algorithm 1.

Algorithm 1 Penalty

Require: Input graph G, source s, target t, $\gamma, \epsilon, \alpha, \pi_f$
original_path = path = computeShortestPath(G, s, t)
H = {original_path}
$G' = G$
while $\mathcal{L}$(path) $\leq (1 + \epsilon) \cdot \mathcal{L}$(original_path) **do**
  applyPenalties(G', path, α, π_f)
  path = computeShortestPath(G', s, t)
  if isFeasible(path) **then**
    H = H $\cup$ path
  end if
end while
return H

It remains to clarify the functions mentioned in Algorithm 1.

- *computeShortestPath* is just an invocation of the (bidirectional) Dijkstra algorithm in order to simplify the discussion. Clearly, the Dijkstra algorithm can be replaced by any other shortest path algorithm.

- *applyPenalties* applies penalties to the current found shortest path as well as to (incoming) edges of nodes lying on the current found *path*. π_f indicates the factor by how much the edges on the current shortest paths are penalized. The new weight of an edge i laying on the current found shortest path is calculated by $\omega_{p_i} = \omega_{p_i} + \omega_{p_i} \cdot \pi_f$, where ω_{p_i} is the current weight of edge i in the penalized graph G'. $\alpha \cdot \sqrt{D(s,t)}$ is the factor added to the edge weights of edges of the current found shortest path. Penalties are not applied to edges incident to the source or target node as it would not make any difference to the algorithm (especially for the target node). Penalizing incoming edges, outgoing edges, or both of nodes of the current found path depends on the problem we want to tackle. Applying penalties only to incoming edges of nodes lying on the current found path is neither particularly specified in the paper of Kobitzsch et al. nor in the work of Bader et al. Nevertheless, penalizing incoming edges and thus avoid to go back to a previous found path again, makes sense in an intuitive way. Not penalizing outgoing edges would help to get away from a previously found path. Figure 5 illustrates the penalizing of only incoming edges after computing the shortest path on the original graph. As the incident edges of the source and target node are not penalized, the edge between node a and node c is the only penalized edge apart from the edges of the shortest path.

- *isFeasible* verifies if a given alternative route fulfills the first (limited sharing) and second (uniformly bounded stretch) condition. Thus, for checking the uniformly bounded stretch, pairwise distance calculations are necessary and highly influence the running time of the algorithm.

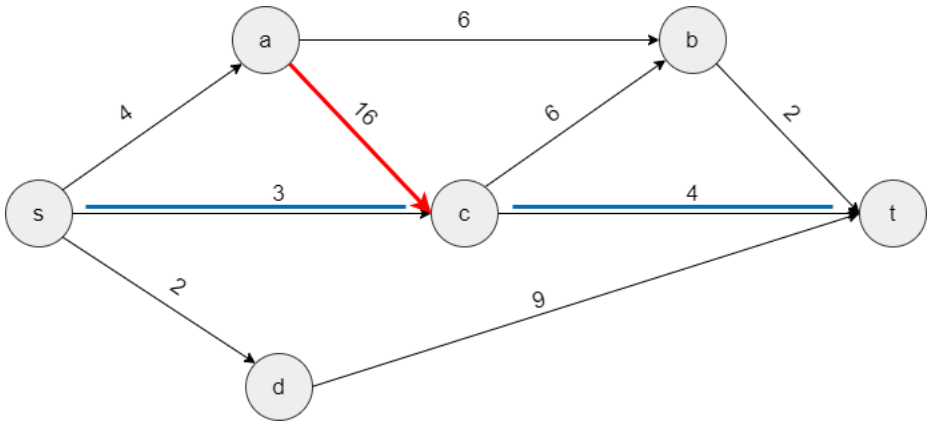


Figure 5 Example of penalizing incoming edges of nodes of the shortest path with $\epsilon > 0$. The shortest path from s to t is marked blue and the penalized edge in the invocation of the while-loop in algorithm Algorithm 1 is marked red.

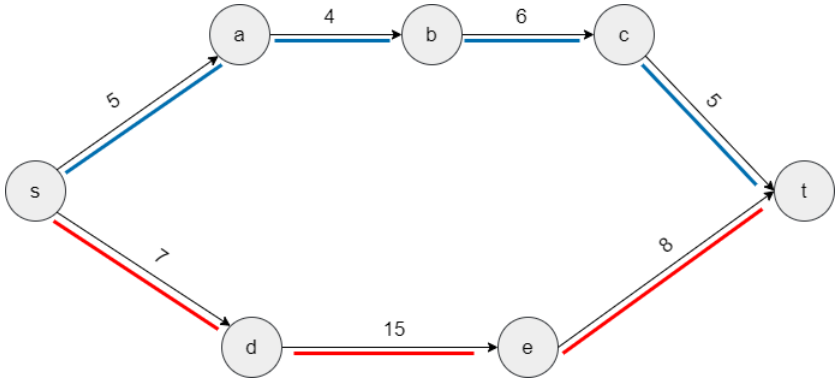


Figure 6 Penalty algorithm after finishing with the following parameters: $\epsilon = 0.5, \pi_f = 0.04, \alpha = 0.5, \gamma = 0.1$ from node s to t. The blue path indicates the shortest path on the original graph and the red one the alternative route.

Remark: Algorithm 1 could end up in an endless loop if there is e.g. only one path from source to target. Then the condition in the while loop will never be false. To avoid this, an additional check inside the while loop could solve the problem. In this check, the length of the current found path *in the penalized graph* G' is examined whether it is too long. If this path gets too long (e.g. the length of the path including penalties does not fulfill the condition of the while loop), one can end the loop.

5.2.2 Running Time Discussion

Analyzing the running time of the Penalty algorithm highly depends on the number of invocations of the shortest path search and number of rounds in the while loop. Figure 6 shows an example of an efficient run of the Penalty approach. Whether or not a run of the Penalty approach is efficient depends on the structure of the graph and the input parameters. Decreasing ϵ in the example in Figure 6 would lead to a missing alternative route. Increasing ϵ can lead to an arbitrarily bad running time. An option to avoid such overhead in running time could be to limit the paths we search for. This requires further information on the problem we try to solve and the graph itself. In addition, for this example ϵ is the parameter that mainly influences the quality and running time of the algorithm. $\pi_f = 0.04$ and $\alpha = 0.5$ remain the same throughout this discussion as suggested by Kobitzsch et al. as an outcome of their practical evaluations. As there is no path overlapping in Figure 6 apart from the source and target node (like in RR), the parameter γ is not relevant.
Taking the example in Figure 7, the running time of the Penalty algorithm is not good. Although there exists only one path from source s to target t, the algorithm requires at least $\mathcal{O}(n^2)$ invocations of the shortest path algorithm caused by the feasibility check of the alternative route $s \to a \to b \to c \to d \to t$ where n is the number of nodes in graph G. This route will then be added to the result set after performing the feasibility check. This results in at least a cubic running

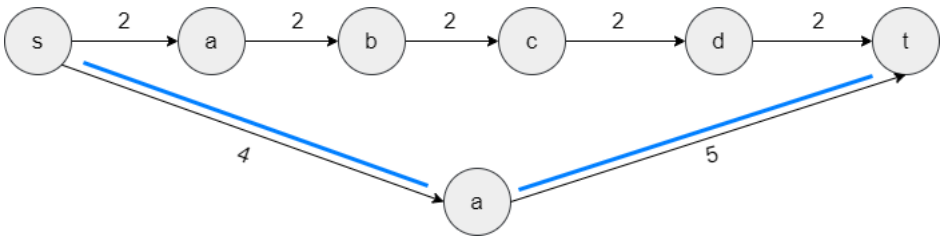


Figure 7 Penalty algorithm after the first round with the following parameters: $\epsilon = 0.2, \pi_f = 0.04, \alpha = 0.5, \gamma = 0.1$ from node s to t. The blue path indicates the shortest path on the original graph.

time. Choosing $\epsilon > 0$ would also result in unnecessary shortest path calculations computing the same shortest path again and again. The number of steps needed for the criterion of the while loop in Algorithm 1 to return false or to meet some other stopping criterion depends on the parameter π_f. As mentioned before, Kobitzsch et al. suggest a specific value ($\pi_f = 0.04$) which seems to work well for road networks. In addition, the rejoin penalty parameter is chosen as suggested: $\alpha = 0.5$.
Additionally, γ can highly influence the running time. If the overlap of the routes is bounded to small overlapping parts or no overlap like in RR, it could be more efficient than allowing high overlaps as the check for the overlap is more efficient than checking the uniformly bounded stretch. Nevertheless, sparse graphs might perform better for a sensible choice of the input parameters. If a graph is more dense, it could probably result in more feasibility checks and more different candidate routes. Summarizing the outcomes of the examples, the running time depends on the following factors: choice of parameters, structure of graph (sparse versus dense), number of paths to be found, and the number of nodes of the found paths. The number of nodes of a found path influences the running time because of the time the feasibility check needs. The choice of parameters should depend on the structure of the graph. Figure 7 illustrates this case.
For RR, γ can be fixed to $\gamma = 0$. But for some input parameters like number of paths to be found, α, and ϵ, various values can be set. Nevertheless, a trade-off between running time and quality of routes is essential.

5.2.3 Quality Discussion

Elaborating the quality of the routes calculated by the Penalty algorithm, Figure 8 is an example of good quality routes concerning the input parameters. For this example, ϵ is the threshold of how many routes can be found. In addition, γ can restrict the length of the part of routes they are allowed to share. Taking the input parameters ϵ and γ, the Penalty approach finds the optimal solution in Figure 8.

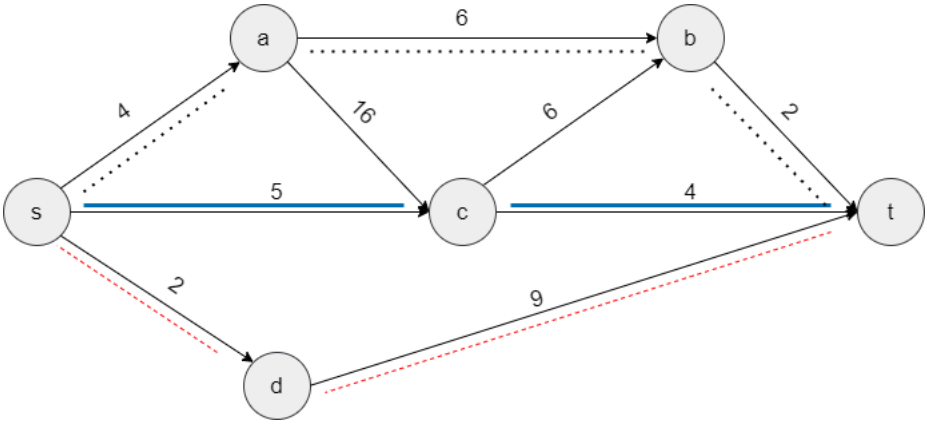


Figure 8 Penalty algorithm after finishing with the following parameters: $\epsilon = 0.5, \pi_f = 0.04, \alpha = 0.5, \gamma = 0.1$ from node s to t. The blue path indicates the shortest path on the original graph, the dashed and dotted paths the alternative routes.

Figure 9 is another example where the Penalty approach finds the optimal solution given the input parameters. In this case, it may not be clear why the Penalty approach succeeds in finding the dark dashed route. After finding the shortest path, edge $a \rightarrow c$ is punished more than one time. Afterwards, the red dashed route is found as the first alternative. Then, the dark dotted route is found. Edge $c \rightarrow b$ is now punished several times. Now the shortest path on G' including the penalties is exactly the dark dashed route. The path $s \rightarrow a \rightarrow c \rightarrow t$ is not found thanks to penalties of the edge $a \rightarrow c$.

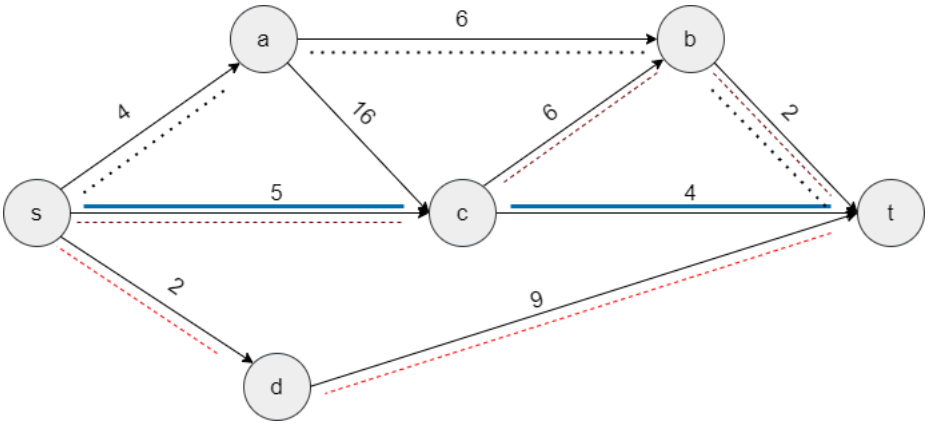


Figure 9 Penalty algorithm after finishing with the following parameters: $\epsilon = 0.5, \pi_f = 0.04, \alpha = 0.5, \gamma = 0.6$ from node s to t. The blue path indicates the shortest path on the original graph, the dashed and dotted paths the alternative routes.

In contrast to the previous examples, Figure 10 indicates an inconsistency concerning the parameters. $\gamma = 0.6$ indicates that more routes can be found while $\epsilon = 0.2$ restricts the overall length. Setting $\epsilon >= \frac{1}{3}$ would lead to more alternative routes fulfilling the restriction of γ. Nevertheless, the algorithm would not miss any routes here if the parameters were set differently.

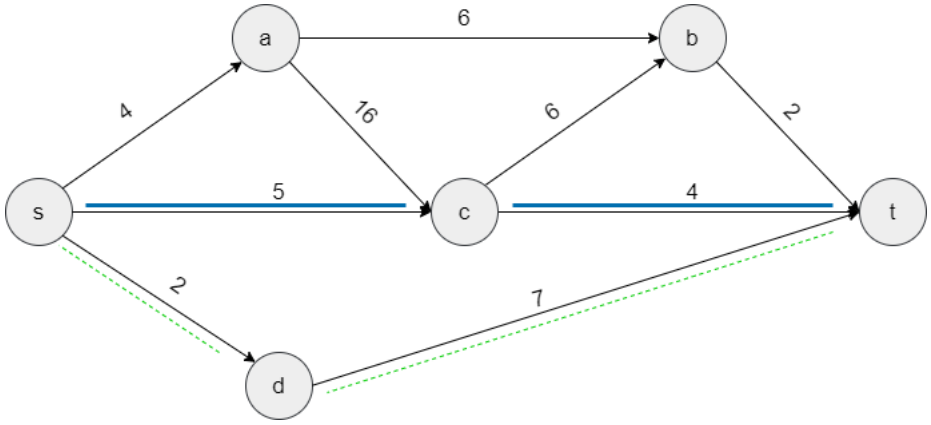


Figure 10 Penalty algorithm after finishing with the following parameters: $\epsilon = 0.2, \pi_f = 0.04, \alpha = 0.5, \gamma = 0.6$ from node s to t. The blue path indicates the shortest path on the original graph and the green dashed one the alternative route.

Summarizing the outcomes of the examples concerning the quality of routes, the Penalty approach works fine if the parameters are chosen accordingly. Missing routes depends on the input parameters, especially on the penalty factors. Choosing the penalty factors arbitrarily small, the Penalty approach will never miss a route. As it is a trade-off between quality and running time, an inconvenient parameter setting leads to a bad trade-off, especially for RR.

5.3 Evaluation

This section evaluates the Penalty algorithm regarding its applicability and suitability for real-world examples. Furthermore, the previous discussion of running time and quality of routes is elaborated for further use cases. In addition, bottlenecks of the algorithm identified in the previous running time discussion are tried to overcome with various improvement suggestions.

The variety of input parameters makes it possible to have a relatively generalized algorithm design but also makes it harder to specialize the algorithm for specific use cases. Especially the finding of adequate values for the input parameters can be based on some theoretic assumptions but it is more an experimental task [18, 3].

Adapting the Penalty algorithm to other problems leaves the specification of the input parameters, especially γ and ϵ. A good and convenient way reaching this goal are practical experiments knowing the exact problem definition and the graph data to operate on. Furthermore, theoretical assumptions on the graph like maximum degree of the data to operate on can help to provide well-working parameters. The experiments made by Kobitzsch et al. are performed on various graph data. This leads to a reliable parameter setting of the penalty factors concerning the quality of

the resulting routes and the alternative graph based on the experiments.
Approaches to critical problems like route planning of self-driving cars should not be optimized at the cost of missing alternative routes. Therefore, it would be more important tackling the running time bottlenecks of the Penalty approach. For less critical problems, the running time can be improved by e.g. increasing the penalty factors. As γ is fixed in the RR problem, experiments can focus on e.g. ϵ to find a good trade-off between running time and the number of alternative routes.

5.3.1 Use Cases

Based on the theoretical discussion, the Penalty approach is suited for use cases where alternative routes are necessary that provide somehow real alternatives. Real alternatives mean that the routes do not differ by small detours but differ in longer contiguous parts. Depending on the underlying problem to solve, routes that share same parts are not a strength of the Penalty algorithm. The penalization of edges intuitively leads to routes that share only small parts or nothing.
RR as a use case or application of the Penalty approach demands a practically evaluated parameter setting in order to find interesting alternative routes on the one hand and reach a good running time on the other hand.

Furthermore, another use case is the planning of routes for drones that fly on the same height. There it is essential to have non-overlapping routes for drones with the same or similar start and end points. As mentioned above, routing of self-driving cars and balancing the amount of cars at a certain road could also lead to the necessity of non-overlapping or nearly non-overlapping routes.

A further use case that is expected to not work well with the Penalty approach is an additional lower-bound constraint. This use case is not expected to be solved efficiently by the Penalty approach without further adjustments. Complex overlapping constraints could also be hard to implement with the Penalty approach without loosing performance in running time.
The following gives a brief overview of possible improvements or where improvements are necessary.

5.3.2 Improvement Suggestions

Kobitzsch et al. as the authors of the improved Penalty approach suggest to use the Penalty approach as a speed-up technology for alternative route planning applying Dynamic Search Levels. Apart from that, the algorithm itself can and should be optimized and improved. First, the shortest path computation is restricted to basic algorithms, as dynamic edge weights require more advanced preprocessing schemes,

e.g. Customizable Contraction Hierarchies. For an advanced preprocessing scheme it would remain to clarify whether the additional effort for the preprocessing enables good speed-ups in total. Nevertheless, whenever performing a shortest path calculation on the original and not penalized graph (e.g. in the feasibility check), Contraction Hierarchies as a speed-up technique can be used. Basic performance improvements like using a bidirectional variant of the Dijkstra algorithm are also proposed in the paper by Kobitzsch et al. In general, parallelization can highly improve the running time. In a first step, the feasiblity checks can be fully parallelized as the graph is not modified. In addition, the check for the uniformly bounded stretch can itself be parallelized.

There are two more factors that mainly influence the running time: the number of steps until the convergence criterion is fulfilled and the uniformly bounded stretch check in the feasibility check. While the first one can be regularized with the input parameters and some knowledge of the underlying graph, the second one should be optimized. Besides the parallelization, a simple but efficient variant is to check only every ks time for the stretch condition. If k is determined depending on the length of the shortest path, it is more likely to not miss an unsatisfied stretch condition. Another possibility to estimate the distance between two points is to get an upper bound with another via-node probably not laying on the part of the path one wants to estimate. This situation is shown in Figure 11, where l_1 and l_2 are landmarks with precomputed distances to all nodes. The necessary preprocessing of the original graph and the selection of landmarks is done equivalently to the ALT algorithm [14]. This results in the following upper bounds: $d(u, v) \leq d(u, l) + d(l, v)$ where $d(s, t)$ is the length of the shortest path between s and t and l is a landmark. Taking the minimum over all landmarks leads to the tightest upper bound. We can also compute lower bounds by taking $d(u, v) \geq d(u, l) - d(v, l)$. Computing the maximum over all landmarks gives the tightest lower bound. The upper and lower bound can be used to decide whether it is necessary to check the uniformly bounded stretch condition exactly using exact shortest paths or discard the route without computing the shortest paths.
Further speed-up techniques like estimating shortest paths more efficiently like suggested by Gubichev et al. or pruning could also improve the running times [15, 19].

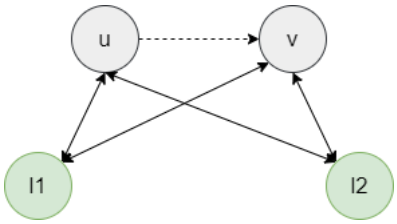


Figure 11 Illustration of bound estimation of shortest paths. In order to estimate the distance between u and v, the precomputed distances to the landmarks are used.

5.4 Summary

From a theoretical point of view, the Penalty algorithms is well-suited for RR. Adapting input parameters like $\gamma = 0$ intuitively leads to a possible algorithm for RR. The clear structure of the algorithm makes it possible to optimize and adapt parts of the algorithm. Furthermore, existing speed-up techniques can be directly applied in the algorithm and thus improve running times. Additionally, the route quality can be highly influenced by the input parameters and therefore be optimized for RR.

CHAPTER 6
Practical Suitability

In the following section, a first version without advanced improvements is implemented and evaluated. As already done in the theoretical suitability part, running time and quality of routes are evaluated. If not stated otherwise, running times are given in milliseconds and distances in meters (relevant for figures and tables).

6.1 Methodology

First, a frontend-backend architecture is taken to implement the algorithm and properly visualize it. Especially for measuring metrics, the overall running time of the algorithm is not of high importance. Thus, concurrency has not to be considered. Nevertheless, the running times give a hint of how suitable the Penalty approach is for real world applications like RR.
Second, the backend is able to track different metrics which are then evaluated using data visualization methods. The main metrics are difference in path lengths, overlap, running time, running time for path feasibility, and the feasibility optimization parameter including its influence to running time. Furthermore, good and bad examples of routes and their alternatives are visualized using the frontend.
Together, the setup allows an evaluation of the Penalty approach specifically for the adaptability to the RR problem.

6.2 Implementation

The architecture and design of the implementation is introduced in the following parts.

6.2.1 Design Decisions

A backend-frontend architecture is used for the overall setup of the system. Furthermore, various design decisions for the algorithm itself are highly important and thus implemented. First, the penalization of edges along the current found path are only performed on incoming edges of nodes of the path. This is plausible due to the fact that the aim of penalizing edges not laying on the path is to avoid hops. As

the edges on the path itself are strongly penalized, it is sufficient to penalize only incoming edges of nodes of the path.
Next, wherever possible, sets and maps are used for fast access. Concerning running time and performance, concurrent executions are not supported. It would be possible to e.g. perform the validation and feasibility of paths concurrently and speed that part up.
As the RR problem requires no overlap, the parameter for overlap (γ) is adapted. This results in higher running times. Thus, the feasibility checks are adapted using an additional parameter *skip_feasible*. *skip_feasible* indicates how many nodes shall be skipped while checking the stretch condition to avoid hops (instead of pair-wise checking).
Additionally, for the evaluation only the shortest path and the first alternative path are considered and evaluated. This leads to more realistic running times without concurrency. Furthermore, this reflects the complexity and balancing between Condition (1) and Condition (3).
Lastly, a bidirectional version of Dijkstra is used to find the shortest path between two nodes. This optimization results in notably better running times.

6.2.2 Backend

Java 11 in combination with Spring Boot is the framework used in the backend. It provides relatively easy and convenient ways to build REST controllers. Various endpoints for simply calling the Penalty algorithm but also for calling the metric endpoint exist. The metric endpoint outputs a CSV file containing multiple runs of the Penalty algorithm. Furthermore, Spring Boot enables user authentication and thread handling [1].
Controller, services and util classes like graphs build the architecture of the backend. In addition, the methods in the services have a test coverage covering all relevant cases and edge cases of the Penalty approach and the Dijkstra implementations. For this purpose, a test graph, which consists of 10 nodes and 9 edges, is used.

6.2.3 Frontend

The frontend is completely written in plain Javascript. In addition, it is mainly responsible for representing the results of the Penalty algorithm. Visualizing the paths is done with Leaflet [2] in combination with jQuery [3] and Bootstrap [4]. The frontend calls the backend using REST requests.

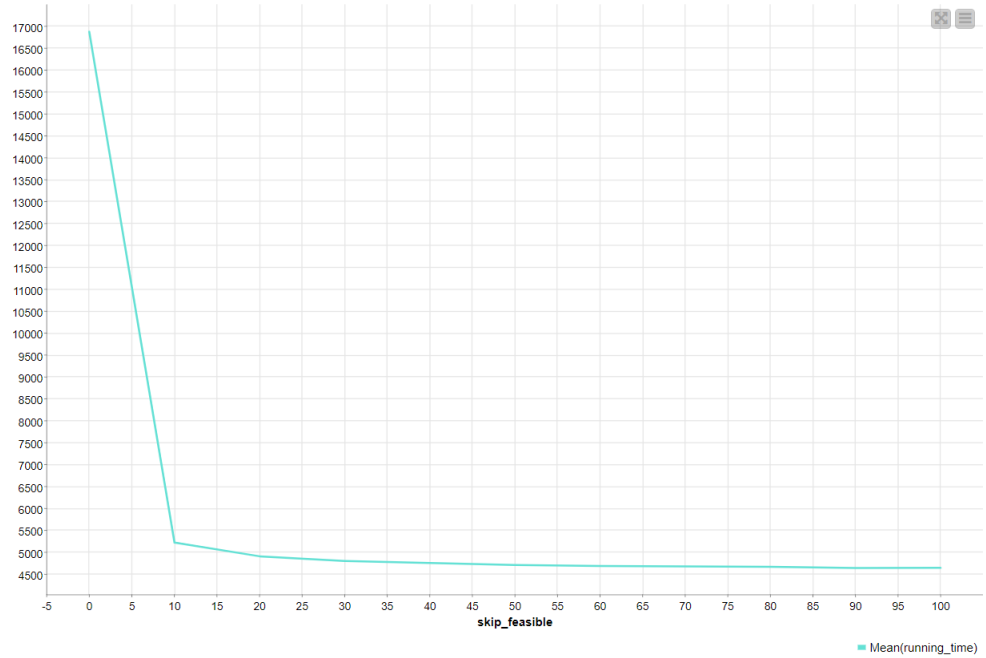
[1] https://spring.io/projects/spring-boot
[2] https://leafletjs.com
[3] https://jquery.com
[4] https://getbootstrap.com

6.3 Experimental Evaluation

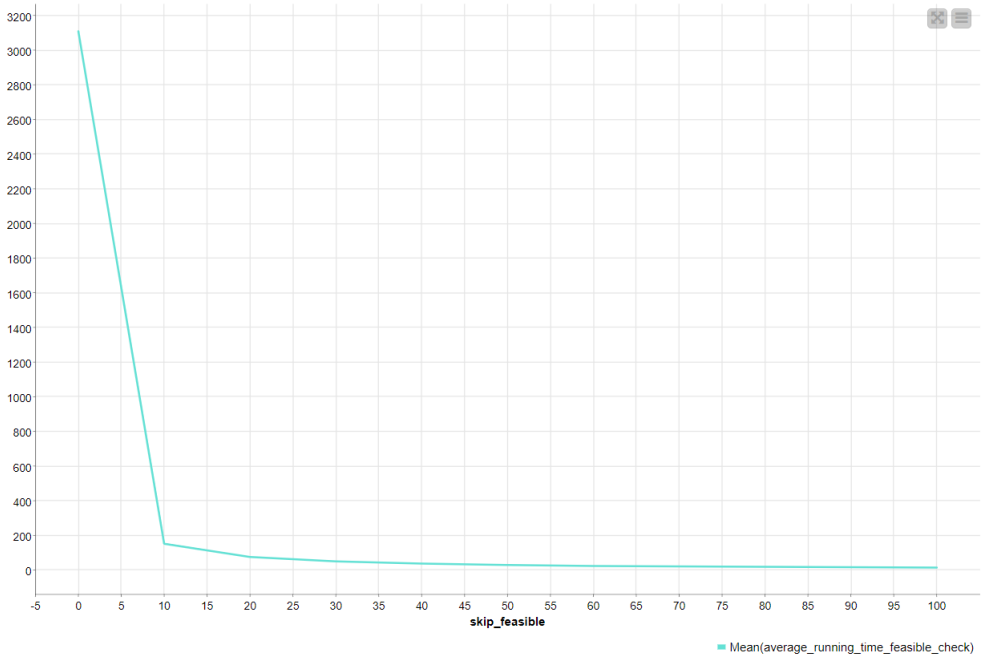
The following evaluates the results from the metrics of the Penalty implementation. First, the running time of the implementation is examined. Second, the quality of the resulting paths is analyzed.

6.3.1 Running Time Discussion

Figure 12b indicates the impact of the *skip_feasible* parameter on the running time of the feasibility check. The absolute gain from $skip\_feasible = 0$ to $skip\_feasible = 10$ is very high concerning the running time. Therefore, Figure 12a and Figure 12b emphasize the importance and impact of the feasibility check and the *skip_feasible* parameter.
Next, Figure 13, Figure 14 and Figure 15 provide insights about the running times in relation to the length of the shortest path. It is clearly visible that there exists a coherence between the length of the paths, the running time of the algorithm (Figure 14 and Figure 15) and the feasibility check (Figure 13). It could probably be a linear correlation, which would be a valid finding based on the design of the algorithm. Figure 16 represents the running time similar to Figure 12a but removing all results where no alternative paths were found. It seems that the running times are shorter when removing the results where only the shortest path was found. This could point to the necessity of a slight improvement of the converging criteria in the Penalty approach.
As the discussion above indicates, the running time of the Penalty approach mainly depends on the feasibility check. The feasibility check itself, especially checking for local optimality, is expensive and can be optimized. In order to get an overview for future work, the following part gives a better intuition of how great the impact of the current optimization using *skip_feasible* is.
Figure 17a allows more insights into the amount of running time required by the feasibility check in relation to the number of feasibility checks per run of the Penalty approach. Figure 17b shows the amount of running time the feasibility check requires related to the *skip_feasible* parameter. Altogether, one feasibility check takes a significant amount of time concerning the total running time. Concluding, the impact of the *skip_feasible* parameter is also highly important and should be considered for later improvements.
The Parallel Coordinates Plot (PCP) in Figure 18 provides a good representation that emphasizes the findings from above concerning e.g. the impact of the *skip_feasible* parameter. Additionally, it can be seen that most of the running times are below 10000ms. Further, the PCP using curved lines shows that there are not many outliers e.g. for the running times. Thus, the previous evaluation of the running times seems to hold for this data.



(a) Average running time of the Penalty algorithm depending on *skip_feasible*. The y-axis represents the average running time in milliseconds and the x-axis the *skip_feasible* parameter.



(b) Average running time of the feasibility check related to *skip_feasible*. The y-axis represents the average running time in milliseconds and the x-axis the *skip_feasible* parameter.

Figure 12 Figures representing the running time of the Penalty algorithm.

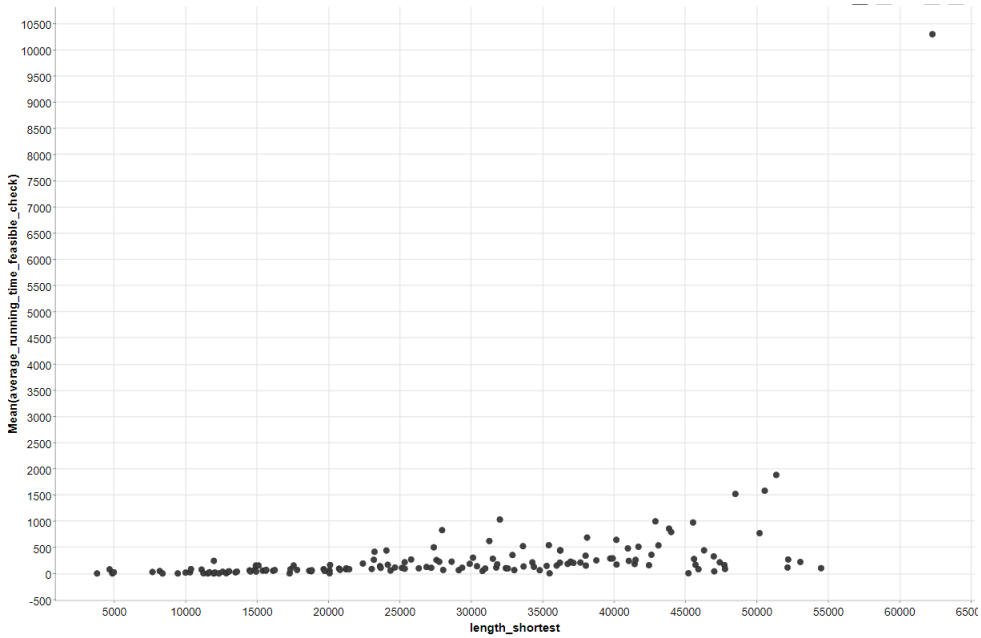


Figure 13 Average running time of the feasibility check depending on the length of the shortest path. The y-axis represents the average running time in milliseconds and the x-axis the length of the shortest path in meters.

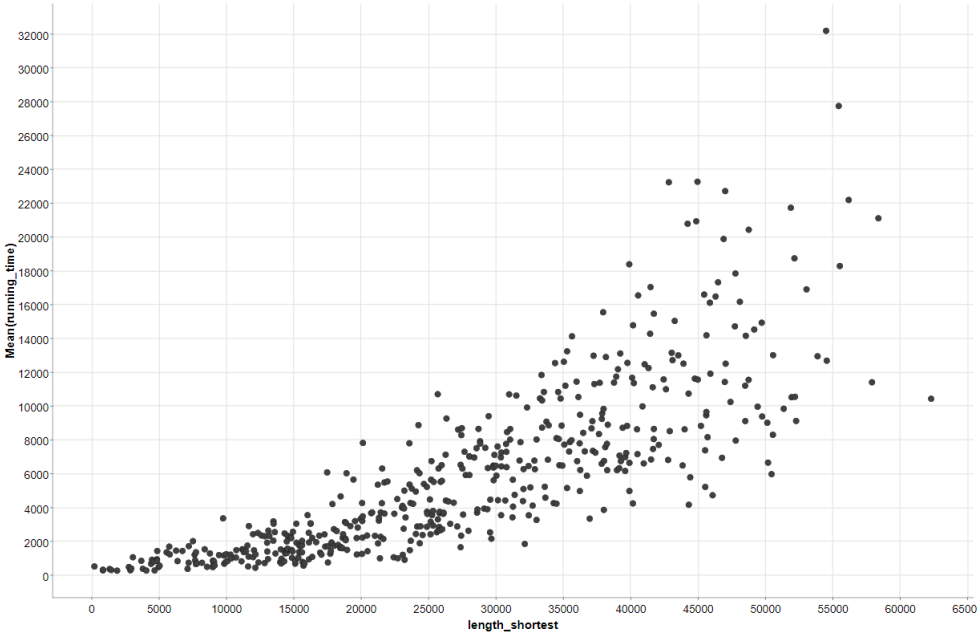


Figure 14 Average running time depending on length of shortest path. The y-axis represents the average running time in milliseconds and the x-axis the length of the shortest path in meters.

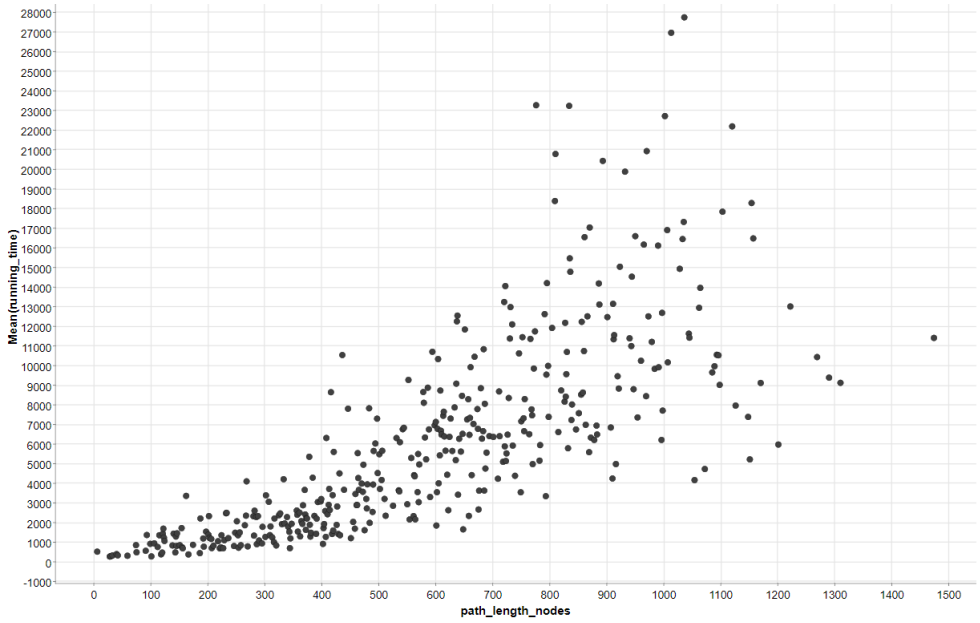


Figure 15 Average running time of the algorithm depending on the number of nodes of the shortest path. The y-axis represents the average running time in milliseconds and the x-axis the number of nodes.

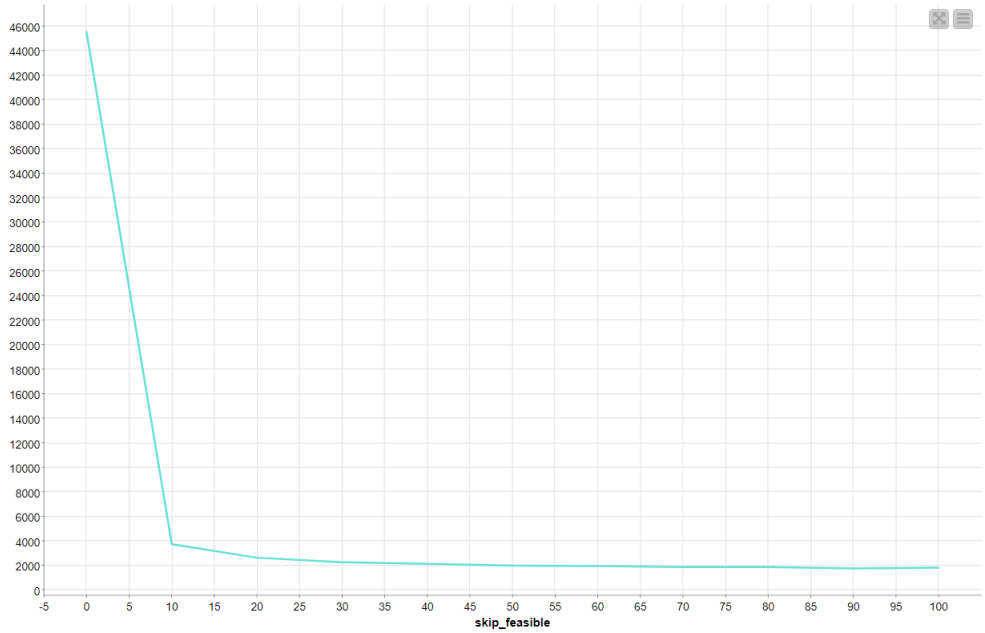
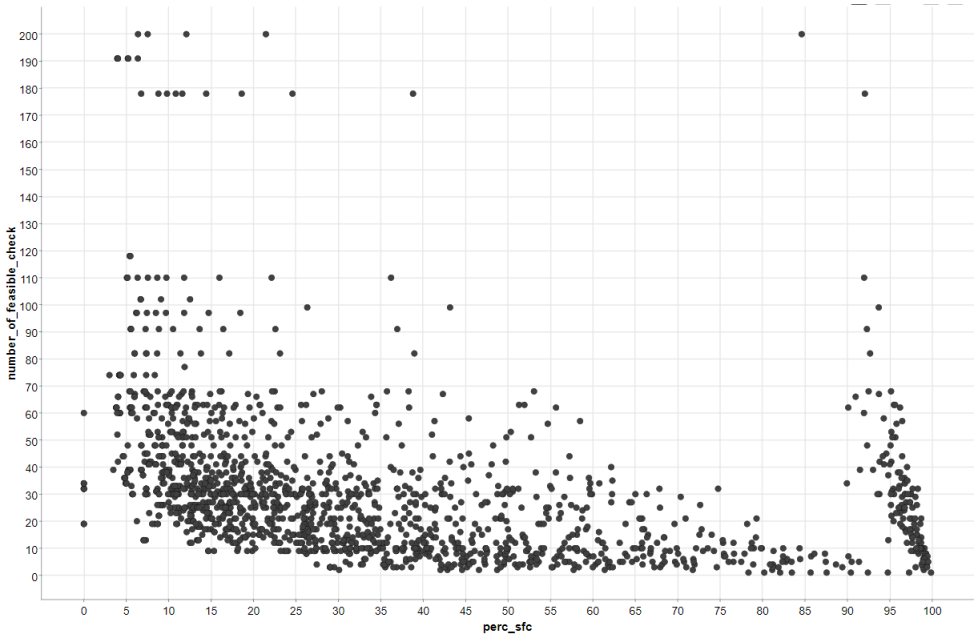
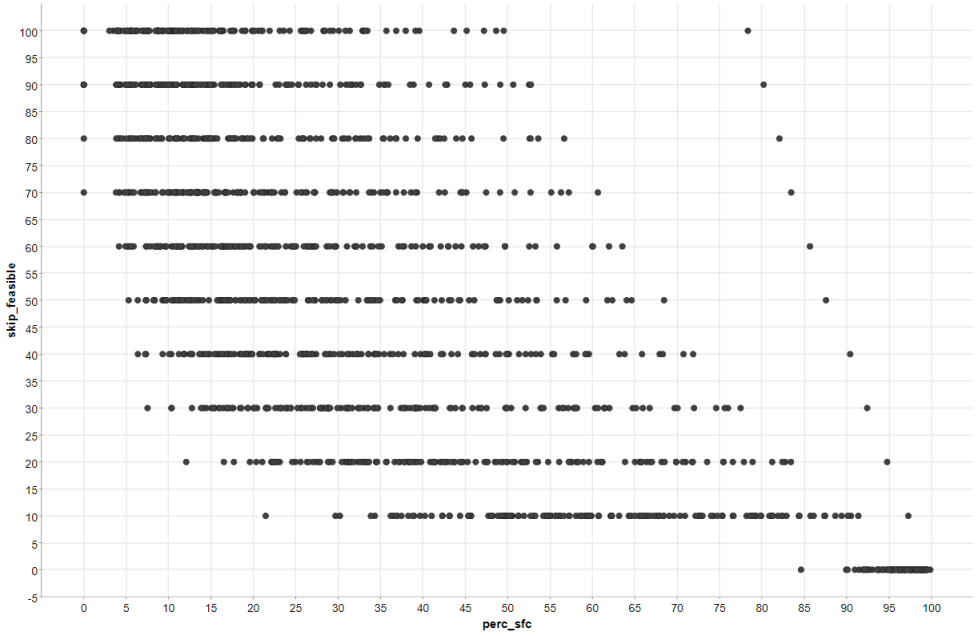


Figure 16 Average running time depending on *skip_feasible* but removing all results where no alternative route was found. The y-axis represents the average running time in milliseconds and the x-axis the *skip_feasible* parameter.



(a) X axis: percentage of feasibility check for total running time. Y axis: Number of feasible checks.



(b) X axis: percentage of feasibility check for total running time. Y axis: *skip_feasible* parameter.

Figure 17 Figures representing the running time of the feasibility check and the impact of the *skip_feasible* parameter.

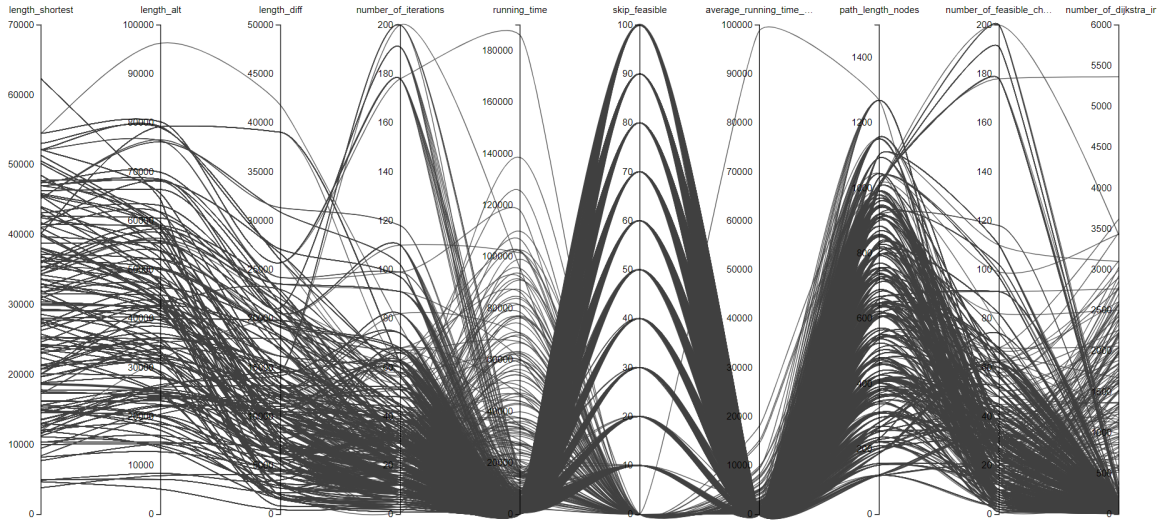


Figure 18 Parallel Coordinates Plot of metric data.

In summary, the overall running times without the usage of concurrency are good and there are many possibilities for improvements of the running time through concurrency and the feasibility check. With the current implementation, the duration of the feasibility check and thus the overall running time depends highly on the *skip_feasible* parameter. Additionally, the converging criterion for requests where no alternative routes are found is probably not sufficient in order to provide the same running times as for requests where at least one alternative route exists.

6.3.2 Quality Discussion

Apart from the running time discussion, the quality of the outcoming paths has to be considered as well. In the current setup for the metrics, the shortest path and the first alternative route are compared. Examining the length differences of the two paths gives a first insight of the quality. Comparing the differences relative to the length of the shortest path, the outcome of Figure 19 indicates that the quality of the results are satisfying. In fact, most of the alternative paths are less than half times longer than the shortest path. For the RR problem and especially Condition (1), the results could be better but are a good starting point.

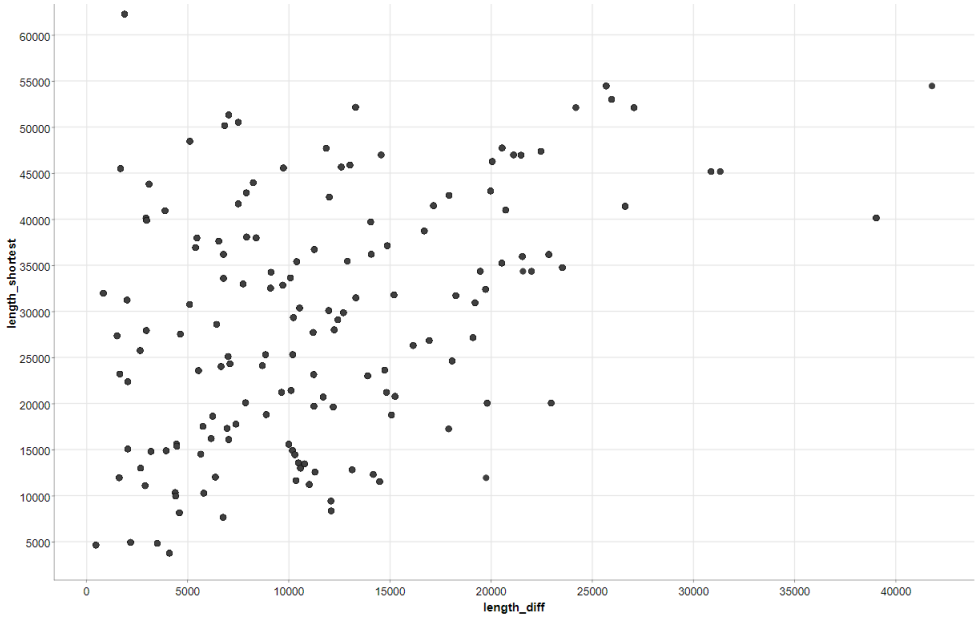


Figure 19 Plot showing the length differences of the found paths in relation to the length of the shortest path. The x-axis represents the length difference and the y-axis the length of the shortest path in meters.

In order to demonstrate good and bad examples of results of the Penalty algorithm, Figure 20 presents a very good result where the difference of the shortest path and the alternative is pretty low. In the following figures, the red path represents the shortest path and the blue paths stand for alternative routes.

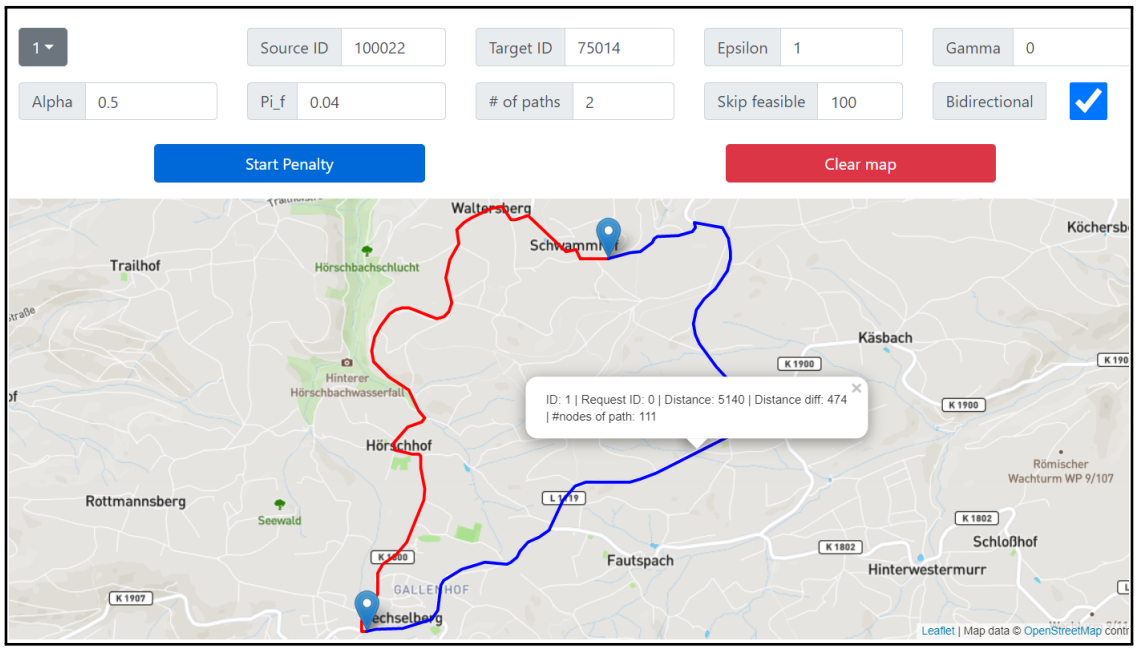


Figure 20 A good example for the RR problem. Setting: source=100022, target=75014, $\epsilon = 1$, $\gamma = 0$, $\alpha = 0.5$, $\pi_f = 0.04$, number of paths $= 2$, $skip\_feasible = 100$.

As an example where the algorithm has not performed well, Figure 21 shows a very long detour comparing to the shortest path. In order to allow for better discussion, Figure 22 additionally emphasizes the impact of the *skip_feasible* parameter. As it can be seen from the table, the parameter seems to have a huge impact on the quality of the result like the length differences between the found paths for higher values of *skip_feasible*.

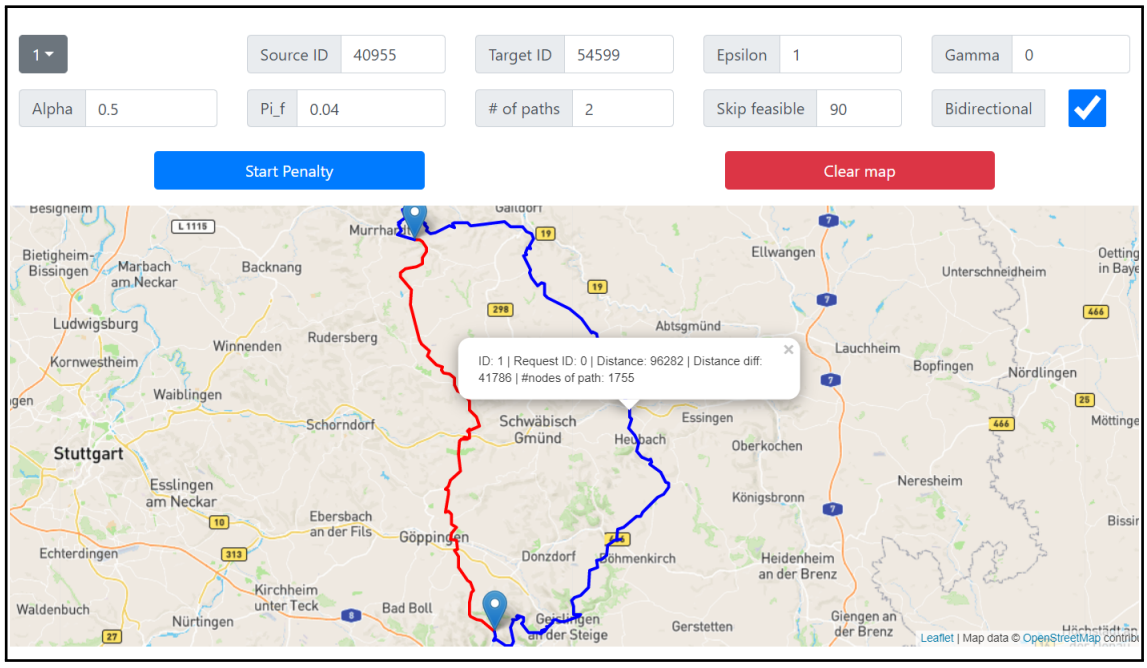


Figure 21 An example for the RR problem where the Penalty approach performed badly. Setting: source=40955, target=54599, $\epsilon = 1$, $\gamma = 0$, $\alpha = 0.5$, $\pi_f = 0.04$, number of paths = 2, $skip\_feasible = 90$.

source	target	skip_feasible	length_shortest	length_alt	length_diff	running_time
51818	24161	50	41,436	68,060	26,624	5008
51818	24161	60	41,436	68,060	26,624	4884
51818	24161	70	41,436	68,060	26,624	4658
51818	24161	80	41,436	68,060	26,624	4725
51818	24161	90	41,436	68,060	26,624	4595
51818	24161	100	41,436	68,060	26,624	4511
52826	79129	30	52,147	79,213	27,066	8941
52826	79129	40	52,147	79,213	27,066	8572
52826	79129	50	52,147	79,213	27,066	8207
52826	79129	60	52,147	79,213	27,066	8021
52826	79129	70	52,147	79,213	27,066	7997
52826	79129	80	52,147	79,213	27,066	7834
52826	79129	90	52,147	79,213	27,066	7869
52826	79129	100	52,147	79,213	27,066	7771
98746	8519	40	45,209	76,076	30,867	8147
98746	8519	60	45,209	76,076	30,867	7851
98746	8519	70	45,209	76,076	30,867	7619
98746	8519	80	45,209	76,076	30,867	7599
98746	8519	90	45,209	76,532	31,323	8747
98746	8519	100	45,209	76,532	31,323	8603
41939	39112	0	40,172	79,203	39,031	106782
41939	39112	10	40,172	79,203	39,031	9246
41939	39112	20	40,172	79,203	39,031	6631
41939	39112	30	40,172	79,203	39,031	5802
41939	39112	40	40,172	79,203	39,031	5359
41939	39112	50	40,172	79,203	39,031	5080
41939	39112	60	40,172	79,203	39,031	4914
41939	39112	70	40,172	79,203	39,031	4818
41939	39112	80	40,172	79,203	39,031	4738
41939	39112	90	40,172	79,203	39,031	4633
41939	39112	100	40,172	79,203	39,031	4580
40955	54599	90	54,496	96,282	41,786	7133

Figure 22 Understanding the impact of the *skip_feasible* parameter to the quality of results. Running times in ms.

Taking the same source and target node as in the example in Figure 21, Figure 23 shows more alternative paths. What can be observed are the small hops in the alternative routes. This is due to the implementation explained in Chapter 4. Furthermore, considering more alternative paths than only the first one, the results for this example could not have been better, as the alternative routes are also overlapping near the source and target. Nevertheless, the *skip_feasible* parameter is responsible for the example in Figure 21 and more alternative paths can lead to better results for the RR problem.

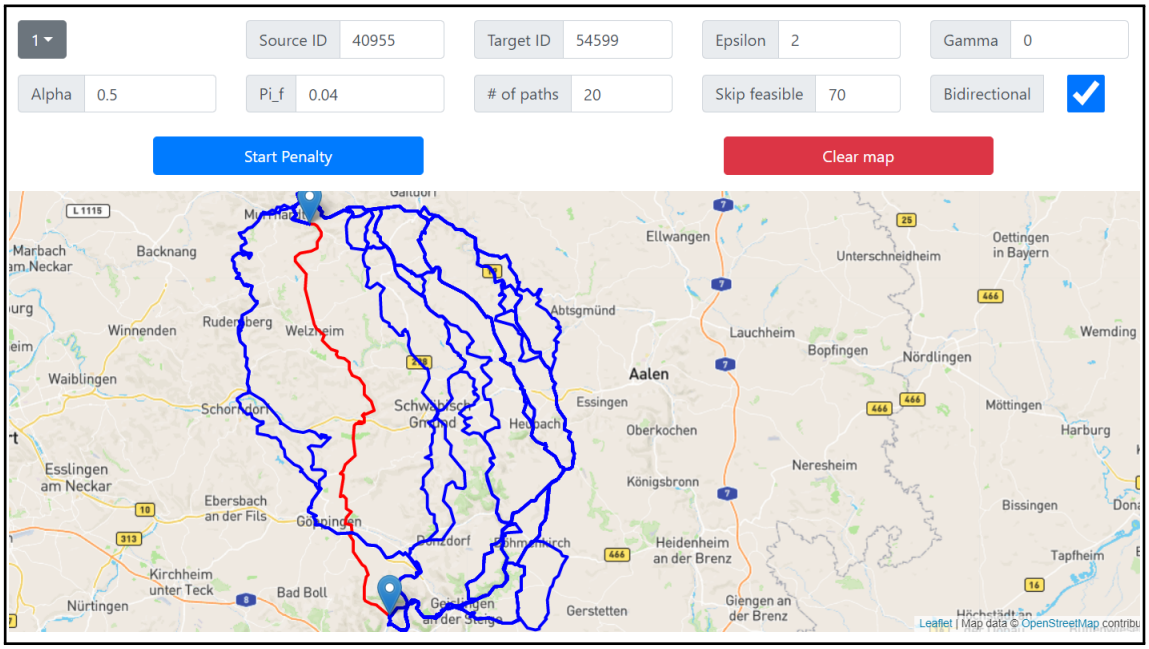


Figure 23 Penalty algorithm: Other alternatives can lead to better results for the RR problem. Setting: source=40955, target=54599, $\epsilon = 2$, $\gamma = 0$, $\alpha = 0.5$, $\pi_f = 0.04$, number of paths = 20, $skip\_feasible = 70$.

6.4 Summary

The practical evaluation of the running time of the Penalty algorithm and the quality of the results gives a better understanding of the algorithm's weaknesses and ways to improve it. In detail, the running time of the algorithm highly depends on the feasibility check and the *skip_feasible* parameter. Further, the quality of the results also depends on the *skip_feasible* parameter. In summary, the running time of the algorithm seems to be good enough for taking the Penalty approach as a basis for RR including further improvements. The quality of the paths is sufficiently good. The results from the practical suitability study are confirmed by the theoretical suitability study. Additionally, the Penalty algorithm is highly adaptable. Therefore, the algorithm is suited for the RR problem. Furthermore, many ways to improve the route quality and efficiency of the algorithm exist.

CHAPTER 7
Improved Penalty Algorithms

Taking the results concerning the theoretical suitability in Chapter 5 and the practical suitability in Chapter 6, improved versions of the Penalty algorithms for RR can be implemented. First, *Contraction Hierarchies (CH)* are introduced as a speed-up technique for shortest path queries [13]. Afterwards, an improved version of the Penalty algorithm for RR is presented including various performance improvements. Additionally, improvement suggestions from Section 5.3.2 are included in the implemented algorithms.

7.1 Contraction Hierarchies

The main idea of Contraction Hierarchies is a preprocessing of the graph in order to speed up the Dijkstra algorithm. Therefore, Geisberger et al. [13] have introduced a preprocessing step that keeps shortest path distances between all nodes. The following terms are necessary for the preprocessing step:

- *Shortcuts*: Shortcuts are added to a graph as special edges. Shortcuts share the same properties as edges but are labeled differently to distinguish them from normal edges. For shortest path queries on a modified graph including shortcuts, shortcuts are treated as normal edges. For CH, shortcuts are added whenever a node contraction changes the shortest path length between two remaining nodes. Then, a shortcut is added between the two affected nodes with an edge weight of the original shortest path length.

- *Node contraction*: If a node is contracted, the node and all incident edges are removed from the graph. This possibly affects other properties of nodes like the node degree.

- *Rank*: The rank of a node is determined by the number of nodes already contracted in a graph. In order to assign a rank for each node, all nodes have to be contracted.

- *Edge difference (ED)*: The edge difference is used to decide which node to contract next. ED for node v is calculated as follows:
  $ED(v) = -(\#\text{ outgoing edges of } v + \#\text{ incident incoming edges of } v) +$
  $\#\text{ of shortcuts to add if removing } v$
  The ED tries to sort the nodes of a graph by their importance in the graph.

> The number of shortcuts that are added if node v is removed can be determined by simulating the removal of node v and checking possible affected shortest path lengths between all remaining node pairs.

Instead of always checking all node pairs in a graph for affected shortest path lengths, it suffices to check all neighboring nodes of the current node v. This heavily speed ups the preprocessing step. In the preprocessing step itself, all edge differences are calculated in a first step and are stored in a priority queue. The lower the ED of a node, the less important is the node in the graph. In order to keep the number of inserted shortcuts low (roughly the number of edges), the least important node according to the ED is selected to be contracted next. The contraction possibly leads to updates of the ED of neighboring nodes. This step of node contraction is repeated until an empty graph is left.
Then, a graph including node ranks and all shortcuts can be returned. This slightly modified graph is used for CH queries.

7.1.1 Queries

For CH queries, the graph resulting from the peprocessing step is used. We further define *upward* and *downward* edges/shortcuts. Let $r(v)$ be the rank of node v.

- *upward edge/shortcut*: For an upward edge/shortcut $e = (v, w)$, it holds that $r(v) \leq r(w)$.
- *downward edge/shortcut*: For a downward edge/shortcut $e = (v, w)$, it holds that $r(v) > r(w)$.

In order to perform a CH query from start node s to target node t, a bidirectional Dijkstra variant is used where one run starts from s and the other one from t. Furthermore, the run starting from s is restricted to only consider upward edges/shortcuts. The run starting from t only considers downward edges/shortcuts in the graph.
The correctness proof is omitted but given in [13]. Additionally, Geisberger et al. show that more engineering in the basic CH algorithm leads to way better running times [13]. For this work, the standard implementation as explained above is used including basic engineering like parallelizations in the preprocessing step. As the path found by the algorithm can contain shortcuts, the contracted edges of a shortcut have to be tracked in order to support fast unpacking and returning of a path consisting only of edges without shortcuts.

7.2 Performance-oriented Penalty Algorithm

Apart from CH and a bidirectional Dijkstra variant as an improvement for shortest path computations, parallelization as a running-time-shortening method is imple-

mented. In Algorithm 1 it is quite obvious that parallelization is only applicable in the feasibility check. Here, however, the load can be nicely balanced over the available cores.
In addition, the proposed solution in Section 5.3.2 for the feasibility check (using ALT to estimate shortest path distances) is outperformed by directly applying CH. Nevertheless, shortest path estimations and engineering can further improve the running time of the algorithm. The CH shortest path computation can always be applied if requests are made on the original graph without penalties. This is the case in the feasibility check and the initial shortest path computation. For the repeating shortest path computations in the while-loop in Algorithm 1 on the modified graph, the (bidirectional) Dijkstra is applied as, in contrast to CH, no preprocessing is necessary. In the feasibility check, theoretically, the performance gain of the CH algorithm in comparison to the bidirectional Dijkstra variant is the same. Nevertheless, using CH instead of a bidirectional Dijkstra variant could make a practical difference.

7.2.1 Data Structures

A good choice of data structures is essential in order to increase the performance of the implementation. Especially maintaining map and set data structures in the graph for parent pointers, shortcuts (for CH), and all edges enables fast lookups and calculations of set differences. Additionally, the priority queues should support fast insertions and extractions.
If only one data structure is used to store the whole graph with all possible modifications like in CH, subgraphs can be stored using sub lists or checking whether (deleted) elements are contained in a set or list. For this work, the aforementioned approach is chosen combining fast lookups and relatively small space overhead for supporting e.g. CH.

7.2.2 Alternative Route Quality

Beside the performance improvements, various possibilities exist to improve the quality of alternative routes. First, $\gamma = 0$ is fixed by the problem definition of RR. Next, ϵ is a crucial parameter for the quality and number of alternative routes but highly depends on the use case, whereas π_f and α can also influence the alternative routes. In detail, α is used to penalize incoming edges of nodes lying on the current found path. Thus, returning to an already found path can be prevented by choosing a high value for α. This is only relevant if $\gamma > 0$. Then, an efficient postprocessing of the alternative graph has to be developed. This work and the next section focus on the evaluation of changes of α, ϵ, and γ. However, changes of γ are just evaluated for

the running time discussion as the focus of this work does not lie on the development of an efficient postprocessing of an alternative graph.

CHAPTER 8
Experimental Evaluation

This chapter contains the evaluation of different versions of the Penalty algorithm for RR presented in Chapter 7. If not stated otherwise, running times are given in seconds and distances/lengths in meters. The evaluation is split in a running time discussion and a quality discussion. Different versions of the Penalty algorithm result from a basic implementation without improvements, a bidirectional Dijkstra variant, another with Contraction Hierarchies implemented, and a last one including parallelization. Additionally, meaningful combinations of the improvements are implemented.
Furthermore, various parameter settings are evaluated in order to demonstrate the flexibility of the algorithm and the impact on running time and quality. First, the running time is discussed in the following section. Afterwards, the route quality with different parameter settings is discussed.

8.1 Setup

For the experiments, different parameter settings are used and represented in Table 1. These settings are mainly for evaluating impacts of different values for α, γ, and ϵ where the first setting includes the default values proposed by Kobitzsch et al. [18]. The purpose of settings 2 and 4 is to demonstrate a changing route quality for other values for γ. Nevertheless, $\gamma = 0$ is fixed for RR.
A higher value for α does not make sense because with a higher value the risk of missing routes is higher. For ϵ, the value highly depends on the use case for RR. In order to give an overview of the impact on the running times, $\epsilon \in \{0.25, 0.5, 0.75\}$ are chosen.

ID	γ	α	ϵ
1	0	0.5	0.5
2	0.2	0.5	0.5
3	0	0.3	0.5
4	0.2	0.3	0.5
5	0	0.5	0.25
6	0	0.5	0.75
7	0	0.3	0.75

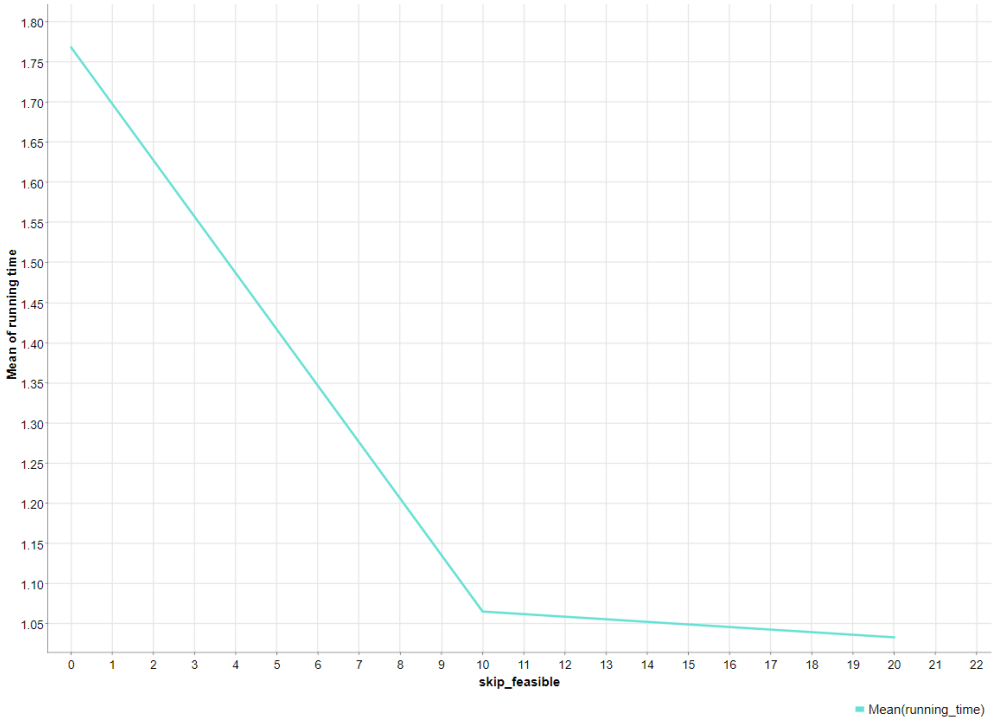
Table 1 Different parameter settings for the experiments concerning γ, α, and ϵ.

In addition to the input parameter settings for the Penalty algorithm, all combinations of improvements (CH, bidirectional Dijkstra, parallelization) are executed for every choice of Table 1 for every node pair. Additionally, the number of paths to be found is set to one, two, and three and the algorithm can terminate earlier if the number of alternative paths is reached or the convergence criterion is met in the while-loop in Algorithm 1. For the experiments, 1560 node pairs (random start node and random target node) are chosen and the corresponding metrics collected.

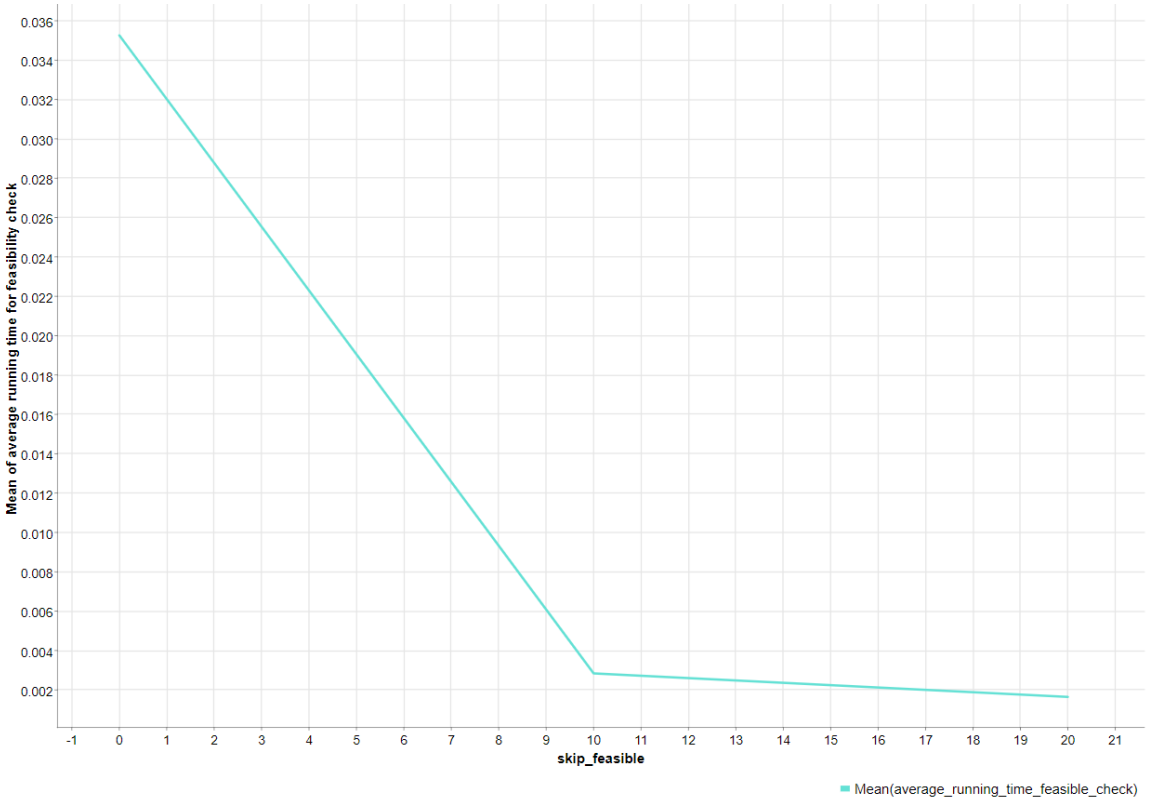
For the next evaluation steps, we fix various parameters in order to evaluate e.g. running times in a meaningful way. If a parameter is not fixed, all observations are included in the evaluation, e.g. if ϵ is not restricted, all values of ϵ are allowed that are contained in Table 1.

8.2 Running Time Discussion

As an initial step, the parameter $skip\_feasible$ is evaluated. The parameter indicates the number of checks for the uniformly bounded stretch that are skipped. In Chapter 6 it is already shown that this parameter has a huge influence on the running time especially for values $2 \leq skip\_feasible \leq 10$. Figure 24a and Figure 24b emphasize this finding. The combination of both figures indicates that the feasibility check highly influences the overall running time of the algorithm.
Nevertheless, the goal of this work is to develop an efficient algorithm where such skips of feasibility checks are not necessary anymore. Thus, the further discussion focuses on a fixed parameter $skip\_feasible = 0$. First, the default values for RR are taken to examine the running time of the algorithm. Therefore, $\gamma = 0$ is fixed per definition of RR and $\alpha = 0.5$ as proposed by Kobitzsch et al. [18]. Figure 25a shows that the running times are positively correlated to the length of the shortest path using the default parameter setting. In contrast, the correlation of the average running time of the feasibility check and the length of the shortest path using the default parameter setting is not that clear as shown in Figure 25b. But a tendency to a positive correlation is given. These findings strengthen the intuition and understanding of the Penalty algorithm and the points where improvements are sensible. In detail, higher lengths of the shortest paths lead to more nodes in many cases. Therefore, the algorithm probably has to perform more feasibility checks.
In order to confirm the tendency from Figure 25b, the length of the shortest path is not always a necessary indicator. Figure 26a and Figure 26b confirm the tendency of a positive correlation between the running time of the algorithm and the number of nodes of the shortest path using the default parameter setting. The number of nodes of the shortest path is actually a more precise measure. This is reflected by less outliers than in Figure 26a and Figure 26b compared to Figure 25a and Figure 25b.

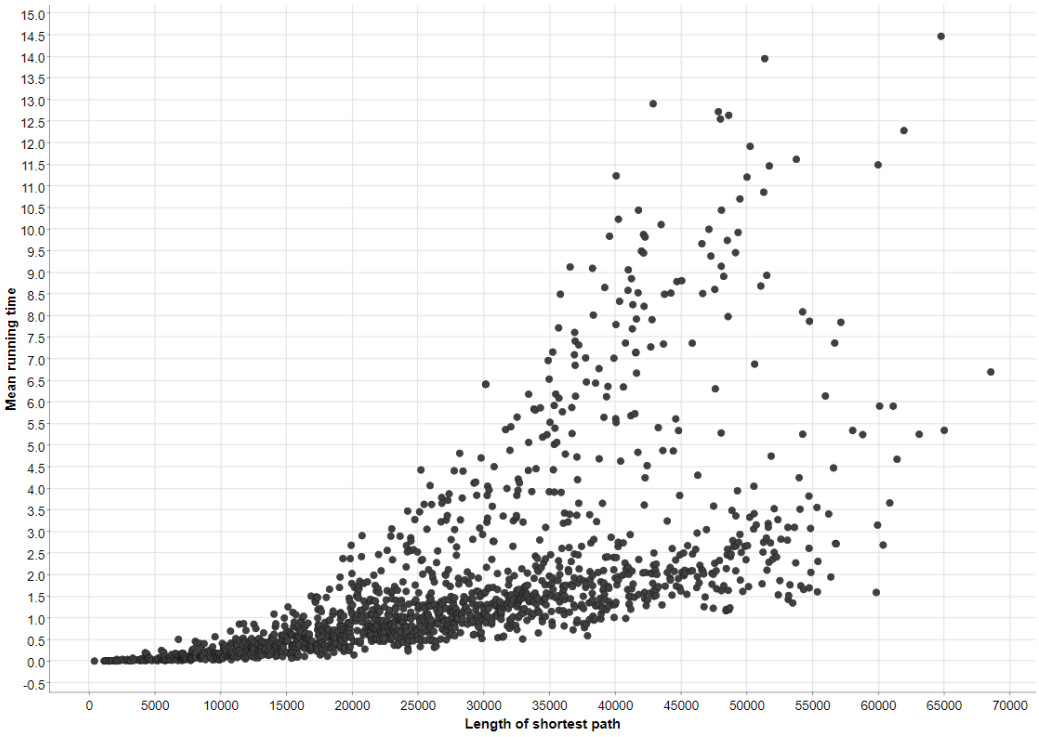


(a) Parameter $skip\_feasible$ in relation to the running time with parameters $\gamma = 0, \alpha = 0.5$. The y-axis represents the running time in seconds and the x-axis the $skip\_feasible$ parameter.

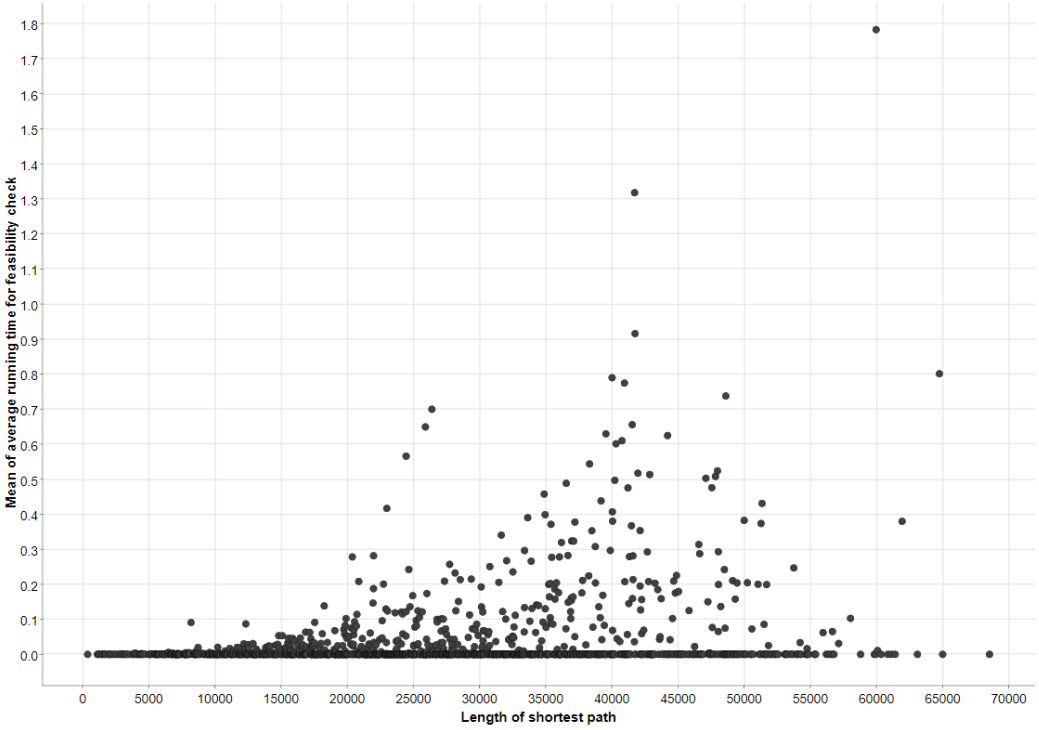


(b) Parameter $skip\_feasible$ in relation to the average running time for the feasibility check with parameters $\gamma = 0, \alpha = 0.5$. The y-axis represents the average running time in seconds and the x-axis the $skip\_feasible$ parameter.

Figure 24 Evaluation of parameter $skip\_feasible$.

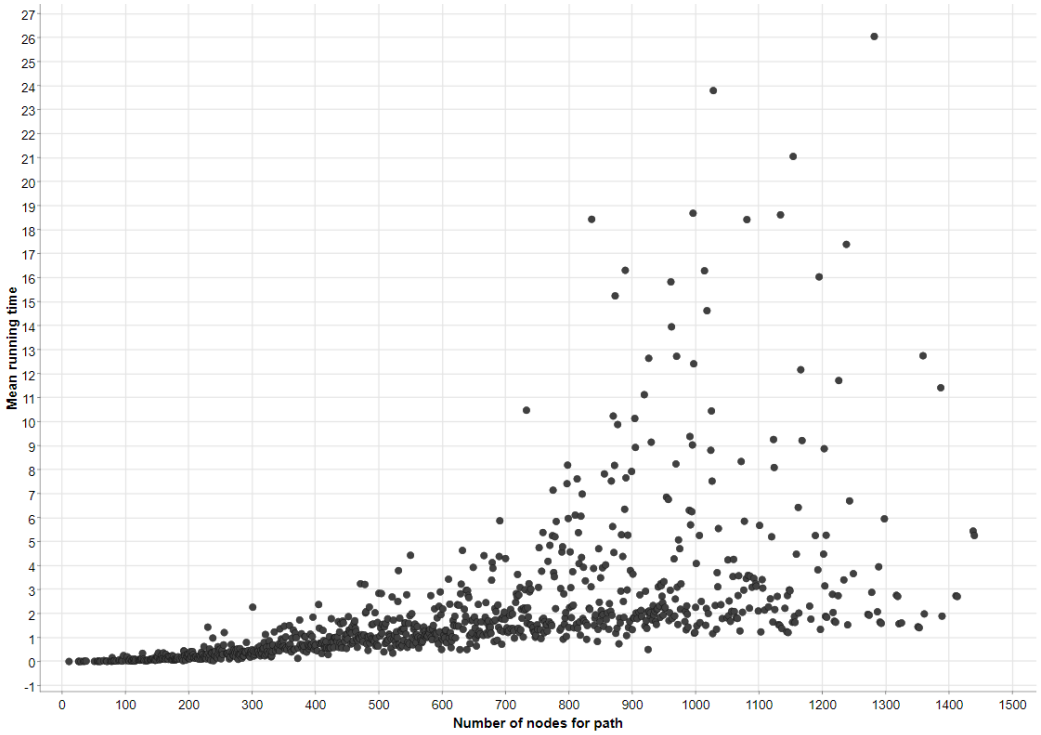


(a) Evaluation of the running time in relation to the length of the shortest path with parameters $\gamma = 0, \alpha = 0.5, skip\_feasible = 0$. The y-axis represents the running time in seconds and the x-axis the length of the shortest path in meters.

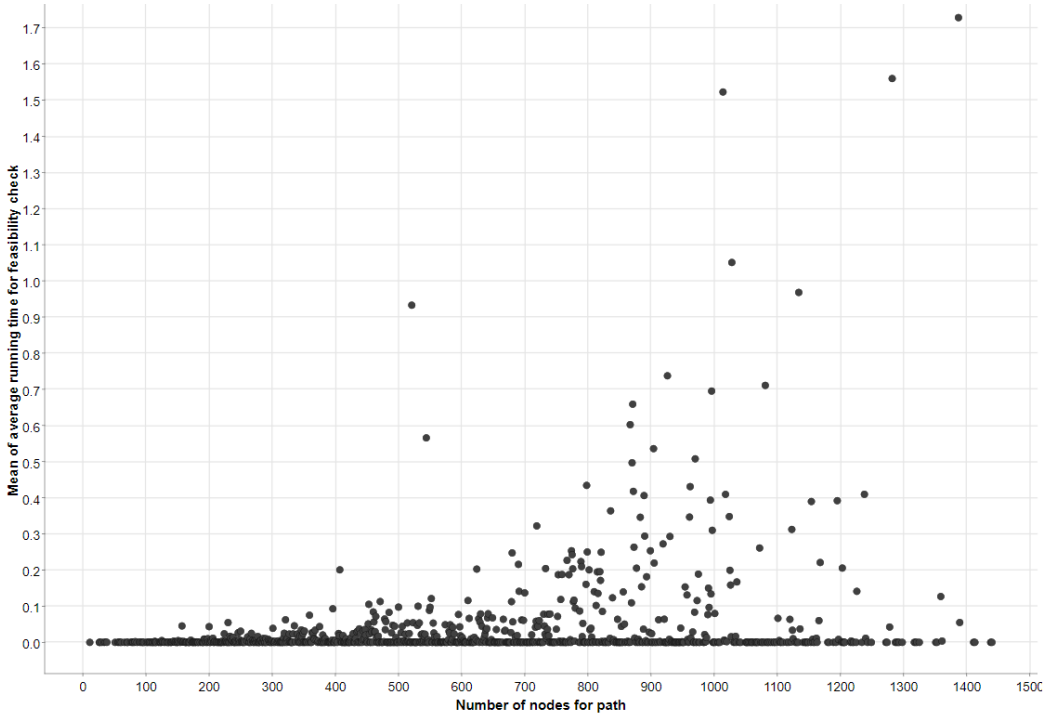


(b) Evaluation of the average running time of the feasibility check in relation to the length of the shortest path with parameters $\gamma = 0, \alpha = 0.5, skip\_feasible = 0$. The y-axis represents the average running time in seconds and the x-axis the length of the shortest path in meters.

Figure 25 Overall running time evaluation with fixed $skip\_feasible$ parameter.



(a) Evaluation of running time in relation to the number of nodes of the shortest path with parameters $\gamma = 0, \alpha = 0.5, skip\_feasible = 0$. The y-axis represents the running time in seconds and the x-axis the number of nodes.



(b) Evaluation of the average running time of the feasibility check in relation to the number of nodes of the shortest path with parameters $\gamma = 0, \alpha = 0.5, skip\_feasible = 0$. The y-axis represents the average running time in seconds and the x-axis the number of nodes.

Figure 26 Overall running time evaluation with fixed $skip\_feasible$ parameter in relation to the number of nodes for the shortest path.

Another interesting measure is the maximum and average running time of the implemented algorithms given fixed input parameters of the algorithm for a comparative evaluation. As it can be seen in Table 2, Table 3, and Table 4, the combination of all three optimizations performs best. Additionally, Table 3 emphasizes the impact of the bidirectional Dijkstra where Table 2 and Table 4 focus on the impact of CH and parallelization. The combination of CH and parallelization seems to be a strong improvement. The parallelization also impacts the running time especially for the combination without CH and the bidirectional Dijkstra. As the last row in Table 4 indicates, CH is the most important improvement concerning running time in comparison to parallelization and the bidirectional Dijkstra.

CH	**Bidirectional Dijkstra**	**Parallelization**	**Max running time (seconds)**
true	true	true	24.243
false	true	true	74.763
false	true	false	110.44
true	true	false	24.249

Table 2 Maximum running times evaluation with parameters $\gamma = 0, \alpha = 0.5, skip\_feasible = 0, \epsilon = 0.5$.

CH	**Bidirectional Dijkstra**	**Parallelization**	**Average running time (seconds)**
true	true	true	0.953
false	true	true	1.193
false	true	false	1.802
true	true	false	0.976

Table 3 Average running times evaluation with parameters $\gamma = 0, \alpha = 0.5, \epsilon = 0.5, skip\_feasible = 0$.

CH	**Bidirectional Dijkstra**	**Parallelization**	**Average running time (seconds)**
true	false	true	1.037
false	false	true	1.296
false	false	false	1.994
true	false	false	1.06

Table 4 Average running time evaluation with parameters $\gamma = 0, \alpha = 0.5, \epsilon = 0.5, skip\_feasible = 0$.

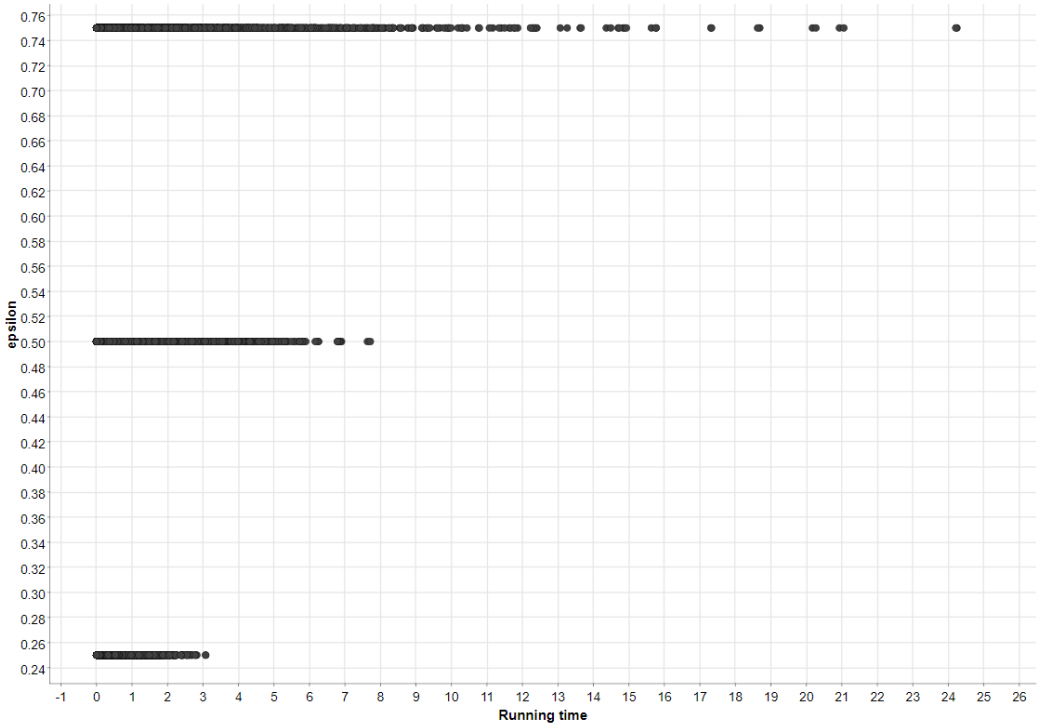


Figure 27 Evaluation of the running time in relation to ϵ with parameters $\gamma = 0, \alpha = 0.5, skip\_feasible = 0$ and with CH, bidirectional Dijkstra, and parallelization. The x-axis represents the running time in seconds and the y-axis ϵ.

Apart from investigating the running time in dependence of improving techniques like CH, the running times further depends on the input parameters. Therefore, we fix the observations to the one where CH, bidirectional Dijkstra, and parallelization is applied because the figures above have shown that this combination performs best. As Figure 27 and Figure 28 indicate, higher values for ϵ and lower values for γ increase the running time of the algorithm. As ϵ specifies the maximum length of alternative routes, it is sensible that higher values for ϵ lead to higher running times. As the algorithm is designed to terminate after a given number of alternative paths have been found, a higher value for γ leads to faster results.
For the input parameter α, no difference in the running time can be observed in Figure 29. For $\alpha = 0.3$, some outliers with high running times can be detected but apart from that no tendency as for ϵ and γ can be detected.
Figure 30 completes the evaluation of the impact of the length of the shortest path to the running time of the algorithm. A positive correlation can be confirmed including all improvements and an adequate parameter setting required for RR ($\gamma = 0$). Furthermore, this figure indicates that no parameter setting exists where the positive correlation is not given. This extends the generality of the claim in comparison to Figure 25a and allows for further experiments with different input parameters.

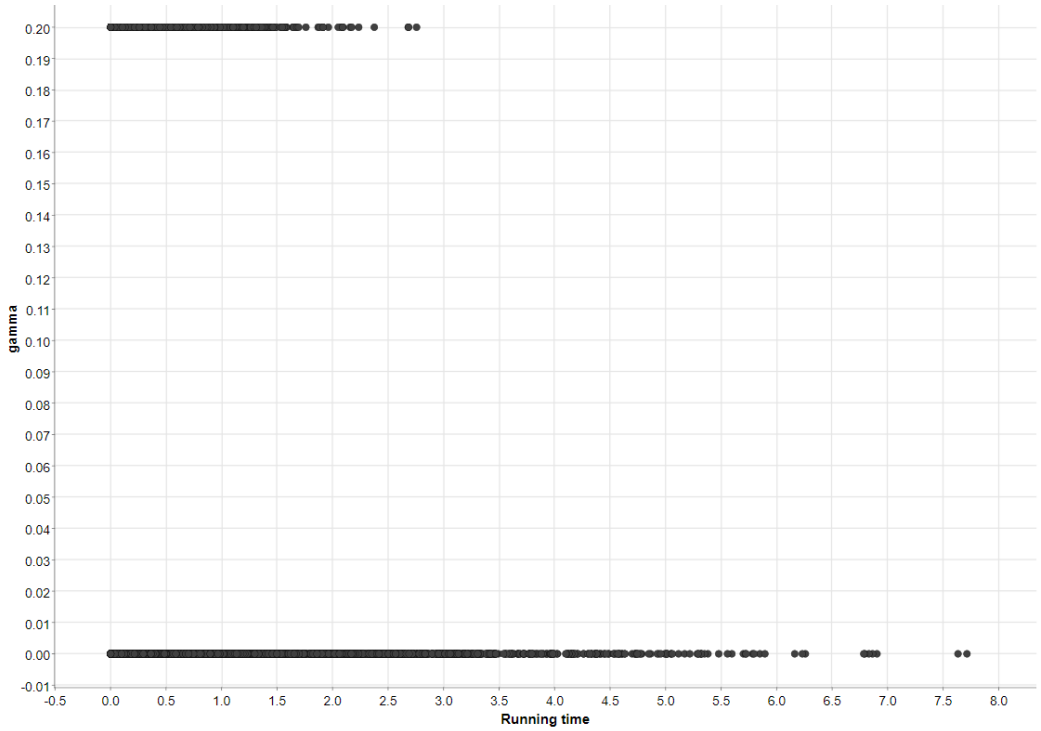


Figure 28 Evaluation of the running time in relation to γ with parameters $\epsilon = 0.5, \alpha = 0.5, skip\_feasible = 0$ and with CH, bidirectional Dijkstra, and parallelization. The x-axis represents the running time in seconds and the y-axis γ.

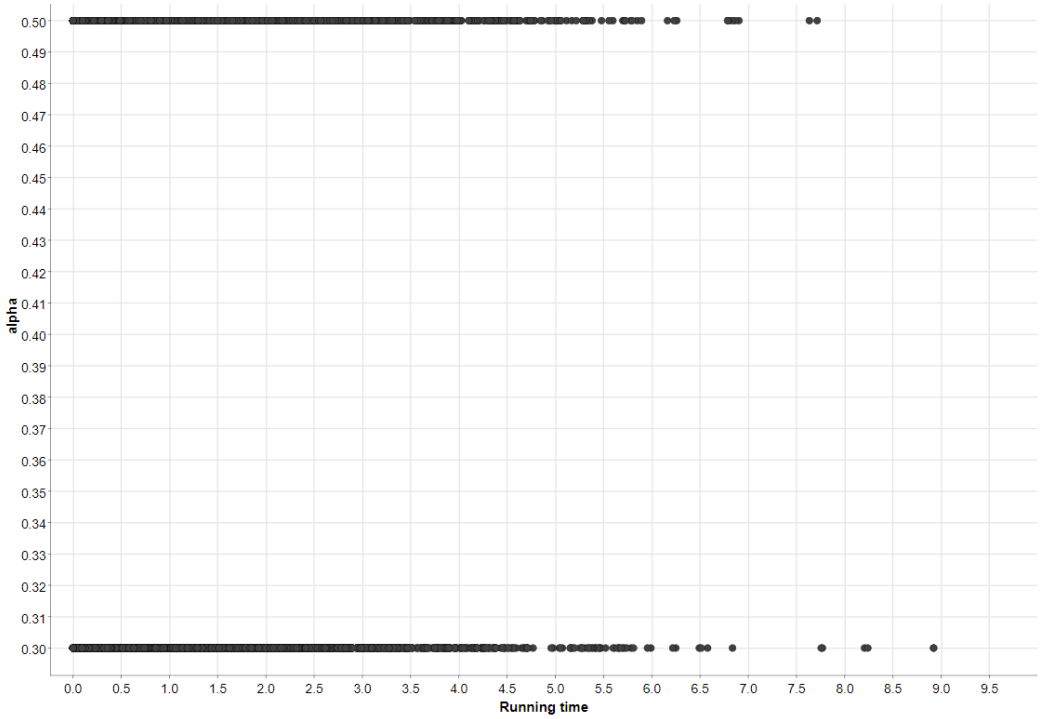


Figure 29 Evaluation of the running time in relation to α with parameters $\gamma = 0, \epsilon = 0.5, skip\_feasible = 0$ and with CH, bidirectional Dijkstra, and parallelization. The x-axis represents the running time in seconds and the y-axis α.

A possibly misleading correlation is presented in Figure 31.What can be observed is that the convergence criterion in the Penalty algorithm seems to be reached later the less paths are found. As the algorithm terminates as soon as the number of specified paths are found, this could explain the faster running times for three found paths and the slower running times for only one path. But the dataset does not contain that many node pairs where three paths are found. Therefore, no reliable evidence can be found for a correlation between the number of found paths and the running time.

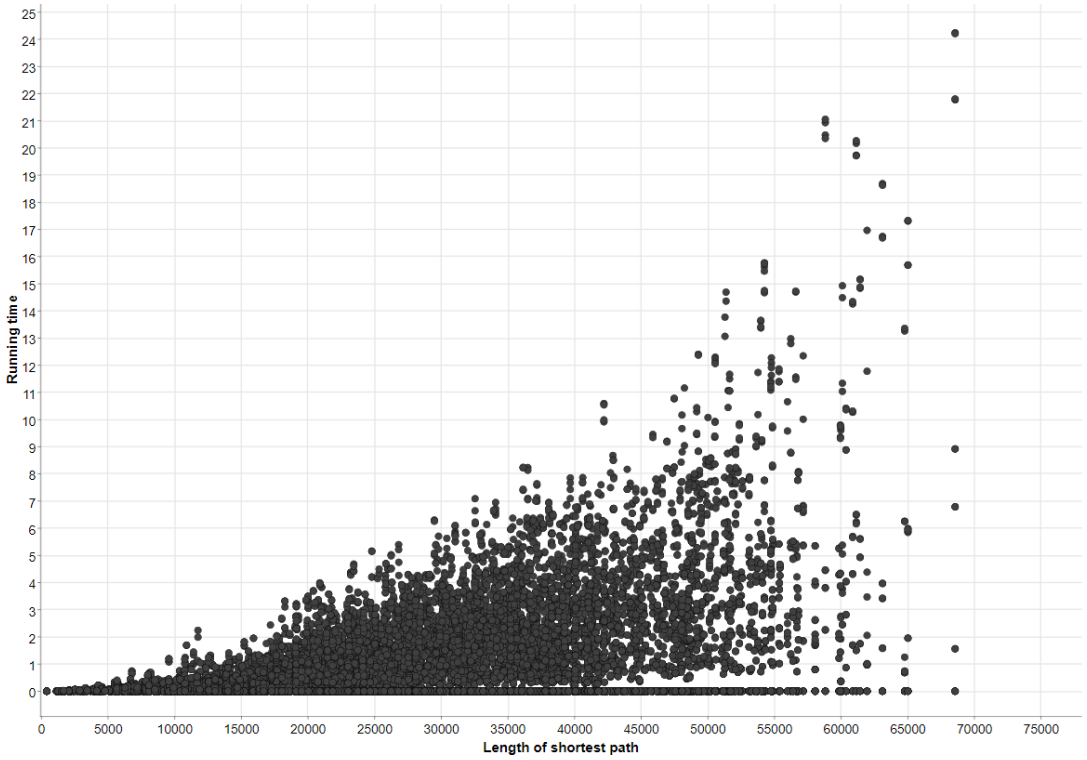


Figure 30 Evaluation of the length of the shortest path in relation to the running time with parameters $\gamma = 0, skip\_feasible = 0$ and with CH, bidirectional Dijkstra, and parallelization. The y-axis represents the running time in seconds and the x-axis the length of the shortest path in meters.

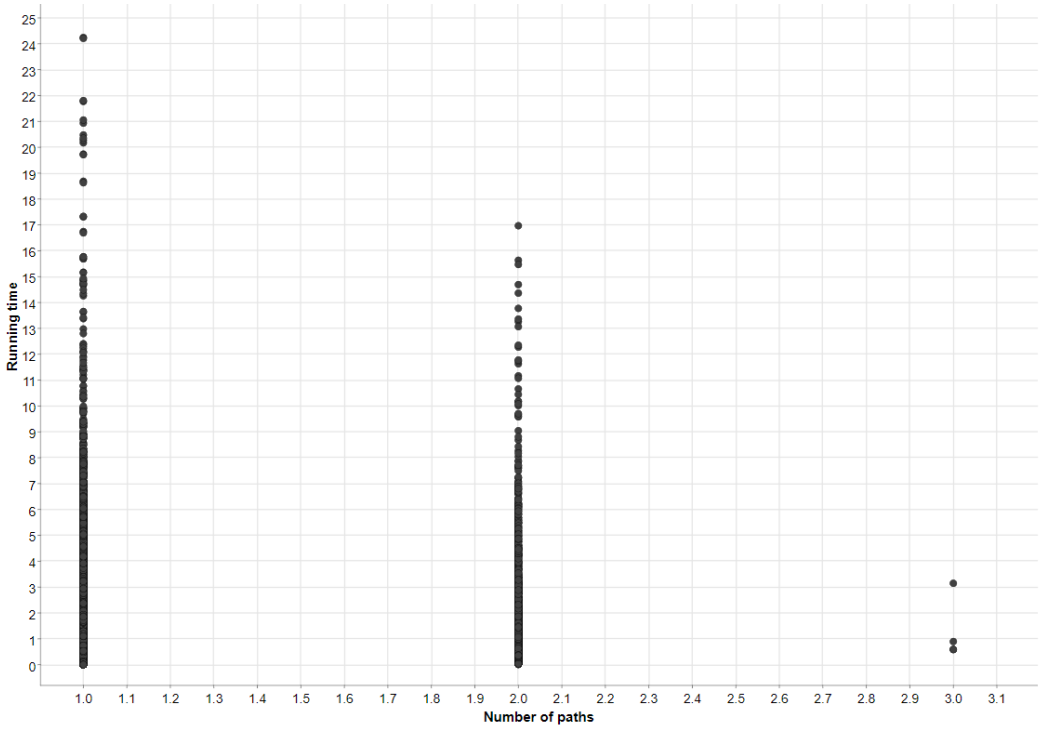


Figure 31 Evaluation of the number of found alternative paths in relation to the running time with parameters $\gamma = 0, skip\_feasible = 0$ and with CH, bidirectional Dijkstra, and parallelization. The y-axis represents the running time in seconds and the x-axis the number of alternative paths.

An unsurprising finding is shown in Figure 32: a positive correlation between the number of feasibility checks and the overall running time of the algorithm with optimal parameter setting for RR. This implies that performance improvements are well-reflected in an improved running time.

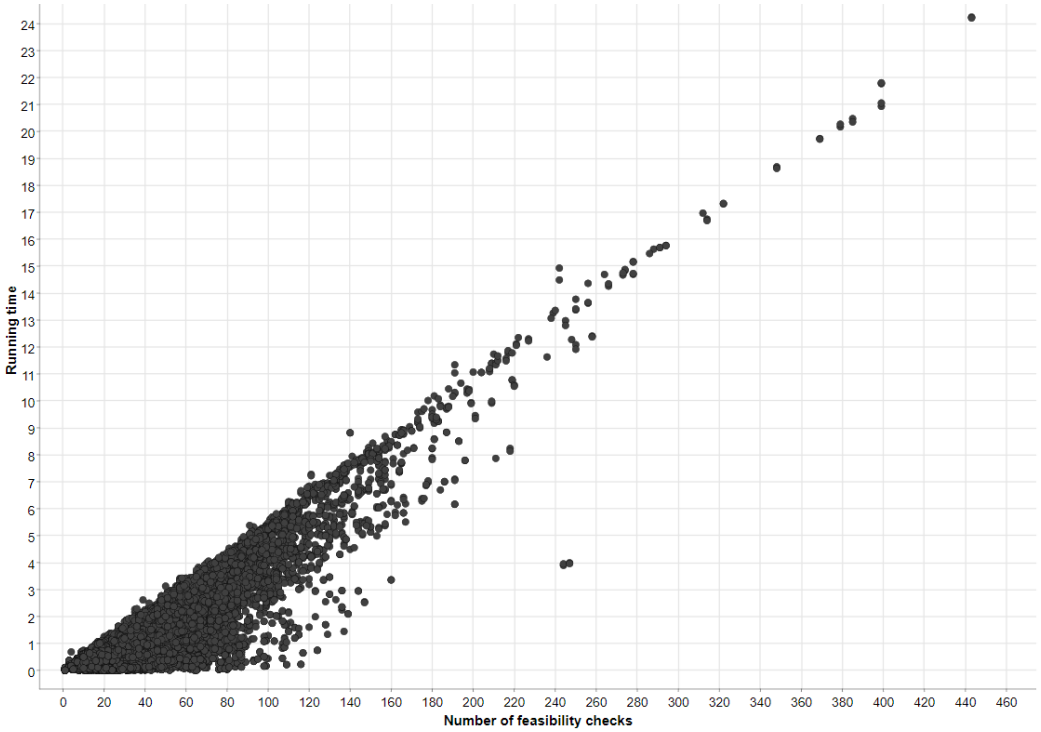


Figure 32 Evaluation of the number of feasibility checks in relation to the running time with parameters $\gamma = 0$, $skip\_feasible = 0$ and with CH, bidirectional Dijkstra, and parallelization. The y-axis represents the running time in seconds and the x-axis the number of feasibility checks.

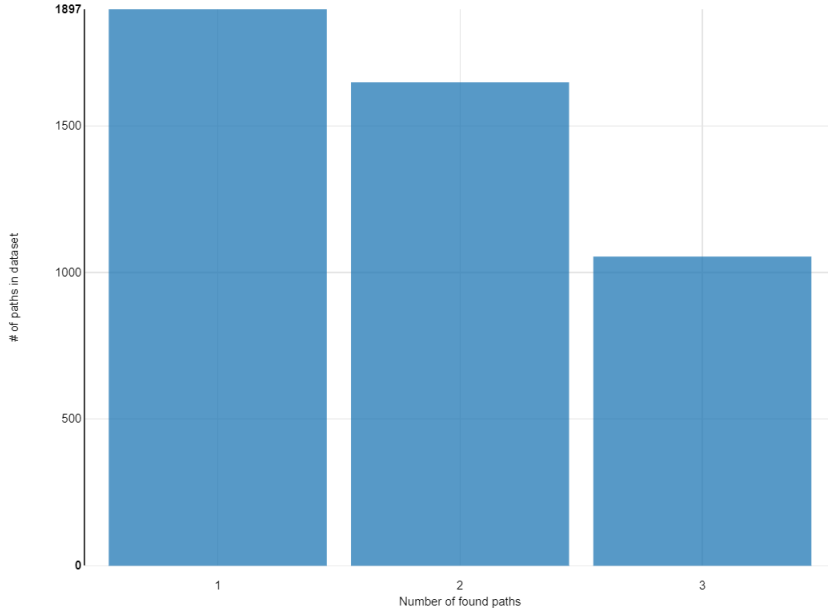
This completes the running time discussion. The sensible parameter settings and impacts of input parameters and improving techniques are examined concerning the running time of the algorithms.

8.3 Quality Discussion

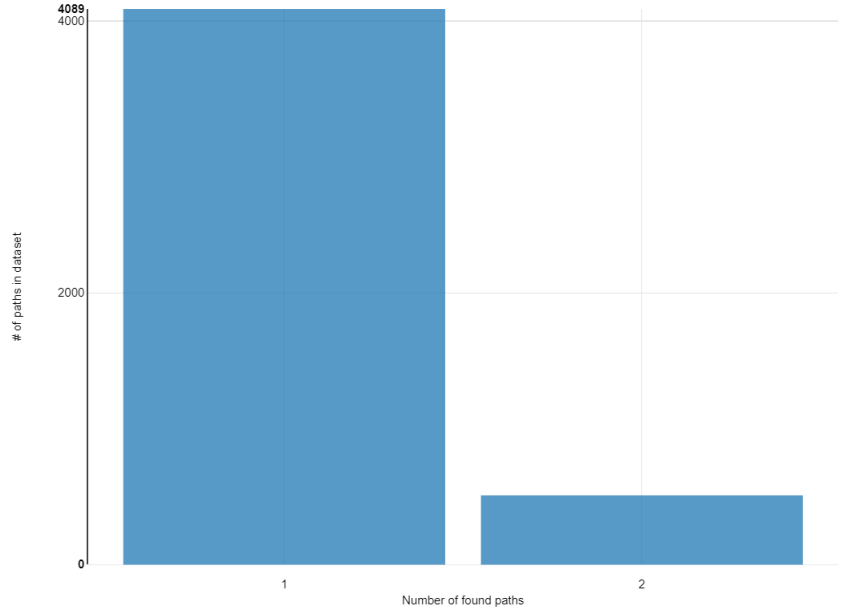
The quality of the results of the Penalty algorithm and its variants is an important measure for RR. First, parameter γ highly influences the number of paths that are found. This is shown in Figure 33a and Figure 33b, where for $\gamma = 0$ (for RR) the number of paths is much lower than for $\gamma = 0.2$.

As $\gamma = 0$ is fixed for RR, for the remaining parameters it has to be examined whether the number of paths differs for different parameter settings. Thus, Figure 34a and Figure 34b indicate that lower values for parameter α could lead to more paths, as incoming edges of nodes laying on the current found path are not penalized that hard. Additionally, for $\alpha = 0.3$ the number of results with two paths is higher than for $\alpha = 0.5$.

Furthermore, higher values for ϵ result in a higher chance to find more paths as ϵ indicates the maximum length of an alternative path. This is shown in Figure 35.

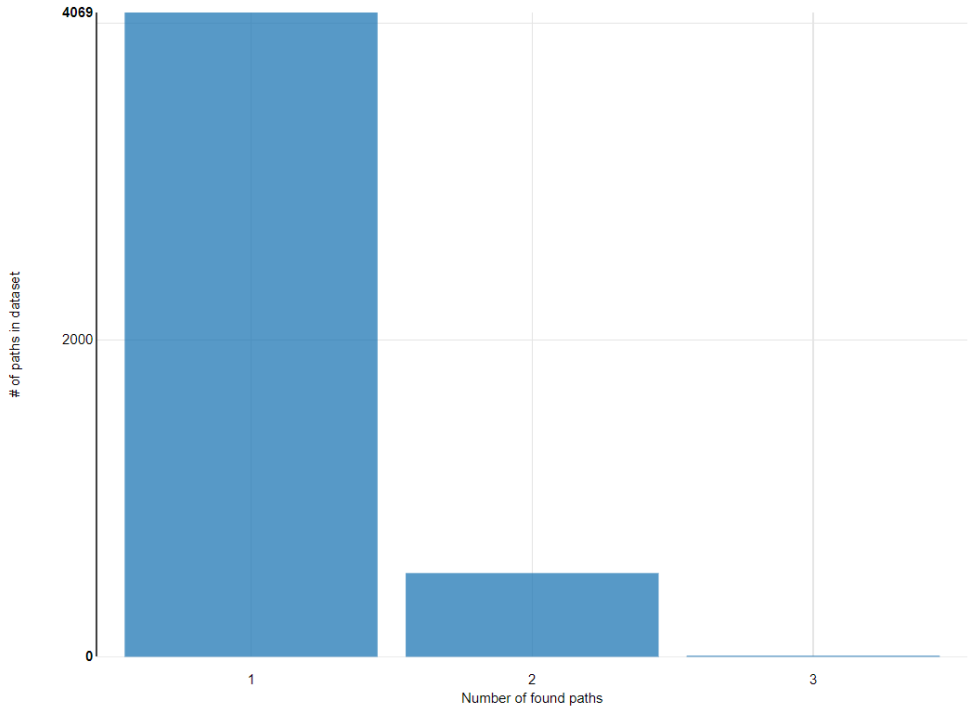


(a) Evaluation of number of found paths for $\gamma = 0.2$.

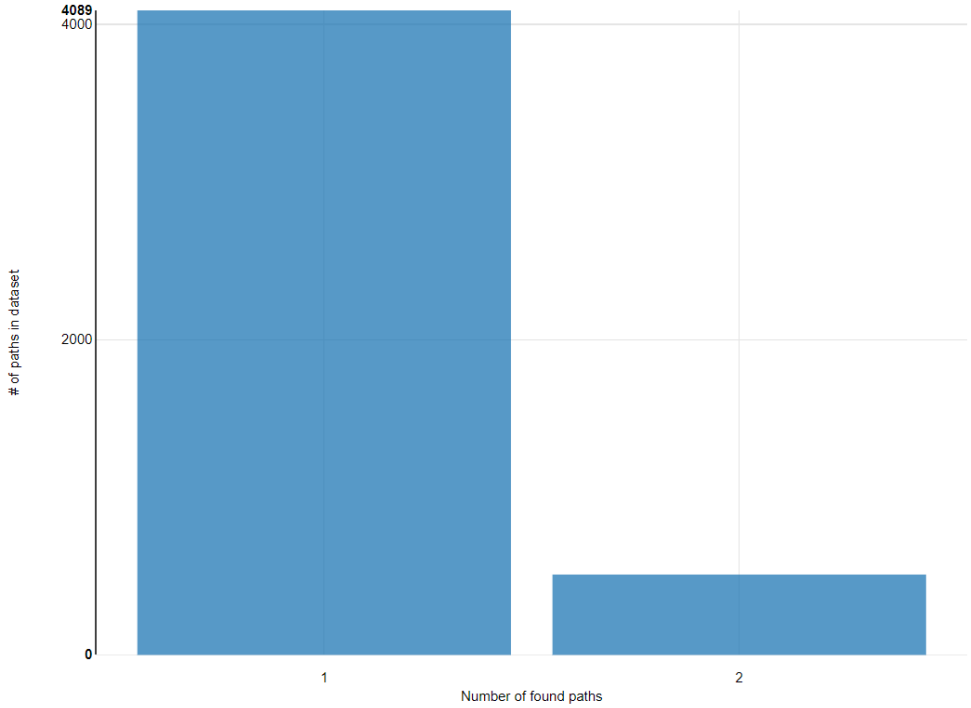


(b) Evaluation of number of found paths for $\gamma = 0$.

Figure 33 Evaluation of parameter γ for number of found paths for $\alpha = \epsilon = 0.5, skip\_feasible = 0$ with CH, bidirectional Dijkstra, and parallelization.

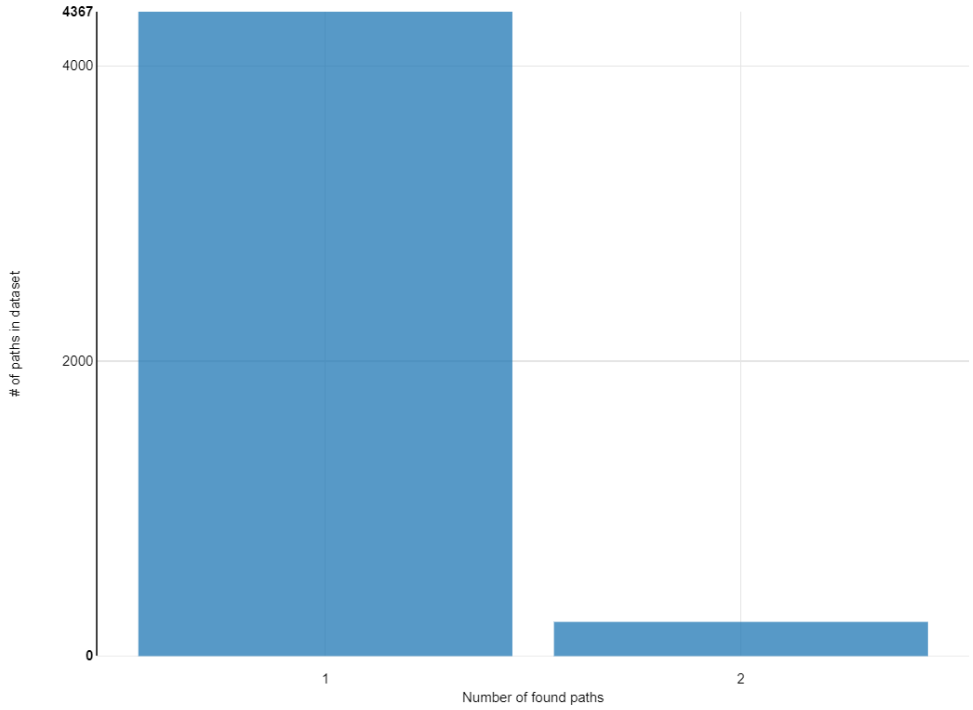


(a) Evaluation of number of found paths for $\alpha = 0.3$.

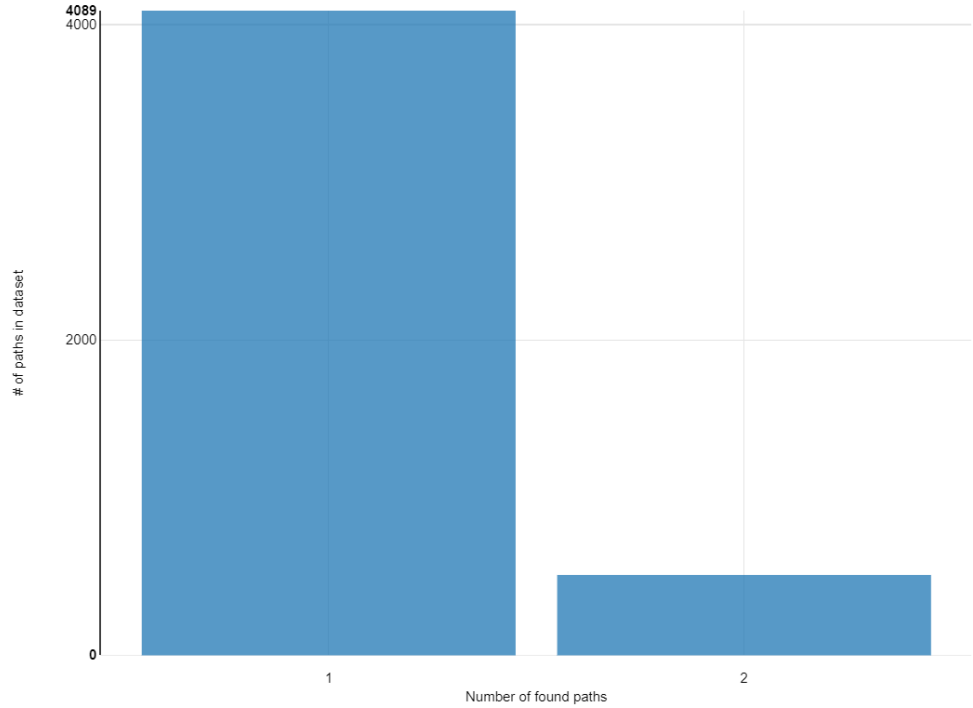


(b) Evaluation of number of found paths for $\alpha = 0.5$.

Figure 34 Evaluation of parameter α for number of found paths for $\epsilon = 0.5, \gamma = 0, skip\_feasible = 0$ with CH, bidirectional Dijkstra, and parallelization.



(a) Evaluation of number of found paths for $\epsilon = 0.25$.



(b) Evaluation of number of found paths for $\epsilon = 0.5$.

Figure 35 Evaluation of parameter ϵ for number of found paths for $\alpha = 0.5, \gamma = 0, skip\_feasible = 0$ with CH, bidirectional Dijkstra, and parallelization.

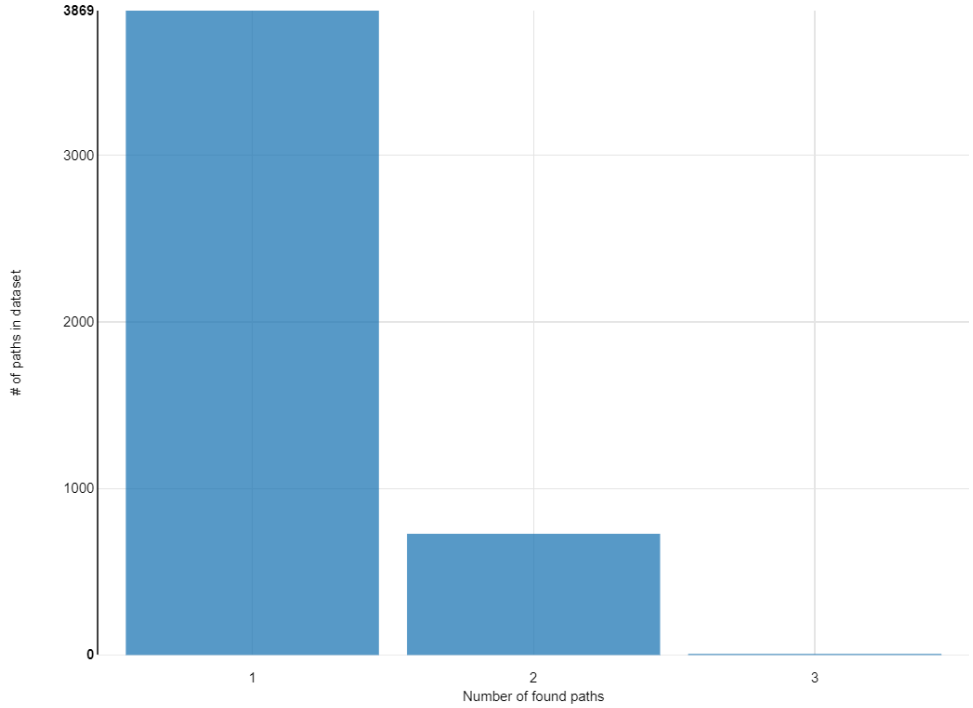


Figure 36 Evaluation of number of found paths for $\epsilon = 0.75$ and $\alpha = 0.5, \gamma = 0, skip\_feasible = 0$ with CH, bidirectional Dijkstra, and parallelization..

In addition, the first row in Table 5 examines the mean of length differences if three paths are found. These values suggest that the shortest path and the first alternative should be preferred as a solution for RR over the first and second alternative path. Besides, Table 5 gives an intuition about the length of the alternatives that are found without any restrictions of the number of paths but limited γ (second row). It can be seen that the first alternative is roughly a third longer than the shortest path. This seems to be a quite good result for RR.

γ	**Number of paths (input parameter)**	**Mean of length difference shortest and first alternative (meters)**	**Mean of length difference first alternative and second alternative (meters)**	**Average shortest path length (meters)**
0	3	6694.75	7345.75	27307 (only two observations)
0	$\{1, 2, 3\}$	8417.80	7345.75	26927.59
$\{0, 0.2\}$	$\{1, 2, 3\}$	3234.28	3399.36	31847.32

Table 5 Evaluation of length differences for $skip\_feasible = 0$ with CH, bidirectional Dijkstra, and parallelization.

If γ is not restricted to equal zero ($\gamma \in \{0, 0.2\}$), the average length differences are very low compared to the average shortest path lengths as presented in the third row in Table 5. Nevertheless, these results might not be valid solutions for RR. However, this finding could be another starting point for further research.

Another hypothesis to check is the following: the length of the shortest path is correlated to the length of the first alternative path. This would imply that the algorithm would not find good solutions for RR for longer routes. Fortunately, Figure 37 disproves the hypothesis as a uniform distribution can be seen.

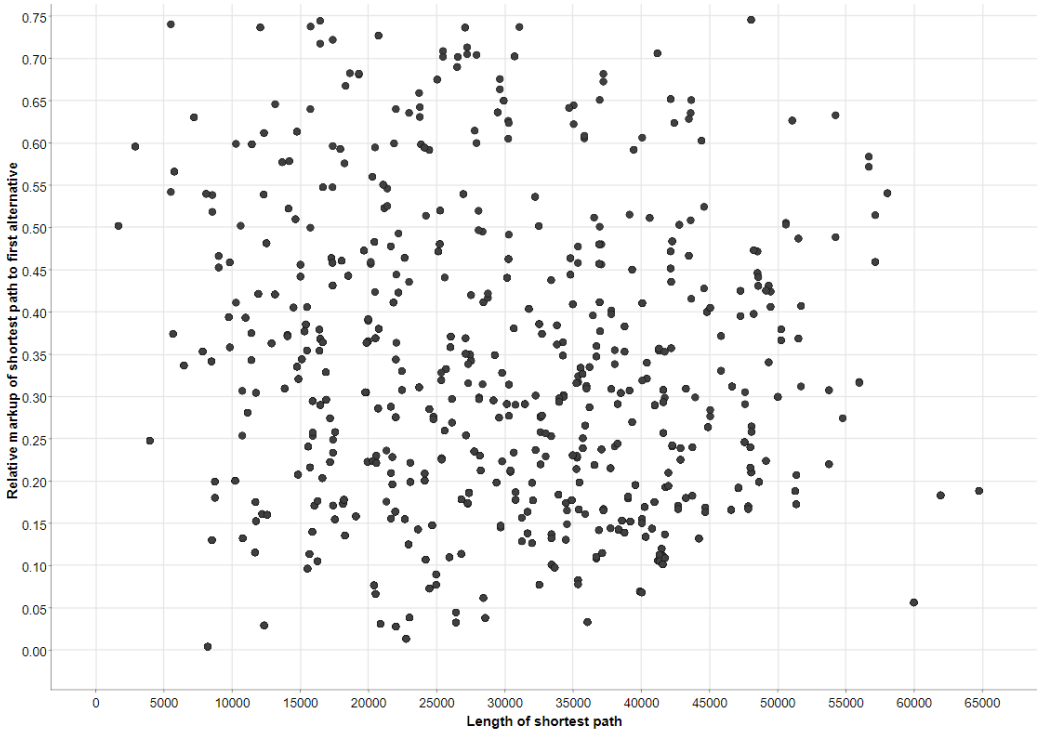
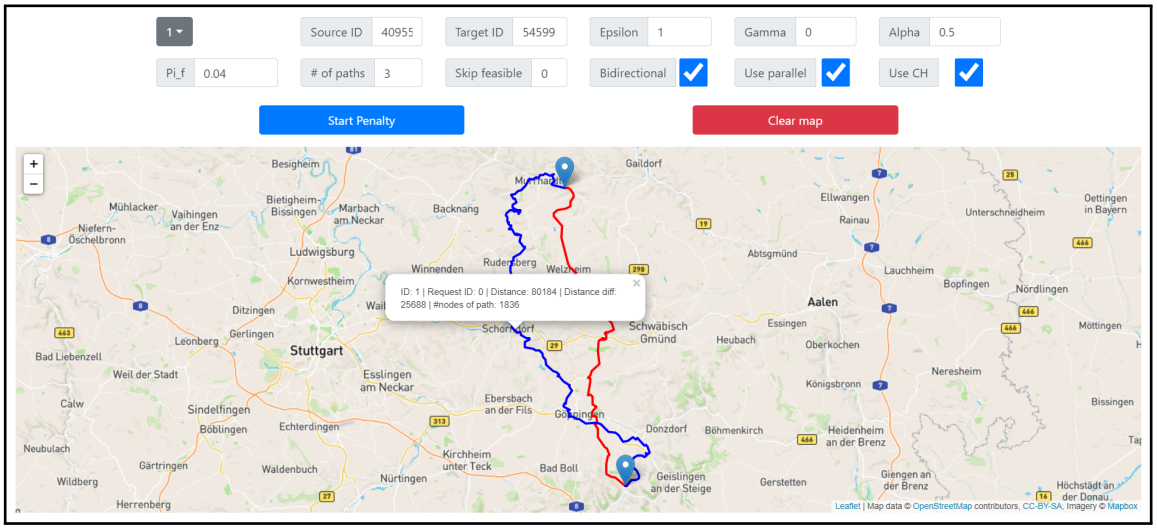


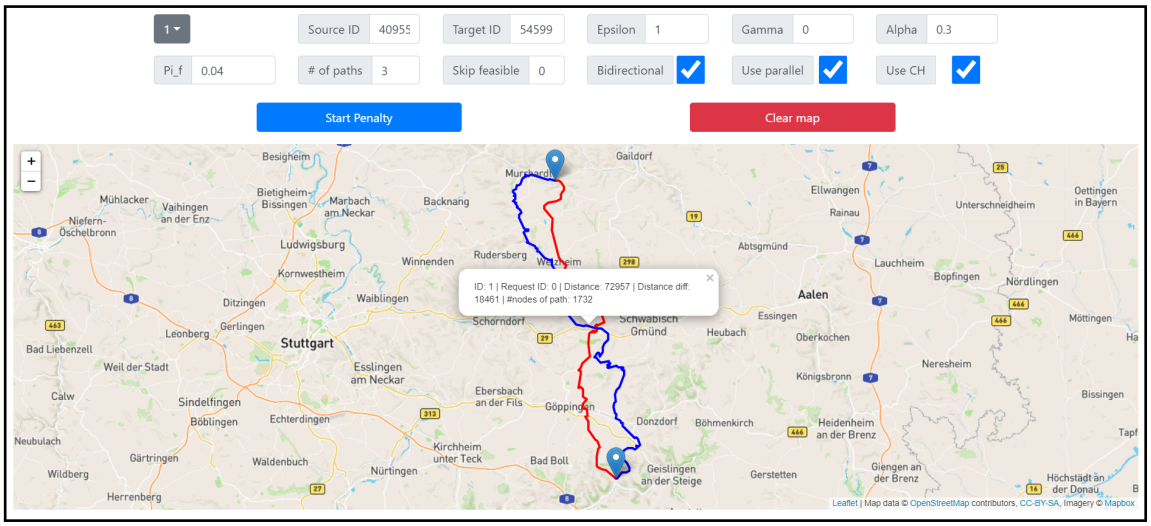
Figure 37 Evaluation of the length of the shortest path in relation to the relative markup of the shortest path to the first alternative with parameters $\gamma = 0, skip\_feasible = 0$ and with CH, bidirectional Dijkstra and parallelization. The y-axis represents the relative markup and the x-axis the length of the shortest path in meters.

8.3.1 Visualizations

For the following visualizations of solutions to RR, the red path is the shortest path and the blue paths are alternative paths found by the Penalty algorithm for RR. As mentioned before, different settings for input parameter α can lead to different results. Figure 38 clearly demonstrates this case. For $\alpha = 0.5$ in Figure 38a, the incoming edges of nodes laying on the the shortest path are penalized harder. This leads to a worse solution for RR compared to Figure 38b.



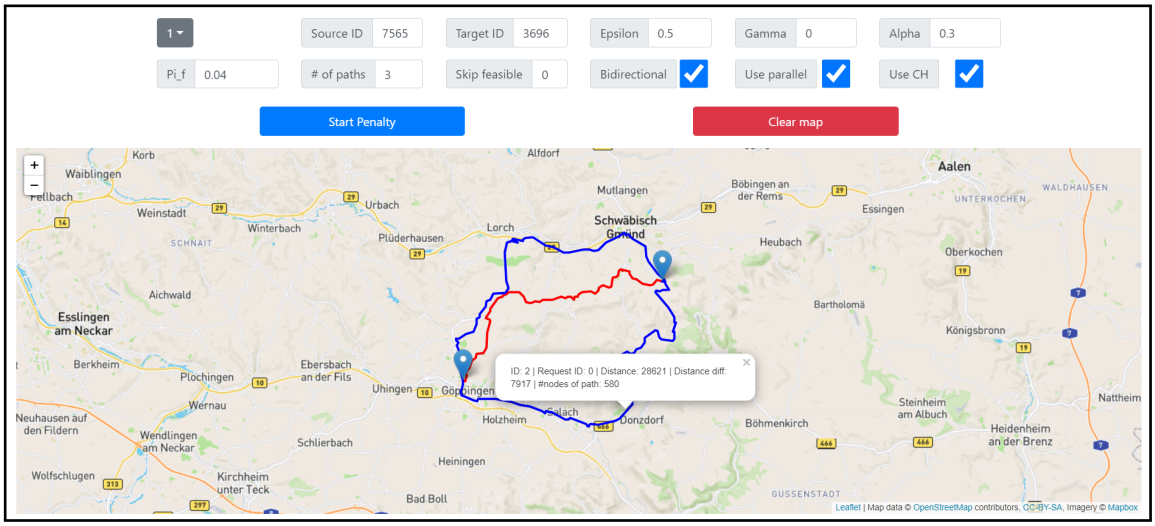
(a) $\alpha = 0.5$, further setting: source=40955, target=54599, $\epsilon = 1$, $\gamma = 0$, $\pi_f = 0.04$, number of paths $= 3$, $skip\_feasible = 0$.



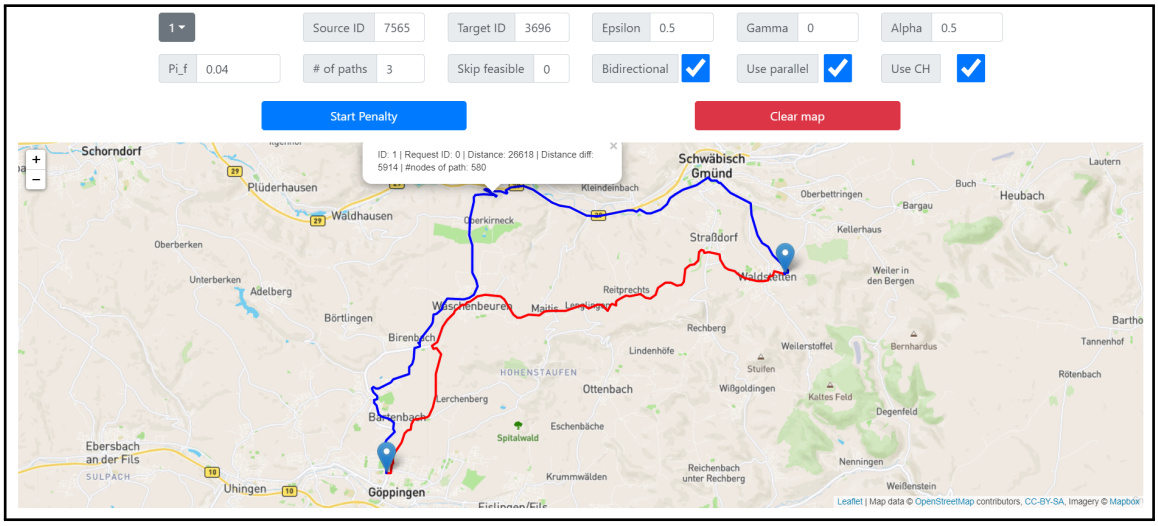
(b) $\alpha = 0.3$, further setting: source=40955, target=54599, $\epsilon = 1$, $\gamma = 0$, $\pi_f = 0.04$, number of paths $= 3$, $skip\_feasible = 0$.

Figure 38 Different values for α lead to different results.

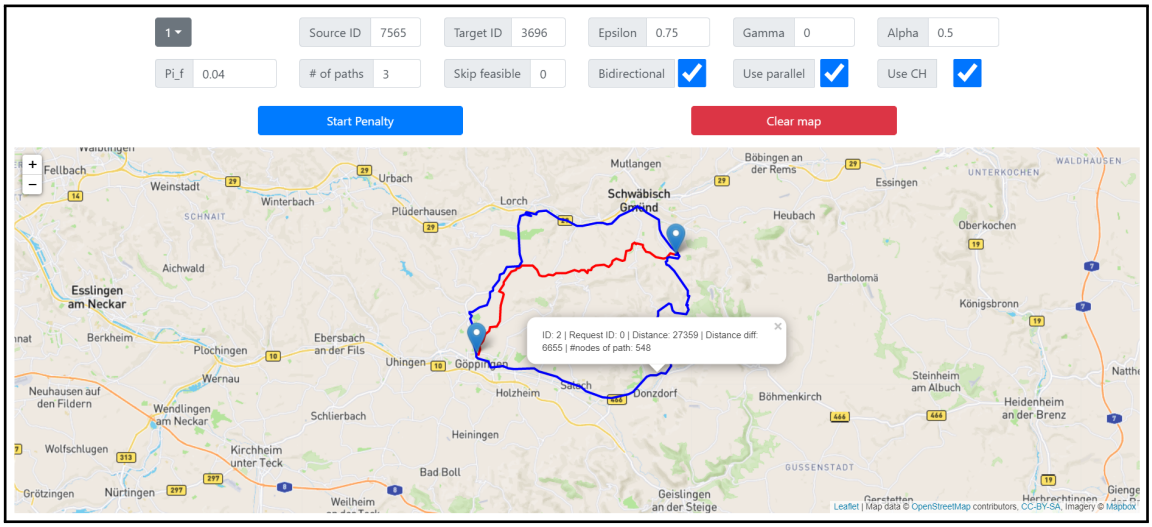
Furthermore, Figure 39 demonstrates the difference of results for varying parameter settings. First, Figure 39a and Figure 39c do not show the same results. Concerning RR, the best solution would be to take the shortest path and the first alternative shown in *Figure 39b*. Besides, Figure 39a has a good parameter setting concerning running time, as the discussion above showed that variations of α do not impact the running time whereas the variation of ϵ does. For Figure 39c, ϵ has to be set pretty high to get two alternative routes. If ϵ is not varied as in Figure 39b, the second alternative will not be found.



(a) $\alpha = 0.3, \epsilon = 0.5$



(b) $\alpha = 0.5, \epsilon = 0.5$



(c) $\alpha = 0.5, \epsilon = 0.75$

Figure 39 Different parameter settings (α and ϵ) lead to different outcomes. Further setting: source=7565, target=3696, $\gamma = 0$, $\pi_f = 0.04$, number of paths $= 3$, $skip\_feasible = 0$.

8.4 Summary of Findings

One of the most interesting and surprising outcome of the running time and quality discussion is the setting of input parameter α. For $\alpha = 0.5$ or $\alpha = 0.3$, no differences in the running time can be detected but the route quality has the potential to become better for lower values of α. As the penalization of incoming edges of nodes laying on the shortest path seems to have no impact on the running time, paths near the shortest path are not penalized that hard.
Generally, the running time is best if all possible improvements (CH, bidirectional Dijkstra, and parallelization) are applied. The choice of ϵ highly influences the running time and also the number of alternative paths that can be found. Furthermore, γ has an impact on the running time and result of the algorithm too. Nevertheless, for RR it is fixed to $\gamma = 0$ but can be a starting point for further improvements including an extensive selection process in the alternative graph.
The running time and quality discussion shows that the adapted Penalty approach is able to solve the RR problem with reasonably good results. Furthermore, the absolute running times demonstrate the applicability of this algorithm for real-world use cases. As there are many starting points for improvements (also concerning the programming language and CH), the duration for RR requests can be improved. Thus, a real-world applicability is possible concerning the running time. The penalization of edges seems to be a working approach to approach the RR problem using a heuristic algorithm.

CHAPTER 9
Conclusion and Future Work

Starting from the definition of RR as a fairly new special case of Alternative Route Planning, the time complexity is shown to be weakly NP-hard. Thus, an efficient exact algorithm does not exist. Therefore, it is evaluated whether the Penalty algorithm proposed by Kobitzsch et al. suits the RR problem. The extensive theoretical and practical suitability study indicates that a variant of the Penalty approach could lead to quite good results concerning running time and quality of results.
Next, further improvements are included in the Penalty algorithm, namely Contraction Hierarchies, the bidirectional variant of the Dijkstra algorithm, and parallelizaton. Applying these techniques together with different parameter settings results in better running times and multiple algorithmic approaches for RR based on the Penalty algorithm. Concerning the quality of routes, the input parameters of the algorithm (namely α, γ, and ϵ) can change the outcomes. π_f determines the factor for penalizing the edges of the current shortest path. This parameter does not change the outcoming alternative routes but can influence the running time. For this work, this parameter is set as proposed by Kobitzsch et al. and not further evaluated.
The choice of ϵ highly depends on the use case. If the definition of RR is taken as it is, ϵ should be unbounded. As this is not possible for the Penalty algorithm in its current form, the value of ϵ is considered to be use case specific. Nevertheless, the evaluation has shown that a value of $0.25 \leq \epsilon \leq 0.5$ could be reasonable. For the RR problem, $\gamma = 0$ is fixed per definition. The evaluation in the previous chapter indicates a huge impact of γ to the running time and quality. Thus, further research could use it as a starting point. Furthermore, α turns out to be a critical parameter for finding a solution for RR. A value of $\alpha = 0.3$ provides better results compared to $\alpha = 0.5$. Additionally, the lower value of α does not really influence the running time. Therefore, $\alpha \leq 0.3$ seems to be a good choice for RR. This completes this work and answers the research questions.

Since RR is a new research topic, there exists further possible future work. If one sticks with the penalty approach, additional adaptions of the input parameter can be examined. If α is assigned a lower value, incoming edges of nodes laying on the current found path are not penalized that hard. This means, that possible more routes could be found that are near the (shortest) path and therefore a better solution for RR. But the stopping criterion might be reached later, as the convergence is possibly slower. For future work, low values $\alpha \leq 0.3$ can be examined whether they lead to better results. Especially in combination with parameter ϵ, both parameters highly influence the quality of the approximation of RR.

Furthermore, if it is possible to define the point where two found routes are a good solution for RR, it might be a good choice to stop the algorithm earlier. In addition, further engineering in the shortest path algorithms like CH and another programming language than Java can improve the running time heavily to enable real-world applications of RR.
As the RR problem requires multiple optimizations, it may be necessary to e.g. prefer Condition (3) over Condition (1). Then it would be sensible to e.g. add a small detour for the shortest path so the length difference is minimized and the lengths of the paths do not exceed some given limit.
Finally, the algorithmic approaches of Chondrogiannis et al. [6] may also be adaptable for the RR problem.

Bibliography

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Alternative routes in road networks. *ACM Journal of Experimental Algorithmics*, 18(1), 2013.

[2] Y. Ando, O. Masutani, H. Sasaki, and H. Iwasaki. Pheromone Model : Application to Traffic. *Engineering Self-Organising Systems*, pages 182–196, 2006.

[3] R. Bader, J. Dees, R. Geisberger, and P. Sanders. Alternative Route Graphs in Road Networks. In *International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems (TAPAS'11)*, volume 6595, pages 21–32. 2011.

[4] F. Barth and S. Funke. Alternative routes for next generation traffic shaping. *IWCTS 2019 - Proceedings of the 12th International Workshop on Computational Transportation Science*, 2019.

[5] A. W. Brander and M. C. Sinclair. A Comparative Study of k-Shortest Path Algorithms. *Performance Engineering of Computer and Telecommunications Systems*, pages 370–379, 1996.

[6] T. Chondrogiannis, P. Bouros, J. Gamper, U. Leser, and D. B. Blumenthal. Finding k-shortest paths with limited overlap. *VLDB Journal*, 29(5):1023–1047, 2020.

[7] D. Delling, M. Holzer, K. Müller, F. Schulz, and D. Wagner. High-performance multi-level routing. 2:73–91, 2009.

[8] D. Delling, K. Müller, and D. Wagner. High-Performance Multi-Level Graphs. 2006.

[9] D. Delling and R. F. Werneck. Faster customization of road networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7933 LNCS:30–42, 2013.

[10] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[11] L. Feng, Z. Lv, G. Guo, and H. Song. Pheromone based alternative route planning. *Digital Communications and Networks*, 2(3):151–158, 2016.

[12] T. Fiser. *Integrated Route and Charging Planning for Electric Vehicles*, volume 5. 2017.

[13] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In C. C. McGeoch, editor, *Experimental Algorithms*, pages 319–333, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[14] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, 2005.

[15] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. *International Conference on Information and Knowledge Management, Proceedings*, pages 499–508, 2010.

[16] Z. He, K. Chen, and X. Chen. A collaborative method for route discovery using taxi drivers' experience and preferences. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2505–2514, 2018.

[17] J. Hershberger, M. Maxel, and S. Suri. Finding the k shortest simple paths: A new algorithm and its implementation. *ACM Transactions on Algorithms*, 3(4), 2007.

[18] M. Kobitzsch, M. Radermacher, and D. Schieferdecker. Evolution and evaluation of the penalty method for alternative graphs. *OpenAccess Series in Informatics*, 33:94–107, 2013.

[19] M. Kobitzsch, S. Samaranayake, and D. Schieferdecker. Pruning Techniques for the Stochastic on-time Arrival Problem - An Experimental Study. pages 1–13, 2014.

[20] L. Li, M. A. Cheema, M. E. Ali, H. Lu, and D. Taniar. Continuously monitoring alternative shortest paths on road networks. *Proceedings of the VLDB Endowment*, 13(11):2243–2255, 2020.

[21] A. Paraskevopoulos and C. Zaroliagis. Improved Alternative Route Planning. 2013.

[22] G. Scano, M. J. Huguet, and S. U. Ngueveu. A k-shortest paths based algorithm for multimodal time-dependent networks to compute alternative routes. 2015.

[23] P. Stroobant, P. Audenaert, D. Colle, and M. Pickavet. Generating constrained length personalized bicycle tours. *4or*, 16(4):411–439, 2018.

D E C L A R A T I O N :

I herby affirm that I have independently written the attached Bachelor's/Master's thesis on the topic:

Improved Algorithms for Rendezvous Routing

based on the Penalty Approach

and have not used any other aids or sources other than those I have indicated.

For parts that use the wording or meaning coming from other works (including the Internet and other electronic text and data collections), I have identified them in each case by reference to source or the secondary literature.

Furthermore, I hereby affirm that the above mentioned work has not been otherwise submitted as a thesis for a ~~Bachelor's/~~Master's examination *). I further understand the pending completion of the review process I must keep the materials available that can prove this work was written independently.

After the examination process has been completed, the work will be submitted to the Library of the University of Konstanz and catalogued. It will thus be available to the public through viewing and lending. The described data given, such as author, title, etc. are publicly available and may be used by third parties (for example, search engine providers or database operators).

As author of the respective work, I **consent /** ~~**do not consent**~~ to this procedure ***)**.

A current confirmation of my enrolment is attached.

(Signature)

Konstanz, 08.10.2021

(Place, date)

*) Please delete as appropriate