Karlsruhe Institute of Technology

Finding Attractive Routes for Bicycles and Pedelecs

Diploma Thesis of

Philipp Glaser

at the Department of Informatics
Institute of Theoretical Computer Science

Reviewer:	Prof. Dr. Dorothea Wagner
Second reviewer:	Prof. Dr. Peter Sanders
Advisor:	Moritz Baum, M.Sc.
Second advisor:	Dr. Moritz Kobitzsch

15. June 2015 – 15. December 2015

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, December 15, 2015

. . . . . .

(Philipp Glaser)

Abstract

Cyclists often have multiple criteria in mind when planning a route. They do not only consider the length of the route, but also the (perceived) safety or the environment of the route. Cycleways through park are often more attractive than heavily used roads.

But not every cyclist weights these criteria the same way and even for one cyclist the weighting may depend on his current situation. In a hurry travel time might be the most important aspect, but even then me might make consider how much he will compromise his safety. In particular he will likely accept a slightly longer travel time, if the alternate route is perceived as much safer. Another often regarded criterion is the elevation difference of the route.

We consider this five criteria in our approach. As basis for our algorithm we use A* search. To get suitable potentials we use a reverse Dijkstra search starting a the target node for each criterion.

The big drawback of performing a multi-criteria search is that the number of routes may grow very quickly, which leads to a long query time as well as to a route set that does not help the user much in his rotue planning. Therefore we filter the routes during the computation to get a small amount of routes to help the user. To do this we use a variation of Pareto-dominance, in which a route are discarded if they would be dominated assuming it would be worse by an ε. Using this heuristic the number of found routes can be reduced by an significant amount and therefore the running time is reduced accordingly. AS a minor drawback the set of routes found with this approach depends on the order in which the routes are found.

Therefore we tested and compared several difference order, which we used for our search. Out of the evaluated orderings the Euclidean distance to the potential of the source node resulted in the fewest routes, if we keep the same ε. Compared to the lexicographic order this resulted in 57 % fewer routes on average on a small graph. On a big graph it even reduced the number of routes by 90 %.

Zusammenfassung

Radfahrer berücksichtigen bei der Planung ihrer Route meistens mehrere Kriterien. Oft spielt nicht nur dir Kürze der Strecke eine Rolle, sondern auch die (empfundene) Sicherheit und die nähere Umgebung der Strecke. Radwege durch Parks sind in der Stadt meistens attraktiver als eine Route die vielbefahrene Straßen nutzt.

Häufig ist die Gewichtung dieser Kriterien jedoch nicht für jeden Radfahrer gleich und kann auch für einen einzelnen Radfahrer von der momentanen Situation abhängen. Unter Zeitdruck ist vermutlich die Reisezeit das wichtigste Kriterium, obwohl auch dann noch Mindestanforderungen bezüglich der Sicherheit der Strecke bestehen. Insbesondere wird man kleine Verzögerungen in Kauf nehmen, wenn die Alternative als viel sicherer empfunden wird. Ein weiteres betrachtetes Kriterium waren Anstiege entlange der Route.

Diese fünf Kriterien betrachte wir nun in unserem Ansatz. Als Basis für unseren Algorithmus nutzen wir eine A* Suche. Um geeignete Potentiale zu erhalten, führen wir für jedes Kriterium eine Rückwärts-Dijkstra-Suche vom Zielknoten aus durch.

Allerdings kann die Menge an Pareto-optimalen Routen schnell sehr groß werden, was sowohl zu einer schlechten Laufzeit der Routenberechnung führt als auch dem Anwender nicht viel weiter hilft bei der Routenwahl. Daher filtern wir die Routen bereits während der Berechnung und bieten so eine kleine Menge an Alternativ-Routen, um den Anwender bei der Routenwahl zu unterstützen. Dazu nutzen eine Variation der Pareto-Dominanz bei der Routen bereits verworfen werden, wenn sie unter der Annahme, die Route wäre um ein ε schlechter, von den bereits gefundenen Route dominiert wird. Dadurch lässt sich die Anzahl der gefundenen Routen signifikant reduzieren und damit auch die Laufzeit entsprechend verkürzen. Allerdings hängt dadurch die Menge der gefundenen Routen davon ab in welchen Reihenfolge die Routen gesucht werden. Daher haben wir auch verschiedene Reihenfolgen getestet und experimentell verglichen. Von der erprobten Reihenfolgen hat sich der euklidische Abstand der Summe aus Routenlänge und Potential des aktuell besuchten Knotens zum Potential des Startknotens als am besten herausgestellt. Mit einem unveränderten ε gegenüber der lexikographischen Reihenfolge lies sich die Anzahl der Routen auf einem kleinen Graphen im Schnitt, um 57 % reduzieren. Auf einem großen Graphen sank die Anzahl sogar um 90 %.

Contents

List of Figures

List of Tables

1 Introduction

Using cycling as transport mode has many benefits for the environment and public health (see also in the WHO Report on "Transport, environment and health" [11]) and promoting cycling as become generally accepted in Germany[1]. On the other hand many people refrain from cycling due to safety concerns [42], so providing a safe alternate route should be an important aspect of route planning for bicyclists.

Choosing a route for a bicycle trip is therefore not an easy task. Often the route choice does not only depend the length of the route or the travel time, but also on several other criteria like exhaustion or the sightliness of the passed scenery. Not every bicyclist might take all of these criteria into account. Some criteria might even depend on the reason for the bicycle trip or the current situation the bicyclist is in.

To find an attractive route for a bicyclist, we therefore need to consider several criteria, which influence the decision of the bicyclist on which route to choose. While it may be possible to identify several criteria involved in the decision, which route to choose, it is bound to be subjective to weight these criteria and find a suitable route for every user. Some people might just want the shortest route, while others prefer the fastest route or a safe route, that should not become to long. This can vary even with the same user, e.g. if he is in a hurry, he might prefer the fastest route, while otherwise he would rather prefer a route with fewer or slower cars.

1.1 Our Contribution

Therefore, we consider multiple criteria in our search to find high quality bicycle routes. The criteria include travel time, distance, energy usage, safety, and the surrounding area. As multi-criteria search is known to find too many routes to provide useful assistance in planning a route and it is also NP-Complete (see [15], we filter the routes during computation to reduce the run time and amount of routes in the result. As part of this work we evaluate different filter techniques to reduce the number routes computed and different priority functions and their impact on the resulting route set and the running time.

[1] www.nationaler-radverkehrsplan.de

1.2 Related Work

In this section we introduce the algorithms that are used in our approach to find sets of routes. Later we present some studies about how bicyclist choose their routes and how these criteria are related to features of the map.

1.2.1 Basic Algorithms

When searching the shortest route, the most common approach is to map the road network to a digraph. Then Dijkstra's algorithm [9] or a similar algorithm is used in this digraph to find the route. As Dijkstra's algorithm visits all nodes from the source in the order of their distance from the source, it can take too long for large graphs. Therefore many improvements have been made to find a single route to a target node in a shorter time period, that either change the order in which these nodes are visited or ignore certain nodes, which are known to be not involved in the shortest route. An overview on current route planning algorithms can be found in [1].

An early improvement on Dijkstra's algorithm, which changes the order in which the nodes are visited, is the A* Search algorithm [19]. It takes an estimation of the distance of each node to the target node into account and visits nodes earlier for which the length of the current path plus this estimated remaining distance is shorter.

1.2.2 Pareto-optimal Routes

Several ways to deal with the multi-criteria problem have already been proposed in the past. Most approaches extend Dijkstra's algorithm to deal with Pareto sets, see [1].

The Multi-Label-Setting [18, 26] algorithm uses a bag of non-dominated labels at each node and keeps the labels with the tentative distances in the priority queue. If the next label in the priority queue is not dominated by the labels of the bag at the corresponding node, the label is added to the node. We need to guarantee, that with the chosen priority function a label, that is extracted at a later step, cannot dominate a label extracted earlier and therefore is already in the bag.

After the label is added to the bag the incident outgoing arcs of the node are scanned and the resulting tentative distances are added to the priority queue, if they are not already dominated at the end node. This check is not necessary to ensure correctness of the algorithm, but leads to a significant speedup.

The Multi-Label-Correcting [5] algorithm expands all labels, that are not dominated, if the node is extracted from the priority queue and adds the labels to the node as well as the into the priority queue. The bags of each node may contain dominated labels, which need to be removed, if a route found later dominates it.

We choose the Multi-Label-Setting approach as it keeps the required data structures simple as we do not need to delete entries from our bags.

Qing Song et al. [33] evaluate multi-criteria urban bicycle routing. They used a multi-label correcting approach, where they expand all non-dominated labels at each visited node

and they also add all tentative distances to the priority queue, to generate all Pareto routes with criteria based on former research: travel time, comfort and flatness. Afterwards they filter the routes, by clustering them and selecting the route in each cluster with the highest difference to other clusters. To compute the travel time they use the road surface, obstacles and uphill/downhill segments. The comfort criterion uses basically the same basic metrics with different weights. In addition infrastructure dedicated for bicycles is favored and ways and crossings shared with motorists are penalized. The flatness criterion took into account the ascend or descend of route segments. The authors not optimize query time in this paper and used a rather small network consisting only of about 10000 nodes and 20000 edges.

In a follow-up conference paper Hrncir and Jakob [22] introduce quietness as additional criterion. They compute four routes using fixed different linear combinations of the aforementioned criteria.

At the ATMOS 2015 conference they presented an additional paper [23], in which they makes use of several heuristics to reduce the amount of computed routes. Among others heuristics they use an pruning mechanism they called Ellipse Pruning, which should likely reduce the number of searched edges in a way similar to the A* Search we use, but is heuristic.

Another recent study by Machuca and Mandow [25] used A* Search to speed up multi-criteria search in road maps. They used an heuristic proposed previously by Tung and Chew [39].

In Pareto Paths with SHARC Delling and Wagner[6] augment the SHARC algorithm to deal with multi-criteria scenarios. They conclude that this multi-criteria variant of SHARC can lead to similar speedups compared to a Multi-Label-Correcting as in the single criteria case.

Geisberger, Kobitzsch and Sanders [16] proposed an algorithm based on Contraction Hierarchies [17] that answers shortest path query with two metrics, where the trade-off between the two metrics can be given at query time.

Funke and Storandt [14] proposed a way to construct contraction hierarchies for multi-criteria shortest path problems, where the minimum cost are a conic combination, but as the number of shortcuts required increases quickly with additional criteria, they kept the contraction incomplete even for three criteria.

In Speed-Consumption Trade-off for Electric Vehicle Route Planning [2], the Baum et al. study the problem of computing routes for electric vehicles, which optimize both energy consumption and speed. They also account for the opportunity to not drive at the maximum allowed speed to save energy. As the complete Pareto set gets very large they also propose several techniques to filter out insignificant solutions on-line that closely resembles the exact Pareto set.

Storandt [35] also proposed a way to use contraction hierarchies to solve the constrained shortest path problem efficiently. In addition they also considered a similar scenario (bicycle routing), where they search the shortest path, while constraining the positive height difference and the other way round.

Raith and Ehrgott [30] also evaluated different solution strategies for biobjective shortest path problems.

Sedeño-Noda and Raith [31] also propose a method to compute all extreme supported non-dominated solutions in the biobjective shortest path problem. The extreme supported solutions are the subset of the Pareto set, that define the convex hull.

1.2.3 Criteria Selection

In Where Do Cyclist Ride [3] the authors evaluate influences on route choices in Portland (Oregon, US). According to the League of American Bicyclists[2] Portland is among the most bike friendly cities in the US. Their study was conducted, because cycling was often not included as a transport mode in regional travel demand models (in North America) and therefore not incorporated in transport planning. The authors research the attractiveness of different facilities and the influence of traffic control devices. This was done by GPS-tracking instead of bicyclist recalling used/preferred routes, which was the mode in most prior revealed preference[3] surveys. The authors also link the attributes of the routes with the additional/reduced distance the cyclists took compared to routes without that attribute. On average the chosen routes were about 12 % longer than the shortest network path. The authors differentiate between commute and non-commute trips. On commute trips the cyclists were a bit less incline to accept detours. They explain this and other differences between commute and non-commute trips with the increased knowledge the cyclists have about the commute trips.

In Motivators and Deterrents of Bicycling [42] the authors analyse several criteria that influence the decision whether to ride or not. In a (stated preference) survey of 1402 current and potential cyclists[4] in Metro Vancouver they evaluated 73 motivators and deterrents of cycling. The respondents were grouped by the frequency of cycling. (potential, yearly, monthly, weekly) The most important motivators were: “away from traffic noise and pollution”, “has a beautiful scenery”, “separated from traffic for the entire distance”, “is flat” and “cycling takes less time”. The most important deterrents were “icy and snowy”, “lot of car, bus and truck traffic”, “vehicles drive faster than 50 km/h”, “has glass and debris” and “risk of motorists who don’t know how to drive safely near bicycles”. As expected, the regular cyclists tended more to ride on average. On the other hand, there were significant differences for some criteria (like rain or distance between 10 and 20 km), which did not deter regular cyclists as much as potential cyclists. For other criteria (like “secure indoor storage”) the score was almost the same between the groups. Of course, only some criteria of these criteria can be retrieved from a static map, like “away from traffic”, “beautiful scenery” or “vehicles drive faster than 50km/h” (unfortunately, only the nominal speed limit can be retrieved from OpenStreetMap).

In an early GPS-based survey Menghini et al. [27] research route choices in Zurich. They evaluate a small number of criteria and they conclude that among their criteria, only

[2] www.bikeleague.org

[3] Revealed preferences as observed in the study in contrast to just stated preferences

[4] i.e., they had access to a bicycle and used it in the past year or considered to use in the future

the maximum gradient and the route length had an impact on the decision, which route the cyclists took. The analysed trips were also very short compared to those in the other studies.

Heinen et al. [20] give an overview on research on cycling as transport mode. Their focus is on the decision whether to cycle or not. They conclude that the decision to cycle is based on many factors like the natural environment, (perceived) safety, travel time and effort. Most of the other criteria they identify are not related to the built environment (weather, attitudes, socio-economic factors). They also conclude that travel time, cost and safety are more important for cycling, when compared to other modes of transport. For example, reduced cost might lead to choose a bicycle over public transport, while safety concerns might deter people from using a bicycle. On the other hand the choice between car and public transport is more often based on other criteria.

An unsafe route can lead to the decision to use another mode of transport instead. They also state that it is actually the perceived values, that lead to the this decision and this perception might differ between the examined groups, e.g. bicyclists might view the same route as safer than non-bicyclists.

1.3 Overview

We give a short overview over the following chapters. The remainder of this thesis is divided into five chapters followed by an appendix which contains implementation details. The next chapter contains terminology and fundamental algorithms used in the later chapters. Chapter 3 describes the criteria used in our approch and the motivation behind each. The algorithms to get the values for each criterion on each edge is also explained there. In Chapter 4 the algorithm used to find the best routes is introduced together with the considered and used optimizations. Chapter 5 contains our test results and analysis. Chapter 6 contains our conclusion and possible future work, which might expand on our results.

2 Preliminaries

In this chapter we define several terms used in this thesis. Then we describe the algorithms that are fundamental to our chosen approach. In addition we explain two algorithms that were used to preprocess our data.

2.1 Terminology

A *graph* $G = (V, E)$ consists of a set of *nodes* V and a set or multiset of pairs of nodes u, v called *edges* E. In a *directed graph* or *digraph* G contains a set or multiset of ordered pairs (u, v) often called *arcs* A instead of the edges. The arc (u, v) is an outgoing arc of u and an incoming arc of v. As we only deal with directed graphs we use often use the word edge, if the meaning is clear and use the word arc, if edge would be ambiguous.

The two nodes in an edge are called *adjacent* or *neighbors*. An edge u, v (or an arc (u, v)) is called incident to the nodes u and v and vice versa. In a digraph, the arc (u, v) is a called *outgoing* arc of the node u and *incoming* arc of the node v.

A *walk* is a sequence of alternating nodes and edges starting and ending with a node. A *path* is a walk, which does not contain a node more than once. As we deal with a road network we often use the term *route* interchangeable with path. The *length* of a walk or path is the number of edges on it. Each edge (u, v) might have an associated length (sometimes called weight) $d(u, v)$, in which case the length of a walk is the sum of the lengths associated with the edges on the walk. The *distance* between two nodes is the length of the shortest path.

A *cycle* is a path with at least one edge, where the first and last node are the same. A cycle of length 1 is called a *loop*. A graph is called *acyclic* if it does not contain a cycle. In a digraph a *directed* cycle, is a cycle where all arcs oriented in the same direction.

A graph is called *connected* if there exists a path between each pair of nodes. A digraph is *strongly connected* if for each pair of nodes $\{u, v\}$ there exists a path from u to v and a path from v to u. If for each pair there exists at least a path from u to v or from v to u, the digraph is called *weakly connected*.

A *tree* is a connected acyclic graph. It might have a special *root* node and is called *rooted tree* then. Of two connected nodes in a rooted tree the one closer to the root is called *parent* node and the other one is called *child* node. The other children of the same parent as called siblings. A directed, acyclic graph *DAG* is a digraph without directed cycles. A *depth-first search* visits all nodes in a component starting a node (the root) and at each node continues with a child node. It visits the siblings if all children of the child node have been visited.

A *polygon* is a plane figure that is bounded by straight line segments that from a loop. This segments are also called *sides* and the points where to sides meet are called *corners*.

Given two partially ordered sets A and B with the elements $a, a' \in A$ and $b, b' \in B$ the *lexicographic* order on the Cartesian product $A \times B$ is defined as $(a, b) \leq (a', b') \iff (a < a') \vee \big(a = a' \wedge b <= b'\big), a, a' \in A, b, b' \in B$.

A *n-orthotope* or *hyperrectangle* is the multidimensional generalization of the rectangle, the Cartesian product of intervals. It has an *hypervolume* or just *volume* that is calculated as the product of these intervals. It can be defined by two points implying all axis of the n-orthotope are parallel to the principal axis.

The *objective space* or *solution space* is the set of all possible solutions.

Multi-objective or *Pareto optimization* involves optimizing more than one objective function at once. We use a_i to denote the value of the ith objective. A solution a (strongly) *dominates* another solution b (assuming a minimization problem), if for all objective functions a_i is better than or as good as b_i and better for at least one objective function. If it dominates the other solution or is equal, we say it *weakly dominates* the other solution. As we deal with a minimization problem we write $a \prec b$, if a dominates b and $a \precsim b$ in case of weak dominance. Using our notation we define the dominance relation as $a \prec b \iff \forall i : a_i \leq b_i \wedge \exists j : a_j < b_j$. We use $a \succ b$ to denote a is dominated by b and $a \not\prec b$, if a does not dominate b.

The dominance relation is transitive ($a \prec b \wedge b \prec c \implies a \prec c$), which can be easily deducted from the definition: $a \prec b \wedge b \prec c \iff \big(\forall i : a_i \leq b_i \exists j : a_j < b_j\big) \wedge \big(\forall i : b_i \leq c_i \wedge \exists k : a_k < b_k\big) \iff \forall i : a_i \leq c_i \wedge \exists j : a_j < c_j \iff a \prec c$. It is also irreflexive ($a \prec a \implies \exists i : a_i < a_i$, which is a contradiction) and asymmetric: $a \prec b \implies b \not\prec a \iff \neg\,(\forall i : b_i \leq a_i \wedge \exists j : b_j < a_j) \implies (\nexists i : b_i \leq a_i) \vee (\forall j : b_j < a_j) \impliedby \forall i : a_i \leq b_i \wedge \exists j : a_j < b_j$, which is a contradiction. On the other hand weak dominance is reflexive ($a \precsim a \iff \forall i : a_i \leq a_i$) and not asymmetric (as it is reflexive, it cannot be asymmetric as well).

Two solutions a, b are not *Pareto comparable*, if neither $a \precsim b$ nor $b \prec a$ is true, which we denote using $a \parallel b$. This relation is not transitive: e.g. $(2, 3) \parallel (3, 1) \wedge (3, 1) \parallel (1, 2) \not\Longrightarrow (2, 3) \parallel (1, 2)$.

The *Pareto set* P is the set of solutions which contains all solutions not dominated by any other solution. All elements in the Pareto are not Pareto comparable ($\forall p, p' \in P : p \parallel p'$).

A *strict weak ordering* is a binary relation $<$ (comparison function) on a set, where the comparison function is transitive, irreflexive, and asymmetric and $\neg\,(a < b \vee b < a)$ is transitive.

2.2 Basic Algorithms

In this section, we introduce the algorithms that are used in our approach. In addition to the route planning algorithms we also explain an algorithm to extract the strongly connected Components in a digraph, which is required to filter our input data. As the last

algorithm, we describe the algorithm used to find out which nodes are inside beautiful areas.

2.2.1 Dijkstra's Algorithm

Most route planning techniques are based on Dijkstra's Algorithm [9], which is an algorithm to find shortest paths in a graph. It can be used to find the shortest path from a source s to a target t or all shortest paths originating in s.

The algorithm maintains a set of the nodes (further called queue), that still have to be visited. It visits the nodes in a graph in the order of their distance to the source node s, starting with s and the tentative distance 0 as the first entry in the queue. The *tentative distance* of an node v with an incoming edge (u, v) is the sum of the known distance from s to u and the length of the edge from u to v. At each step the entry (containing the tentative distance to an node) with the shortest tentative distance from the source s is taken from the queue. This is also called *extracting* an entry from the queue. When a node is visited (i.e. taken out of the queue) the first time, its distance is (from the entry in the queue) is written to the current node and its outgoing edges are scanned and the adjacent nodes with the tentative distances are added to the queue of the nodes to be visited. If the node was already visited before, the extracted entry is discarded. The step of scanning the incident edges of the current node and adding the tentative distances to the queue is also called *relaxing* the edges. The distance is tentative because it might be that a shorter distance is or was in the queue or is added later. It is final (or discarded) after it is extracted from the queue. To just find the shortest path to a target node t, the search can be stopped after extracting t from the queue, otherwise the search ends if the queue is empty. The edges used to get the final distances form a tree, called *shortest path tree*, and adding a new edge to it is called *expanding* this tree. This tree is not unique as there could be two shortest paths from s to an node u.

Modern implementations (e.g. [13]) use a heap-based priority queue to save the nodes, that need to be visited, with the tentative distances as priorities. This results in a worst case running time of $O(m + n \log(n))$ [13].

2.2.2 A* Search Algorithm

The A* Algorithm [19] modifies the order in which the nodes are processed. The priority of an node v is changed by adding a heuristic estimate (called *potential* and denoted $\pi(v)$) of the remaining distance so that nodes closer to the target are visited earlier. As the heuristic estimate is specific to the target node, A* Search needs such an estimate for each target node and it is more involved to use to find the distances from the source to all other nodes. A* Search starts with the source node s and the heuristic estimate $\pi(s)$ as the initial entry in the priority queue.

As long as the heuristic never overestimates the distance to the target the resulting distances are correct and the heuristic is than called *admissible*. If for each adjacent pair (x, y) the difference of the potentials $\pi(x) - \pi(y)$ is smaller or equal to the length of

the edge (x, y), nodes are only processed once and the heuristic is called *monotone*. The Algorithm is can also be seen as generalization to Dijkstra's Algorithm with the priority of a node set to $dist(s, u) + \pi(x)$ [1], where Dijkstra's Algorithm uses 0 as potential for each node.

If the heuristic estimation is exact (and the distance is unique), A* Search only expands the path to the target node, as the priority does not change along this path, and only the adjacent nodes to the nodes on this path are added to the priority queue.

2.2.3 Tarjan's Strongly Connected Components Algorithm

As our input data might contain weakly connected or unconnected components we use Tarjan's Algorithm [36] to keep only the largest strongly connected component. Otherwise we could get empty routes or other surprising results, especially with random queries. Tarjan's Algorithm uses depth first searches on an all unvisited nodes in a digraph. Each node get an unique *id* in increasing order and a second id, called *lowlink*, which represents the lowest number that can be reached from this node, which is initially its own id. As a node is visited it is placed on a stack and one of its child nodes is visited afterwards.

If the child node has a lower id than the parent node it has been visited before and the lowlink of the parent is set to the id of the child, if it is smaller than its current lowlink. Otherwise the depth first search continues with a child and after the search returns from this branch the lowlink of the parent is set to the lowlink of the child, if it is smaller.

After all children are processed, the nodes are popped from the stack until the parent node is reached and added to the same strongly connected component. The worst case performance of this algorithm is in $O(|V| + |E|)$.

2.2.4 Ray Casting Algorithm for the Point in Polygon Test

As we also consider the environment of our route, we use a point-in-polygon test to check if a route segment is in a nice area, which is described by a polygon in our input data. To find out if a point is inside or outside a polygon, a ray can be casted from somewhere outside the polygon to the point. This algorithm was already known in 1962 [32], but the original source is unknown. On the ray, the number of intersections with the polygon borders is counted and if it is odd, when it reaches the point, the point is inside the polygon, otherwise it is outside.

As some areas in our input data consists of multiple polygons, we need a point-in-polygon test, which works nonetheless. The ray-casting algorithm works even with multiple polygons and even if these are not oriented. We sort the edges of each polygon based on the minimum in the first dimension and afterwards we keep an active set of edges sorted by the maximum in this dimension. As the algorithm needs to sort all polygon lines and the nodes in our graph, we need at least $O(l \log l + n \log n)$ with l and n as the number of lines respective the number of nodes.

We repeat the following for all polygons and all way nodes. For each line (latitude in our case) we start at $-\infty$ (longitude) with a crossing counter initially set to 0 and at each

crossing we increase the crossing counter. For each point of interest along the line the point is inside the polygon if the crossing number is odd, otherwise it is outside.

For the actual point-in-polygon we only know that for each node/point all lines on the same row need to be considered. So in the worst case all lines will cross all rows and each point would be on a different row. We would end up with an additional $O(nl \log l)$ complexity, as we need to sort all intersection points for each row. In practise the lines are much shorter and often multiple nodes are on the same row, and therefore the running time is much lower.

3 Criteria Selection

As bicyclists are interested in more than just the travel time or the length of a route, we introduce the criteria we considered in this chapter. In addition to these two criteria bicyclists could consider the slope or the safety of the route or how beautiful the scenery is next to the route. As they are likely interested in optimizing multiple criteria, we analyse these criteria and the routes that we get, if we search routes with the optimization of these criteria and different combinations of these criteria in mind.

For each criteria the motivation is given as well as the way its value is calculated. All criteria are positive and to be minimized, as otherwise it would be hard to avoid cycles, which do not improve the quality of the resulting route. If we would maximize a criterion we could improve the resulting route by adding a non-negative cycle to the route. If we allow negative edge weights, if would be complex to avoid negative cycles, which lead to the same problem, that routes could be arbitrarily improved by using these cycles arbitrarily often.

The calculation of each criterion for each edge is done in a preprocessing step. After this step the cost of each edge is fix.

Distance. The first criterion we use is the length of the route, which is often important to bicyclists. As the shortest theoretically possible route is the straight line between the source and the target, this criteria implies the shortest detours and therefore likely leads to fewer or less sharp turns (but this cannot be guaranteed). As our criteria are based on weighted route segments, this criterion should correlate rather well with the other criteria.

The distance can be calculated directly from OpenStreetMap data [4]. In OpenStreetMap, all ways are represented as series of nodes and each pair of consecutive nodes represents a straight way segment. For each node, the position is given in nano degrees in relation to the World Geodetic System (WGS 84, see [28]); we calculate the distance between adjacent nodes using Vincenty's formulae [40] (as seen in Algorithm 1). The precision we use to safe the distance is 1dm. The length of a route is given as the sum of the respective edge lengths.

Travel Time. Another criterion, that we consider, is the travel time. Often, this criterion is even more important than the distance, especially when in a hurry, but in other situations (e.g. bike tours) it does not matter that much.

We determine the travel time on an edge in our graph as the quotient of the distance and the travel speed. The speed is given in Table 3.1, which maps OpenStreetMap tags to an expected travel speed. If multiple tags apply, which can occur on unpaved roads, we

$lat_1, lon_1 \leftarrow$ First coordinates (using radians);
$lat_2, lon_2 \leftarrow$ Second coordinates (using radians);
$a = 6378137$;
$b = 6356752.314245$;
f is the flattening of the ellipsoid;
$f = (a - b)/a$;
$U_1 = \arctan((1 - f)\tan(lat_1))$;
$U_2 = \arctan((1 - f)\tan(lat_2))$;
$L = |lon_1 - lon_2|$;
Loop
  l_1, l_2 are the longitudes on the auxiliary sphere;
  $l_1 = \cos(U_2) \times \sin(\lambda)$;
  $l_2 = \cos(U_1) \times \sin(U_2) - \sin(U_1) \times \cos(U2) \times \cos(\lambda)$;
  σ is the arc length between points on the auxiliary sphere;
  $\sigma = \arctan(\frac{\sqrt{l_1^2+l_2^2}}{\sin(U_1)\times\sin(U_2)+\cos(U_1)\times\cos(U_2)\times\cos(\lambda)})$;
  α is the azimuth at the equator;
  $\sin(\alpha) = \cos(U_1) \times \cos(U_2) \times \sin(\lambda)/\sin(\sigma)$;
  $c_\alpha^2 = 1 - s_\alpha^2$;
  $cs2s = \cos(\sigma) - \frac{2\sin(U_1)\sin(U_2)}{c_\alpha^2}$;
  $C = \frac{f}{16} \times c_\alpha^2 \times (4 + f \times (4 - 3 \times c_\alpha^2))$;
  $\lambda = L + (1 - C) \times f \times s_\alpha \times (\sigma + C \times \sin\sigma \times (\cos(\sigma) - 2\times))$;
  if $\lambda <= 10^{-12}$ *or after 150 iterations* **then**
    $u^2 = \cos^2\alpha\frac{a^2-b^2}{b^2}$;
    $A = 1 + \frac{u^2}{16384}\{4096 + u^2[-768 + u^2(320 - 175u^2)]\}$;
    $B = \frac{u^2}{1024}\{256 + u^2[-128 + u^2(74 - 47u^2)]\}$;
    $D = \cos(2\sigma_m)(-3 + 4\sin^2\sigma)(-3 + 4\cos^2(2\sigma_m))$;
    $E = \cos\sigma(-1 + 2\cos^2(2\sigma_m)) - \frac{1}{6}BD$;
    $F = \cos(2\sigma_m) + \frac{1}{4}BE$;
    $\Delta\sigma = B\sin\sigma F$;
    return $bA(\sigma - \Delta\sigma)$
  end
EndLoop

Algorithm 1: Vincenty's formulae

Table 3.1: Travel speeds on bicycles (B), pedelecs (P) and S-Pedelecs (S). The link-tags (e.g. primary_link) are considered equal to the respective highway type as in the OSRM data.

OSM-Tag		Speed (km/h)		
		B	P	S
highway	trunk	15	22	35
	primary	15	22	35
	secondary	15	22	35
	tertiary	15	22	35
	residential	15	22	35
	unclassified	15	22	35
	living_street	15	22	35
	cycleway	15	22	35
	road	15	22	35
	service	15	22	35
	track	12	15	15
	path	12	15	15
	footway	6	6	6
	pedestrian	6	6	6
	platform	6	6	6
	steps	2	1	1

OSM-Tag		Speed (km/h)		
		B	P	S
bridge::movable		5	5	5
surface	asphalt	15	22	35
	cobblestone:flattened	10	10	10
	paving_stones	10	10	10
	compacted	10	10	10
	cobblestone	6	6	6
	unpaved	6	6	6
	gravel	6	6	6
	fine_gravel	6	6	6
	pebbelstone	6	6	6
	ground	6	6	6
	dirt	6	6	6
	earth	6	6	6
	grass	6	6	6
	mud	3	3	3
	sand	3	3	3

take the minimum of the results. If the road has as speed limit, then we restrict the speed to it. The speeds for the bicycle are taken from the OSRM-project[1]. The unit we use is 10ms. To obtain the travel time of a route we sum up the travel times of its edges.

Elevation Difference. We could use the elevation difference of the route as possible criterion, but plain differences in elevation alone provide mostly uninteresting additional routes and also result in bad running times due to the extremely large search space. Therefore, we only incorporate it into the energy consumption criterion explained below.

We can deduct the difference in elevation between two coordinates from the data provided by the Shuttle Radar Topography Mission [12]. The Shuttle Radar Topography Mission provides elevation data sampled over a grid of 1 arc-second by 1 arc-second (approximately a 30 m). Previously only a lower resolution data set was available, but since August 2015 this data is accessible publicly.

Energy Consumption. To model the energy consumption criterion we use a linear combination of the distance (d) and the elevation difference (a) between the two points of the edge. If there is a descent, we set a to 0. We use the formula $8 \times a + d$, which is based on a rule of thumb used in hiking called Naismith's rule. With this criterion we take into account the varying energy requirements of different routes. However, on a steep dirt track the pedelec requires more energy than on a asphalted flat road. An improved energy consumption model could also take the surface of the road into account or further influences. We expect a better energy consumption model to provide even better results and this criterion could be amended in the future.

Of course, this criterion is also important, if a non-electric bicycle is used. It therefore also be called exhaustion or similar. This is also a very common criterion also used in prior research [35] of route planning for bicycles.

In another study located in Zurich by Menghini et.al. [27] the authors conclude that the product of the route length and the maximum gradient was the only criterion they indentified, which had an impact. Other studies identified a lot more criteria (e.g. [20]).

Safety. In many studies (e.g. [20], [42]) this criterion was considered as very important to cyclists. Safe routes often lead to choosing cycling over other modes of transport. On the other hand, a lack of a safe route is often the reason, that deters people from using a bicycle.

To account for safety, we associate a safety factor with each edge, which we calculate as the product of the distance (km) and the value found in a lookup table, see Table 3.2. The data in the lookup table is based on a study by Teschke [37] and mapped to OpenStreetMap tags. For roads, which could no be found in the lookup table, we use a default value of 80. We calculate the safety of a route as before as sum of its edges.

[1] project-osrm.org

Table 3.2: Safety of different roads. Smaller values are better.

OSM-Tag	Safety
trunk,primary-tertiary	
w/ parked cars	
No bike infrastructure	100
Separate bike lane	69
w/o parked cars	
No bike infrastructure	63
Separate bike lane	54

OSM-Tag	Safety
residential	51
bicycle road or route::bicycle	49
cycleway::track	87
highway::cycleway	59
designated:bicycle	59
highway::track or path	
surface::asphalt	79
other surface	73

Figure 3.1: Coverage of the tags used for the niceness criterion in Karlsruhe. Most of the areas inside the city are parks. The large areas in the north and the south are forests. The areas are shown even if no way passes through it. The image was created using http://overpass-turbo.eu/.

Table 3.3: Niceness of different surrounding areas assuming a sunny summer day (smaller is better).

OSM-Key	OSM-Value	Niceness
landuse	meadow	80
landuse	grass	80
landuse	farmland	80
landuse	orchard	80
landuse	allotments	80
landuse	forest	80
natural	wood	80
natural	scrub	80
natural	river_terrace	80
natural	heath	70
natural	moor	70
leisure	park	70
leisure	nature_reserve	70
tourism	attraction	70
default value		100

Niceness of the Surrounding Area. The environment can substantially influence the perceived quality of a route as bicyclists might enjoy the scenery during their trip. In fact, a beautiful scenery was identified as the second best motivator to cycle in "Motivators and deterrents of bicycling" [42]. If a bicycle tour is planned, it might even be the reason to look for a route in the first place. With this criterion we rate the view and other aspects, which are determined by the surrounding area of the route (e.g. a forest might provide shade). In the remainder of the thesis this criterion is often just called "niceness".

For each node we calculate, if it is inside a nice area, which are described in OpenStreetMap as closed ways or collections of ways called relation, both of which describe a polygon. From OpenStreetMap we extract all polygons which have one of the key-value-combinations given in Table 3.3.

Inside cities large beautiful areas are usually tagged as parks. As an example the nice areas inside Karlsruhe are given in Figure 3.1. Using this tag we can distinguish the large beautiful areas inside cities. This criterion is rather subjective, so personal preferences might vary. Other tags are either rare or are attributed to very small areas, so they are less useful to reliably rate some nodes as better than others, e.g. areas tagged with "leisure=playground" are usually inside areas tagged with "leisure=park" and/or they are too small to have an impact on the sightliness of a route. Another example would be the tag "amenity=campus" which could be a campus, which might be sightly, or just an area with university buildings, which could be indistinguishable from other parts of the city.

As with the other criteria we minimize this criterion, so we can search for the shortest route and avoid the problem of 'improving' the route with cycles. If a route edge is inside

a ‘beautiful’ area the value is taken from the lookup table 3.3. Else the value defaults to 100. This value is multiplied with the length of the respective edge. In our scenario we assume a sunny summer day on which forest may be a preferred environment.

To compute the niceness of an edge we use a simple ray casting algorithm to calculate if a node of the way is inside a polygon[2]. The niceness of a way is computed as the average of the niceness of its end nodes multiplied with the length of the edge. We process all polygons of one kind of area at once and we repeat this for each kind of area.

[2]The algorithm is described in Section 2.2.4

4 Our Approach

In this chapter we explain our approach to finding the Pareto set and what heuristics we used to reduce the number of resulting routes. At first the basic algorithm is explained, which uses a variant of the A* search with an heuristic function that is based on prior Dijkstra searches and finds the whole Pareto set. We call the function that prioritizes the entries in our priority queue *priority function*. It is a strict weak ordering, i.e. a binary relation $<$ that is irreflexive, transitive, and asymmetric and $\neg(a < b \wedge b < a)$ (written $a \sim b$) is transitive as well. With this properties all elements, that cannot be compared pairwise using $<$ form an equivalence class, i.e. all entries in the priority queue can be partitioned using this function.

We also explain the different priority functions we use in the priority queue of our A* search and the different filtering techniques we apply during our search.

4.1 Basic Approach

Our approach is based on the A* search algorithm. To get the heuristic estimates we start a Dijkstra search for each criterion, that just uses this criterion on the graph, which has all edges reversed (later called *reverse Dijkstra searches*). The resulting heuristic function, hereafter called *potential* and denoted π, is admissible, i.e. never overestimates the length, which is required to ensure the correctness of the A* search. It is also monotone (consistent), i.e. it satisfies the additional condition $\pi(x) \leq d(x, y) + \pi(y)$ (using $\pi(x)$ as the potential of the node x and $d(x, y)$ as the length of the edge from x to y). In our algorithm both properties required as otherwise the label added to the priority queue could dominate a previously added label. The label l extracted in the current step is referred to as *current label* and the corresponding node as *current node*. The set of labels associated with a node x is called *bag* $L(x)$. Before adding a label to a bag we remove the potential $\pi(x)$ of the node. To refer to the value of the ith criterion of a label we write l_i. Likewise we use $\pi_i(x)$ and $d_i(x, y)$ as the value of the ith criterion of the potential of x respective of the length of the edge (x, y).

As noted in Section 2.1 we used $a \prec b$, if a dominates b, $a \precsim b$, if a weakly dominates b, and $a \parallel b$, if a and b are not Pareto comparable.

Theorem 1. *The heuristic estimation given by the results of our Dijkstra search $\pi(x)$ always weakly dominates the distance to a neighbor y visited using the edge (x, y) with the cost $d(x, y)$ plus the heuristic estimation $\pi(y)$.*

Proof. We prove this by contradiction: If $\pi_i(y) < \pi_i(x) + d_i(x, y)$, the edge (x, y) should have been found by the Dijkstra search for the ith criterion and been used to compute

$\pi(x)$, which would lead to $\pi_i(x) + d_i(x, y) = \pi_i(y)$, which is an contradiction. As this is holds for all criteria, we get $\pi(x) + d(x, y) = \pi(y)$, if the edge (x, y) is part of a shortest path in all criteria, and $\pi(x) \prec d(x, y) + \pi(y)$, if it is not. □

Theorem 1 leads us to:

Lemma 1. *Labels created by relaxing edges are always weakly dominated by the current label.*

Proof. Using Theorem 1 we just add the current label l to both sides of the equation: $l + \pi(x) \precsim l + d(x, y) + \pi(y)$. □

As already mentioned we first use a reverse Dijkstra search for each criterion involved in our later A* search. As a result we get a potential for each node in our graph, that we can use in our A* search. The we start our A* search with the potential, that contains these distances from the source node s to the target node t.

During the A* search after extracting a label l that belongs to the node x, we check if it is weakly dominated by a label in $L(t)$. If it is we continue with the next entry in the priority queue. Otherwise, we subtract the potential to get $l' = l - \pi(x)$ and check if it is already weakly dominated by a label in the bag $L(x)$. If it is not, we add it to be bag and proceed with the edge relaxation step. At this step we check for each edge (x, y), if $l'' = l' + d(x, y)$ is already weakly dominated by a label in $L(y)$ or if $l'' + \pi(y)$ is already dominated an entry in the bag of the target node. If it is no we add $l'' + \pi(y)$ to the priority queue.

At the beginning of our A* Search we already know some routes of our Pareto set due to our previous single-criterion Dijkstra searches. We could use only these to prune our A* Search to avoid some label comparisons at each node.[1] Although using all routes found during our A* search leads to an unknown number of comparisons at each node, using all previously known routes turned out to be the fastest.

We tried several different priority functions for our Priority Queue[2], mainly due to the fact that with the heuristics introduced later, the priority function also impact the set of resulting routes. While the priority function used does not change the result, if no heuristics are used, it can still have an impact on the running time. If a label that dominates many other labels is extracted earlier, fewer labels are added to the priority queue.

With our approach we need to prove that entries, that are extracted later cannot be dominated by entries that are extracted earlier as settled entries will never be removed. For this we need to prove that each label added to the priority queue is greater or equal compared to the current label using this priority function. We also need to prove that a label extracted later (which is greater or equal to the current label) cannot dominate a label

[1] Or use some/all routes from the Dijkstra searches and/or only some of the routes found during the A* search. E.g. we also tried to use a subset of the size $\log |L(t)|$.

[2] Due to the intransitivity of $\neg (a \prec b \vee b \prec a)$, $\prec$ cannot be used as a priority function as it does not define a strict weak ordering.

extracted earlier. We do not prove that the labels are extracted in the correct order as we rely on a well known priority queue implementation[3].

4.1.1 Lexicographic Ordering

The first ordering we examine for our priority queue is the lexicographic order. We use $l <_{lex} l'$ to denote that l is lexicographically smaller than l'. We also subscribe other relational operators with "lex" to express a lexicographic comparison. Instead of l we also write $(l_1, \ldots, l_n)$, if we need to refer to a single criterion and we write $l = l'$, if l and l' have the same values in each criterion. We continue with the proof, that labels inserted during edge relaxations are lexicographic greater or equal to the current label (the label we just extracted from the priority queue).

Lemma 2. *Labels added to the priority (l') queue are lexicographically greater than or equal to the current label (l).*

Proof. Using Lemma 1 we just need to prove that weakly dominated labels are lexicographically greater or equal.

$$l \precsim l' : \begin{cases} l = l' \iff \forall i : l_i = l'_i \implies l =_{lex} l' \\ l \prec l' \iff \forall i : l_i \leq l'_i \wedge \exists j : l_j < l'_j \\ \qquad\qquad \implies \forall 1 \leq i \leq j : (l_i \leq l'_i) \wedge (l_j < l'_j) \\ \qquad\qquad \implies l <_{lex} l' \end{cases}$$

This concludes our proof. □

After we have proven that we do not add labels to the priority queue that are lexicographically smaller than the current label, we need to prove that labels extracted later cannot dominate labels extracted earlier. Due to Lemma 2 we know all labels, that we extracted previously, are smaller than or equal to the current label.

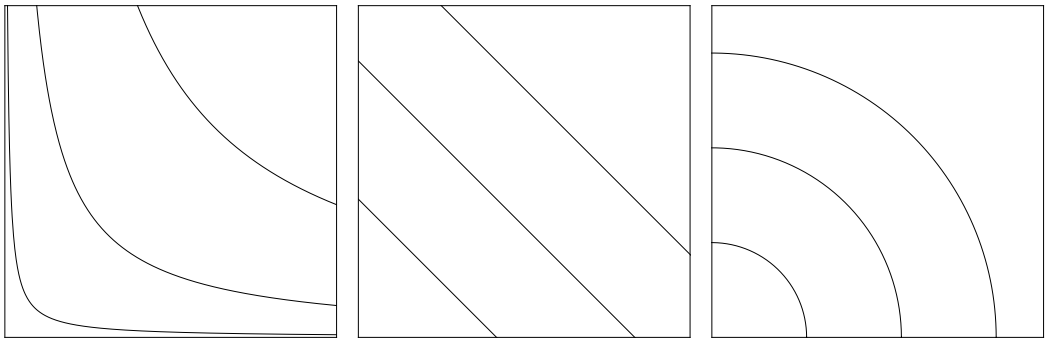
Lemma 3. *A Label cannot dominate a lexicographically smaller or equal label.*

Proof.

$$l \leq_{lex} l' : \begin{cases} l =_{lex} l' \iff \forall i : l_i = l'_i \implies a = l' \implies \neg(a \prec l') \impliedby \neg(a \prec a) \\ a <_{lex} l' \iff \forall 1 \leq i \leq j : (a_i \leq b_i) \wedge (a_j < b_j) \\ \qquad\qquad \implies \exists j : (a_j < b_j) \implies \neg\big(\forall j : (a_j < b_j)\big) \\ \qquad\qquad \implies \neg\big(\forall j : a_j \leq b_j \wedge \exists i : a_i < b_i\big) \\ \qquad\qquad \iff \big(\neg(a_1, \ldots, a_n) \prec (b_1, \ldots, b_n)\big) \end{cases}$$

This concludes our proof. □

[3] C++ std::priority_queue as implemented by gcc-4.9



(a) Volume (b) Manhattan (c) Euclid

Figure 4.1: Priority functions: The values which have the same priority are shown as lines in the case of two criteria. Lexicographic ordering is not shown as each point has its own priority, if it is used.

4.1.2 Ordering Based of the Volume of the n-Orthotope

Another possible priority function for the priority queue we use is the volume of the n-orthotope defined by the point just below potential of the source node and the sum of the current label and potential of the current node (furthermore just called *volume* of the label). We use the point 1 lower in each dimension than the potential to avoid 0 as volume. With 0 as volume we cannot guarantee that labels with the same volume cannot dominate each other, e.g. there might be two shortest paths for the distance criterion, which both would have volume 0, although one of those path could be dominated by the other (e.g. worse in all other criteria). We could also use the real zero-point, but this would likely give results that are more similar to the Manhattan distance, we also evaluated (See also Fig. 4.1)[4]. With this ordering we also need to prove, that labels created by relaxing the edges have a larger volume than the current label (if we avoid volume 0).

Lemma 4. *Labels added to the priority queue have a volume greater than or equal to the current label.*

Proof. Again we use Lemma 1 to prove this. We use l to denote the modified label used to calculate the volume as described in the text, i.e. $l - \pi(s) - (1, \ldots, 1)$, to ensure all our values are strictly positive and we l' as the label created by edge relaxation.

$$l \precsim l' : \begin{cases} l = l' & \implies \prod i : l_i = \prod i : l'_i \\ l \prec l' & \iff \forall i : l_i \leq l'_i \wedge \exists j : l_j < l'_j \\ & \implies \prod i : l_i < \prod i : l'_i \end{cases}$$

□

[4]With the volume we get the nice mathematical property that multiplying the value in one criterion scales the volume by the same factor, i.e. of the criteria and we do not need to normalize it.

Next we prove that that labels extracted earlier from the priority, which have a smaller or equal volume, cannot be dominated by the current label. We use the same notion as in Lemma 4.

Lemma 5. *Labels cannot dominate other labels with a smaller or equal volume.*

Proof. Using $A \implies B \iff \neg B \implies \neg A$, we prove that $l \prec l'$ implies l to have a smaller volume than l' (written $l <_{vol} l'$).

$$\begin{aligned} l \prec l' &\iff \forall j : 0 < l_j \leq l'_j \wedge \exists i : l_i < l'_i \\ &\implies \textstyle\prod_i l_i < \prod_i l'_i \iff l <_{vol} l' \implies l \leq_{vol} l' \end{aligned}$$

□

4.1.3 Ordering Based of the Manhattan Distance

The next priority function we tried was the Manhattan distance d_{Man} from the zero-point[5] normalized using the potential of the source node: $d_{Man}(l) = \sum_i \frac{l_i}{\pi_i(s)}$.

As before we prove that labels added during edge relaxations are greater than or equal to the current label and that labels extracted from the priority queue (which are equal to or greater than previous labels) cannot dominate labels extracted earlier.

Lemma 6. *Labels added to the priority queue have greater than or equal Manhattan distance compared to the current label.*

Proof. Using Lemma 1 again:

$$l \precsim l' : \begin{cases} l = l' & \implies \sum i : \frac{l_i}{\pi_i(s)} = \sum i : \frac{l'_i}{\pi_i(s)} \\ l \prec l' & \iff \forall i : l_i \leq l'_i \wedge \exists j : l_j < l'_j \\ & \implies \sum i : \frac{l_i}{\pi_i(s)} < \sum i : \frac{l'_i}{\pi_i(s)} \end{cases}$$

That concludes our proof. □

Lemma 7. *Labels with a greater or equal Manhattan distance cannot dominate labels with a smaller Manhattan distance.*

Proof. Using contraposition again, we prove that $l \prec l'$ implies l to have a smaller Manhattan distance than l'.

$$l \prec l' \iff \forall j : l_j \leq l'_j \wedge \exists i : l_i < l'_i \implies \sum_i \frac{l_i}{\pi_i(s)} < \sum_i \frac{l'_i}{\pi_i}$$

That concludes our proof. □

[5]Using the potential instead just subtracts n, where n is the number of considered criteria

4.1.4 Ordering Based on the Euclidean Distance

The last ordering we evaluate is based on the normalized Euclidean distance to the potential of the source node: $d_{Euc}(l) = \sum_i \left(\frac{l_i - \pi_i(s)}{\pi_i(s)}\right)^2$. We again need to guarantee that labels added by edge relaxation are greater than or equal to the current label.

Lemma 8. *Labels added to the priority queue have greater than or equal Euclidean distance compared to the current label.*

Proof. We use the same approach as in Lemma 6.

$$l \precsim l' : \begin{cases} l = l' & \implies \sum i : \left(\frac{l_i - \pi_i(s)}{\pi_i(s)}\right)^2 = \sum i : \left(\frac{l'_i - \pi_i(s)}{\pi_i(s)}\right)^2 \\ l \prec l' & \iff \forall i : l_i \leq l'_i \wedge \exists j : l_j < l'_j \\ & \implies \sum i : \left(\frac{l_i - \pi_i(s)}{\pi_i(s)}\right)^2 < \sum i : \left(\frac{l'_i - \pi_i(s)}{\pi_i(s)}\right)^2 \end{cases}$$

That concludes our proof. □

Lemma 9. *Labels with a greater or equal Euclidean distance cannot dominate labels with a smaller Manhattan distance.*

Proof. Again the proof is analog to the one for Lemma 7:

$$l \prec l' \iff \forall j : l_j \leq l'_j \wedge \exists i : l_i < l'_i \implies \sum_i \frac{l_i}{\pi_i(s)} < \sum_i \frac{l'_i}{\pi_i}$$

That concludes our proof. □

4.1.5 Recapitulating the Priority Functions

The two proofs for the Euclidean distance are analog to the proofs for the Manhattan distance. Using the Lemmas 2 to 9 leads us to:

Theorem 2. *Lexicographic ordering and the orderings based on comparing the volume, the Manhattan distance or the Euclidean distance all ensure that an label added to the priority queue is never smaller than the label we just extracted and that an extracted label never dominates a label extracted earlier.*

We can use this approach to find the complete Pareto set, but it takes a long time to do so. Therefore we implemented some techniques to speed up to this process.

4.2 Speed-Up Techniques

In our graph all edges have nonnegative values for each criterion in our graph, so an edge that leads to the node we visited previously, cannot improve the routes found. Therefore

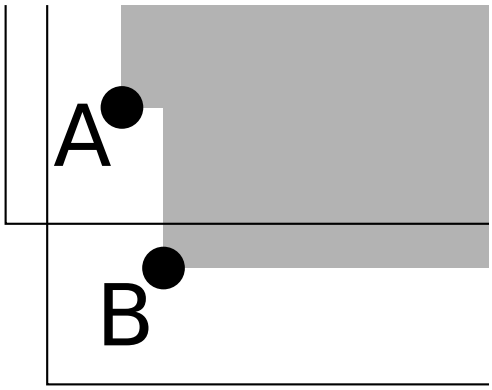


Figure 4.2: ε-dominance compared to dominance: With dominance the order in which the points are added does not matter. With the addition of an ε-environment we lose this property. If a route with the label A is found before a route with label B, both routes are added to the bag. Otherwise only B is added.

we can save many priority queue operations by avoid to insert the label we get from relaxing this edge. As the average degree is rather low in a road network this leads to a significant speed-up (see also [10]).

To further reduce the number of operations of the priority queue we can also avoid inserting elements into the priority queue, which we would extract in the next step as had been done before by Disser et. al. [10] and is called label forwarding. We actually use label forwarding only with lexicographic ordering as we know which edge leads to the node, which would be inserted next. The next element, which we will be extracted, is the element which led to the current node in the Dijkstra search, that was used to get the potential in the first criterion used.

4.3 Heuristic Route Filtering

As the number of routes can grow exponentially in the number of nodes (see[18]) even if only two criteria are used an algorithm that finds all Pareto optimal routes might also need time exponential to the number of nodes. As the Pareto set can grow very large, we use heuristics to reduce the number of computed routes. We examine two heuristics to reduce the number of resulting routes. The first is ε-dominance [41, 38, 24, 29] and the second is based on avoiding small detours if two routes join.

4.3.1 ε-Dominance

To reduce the size of the solution we use ε-Dominance instead of the ordinary dominance relation. ε defines an offset by which new routes found must be better than the previously found routes. With ε dominance a label l dominates another label l' if l dominates $l' + \varepsilon$ (assuming a minimization problem), e.g. with $\varepsilon = (2, 2)$ $(2, 1)$ is discarded if $(1, 2)$ was

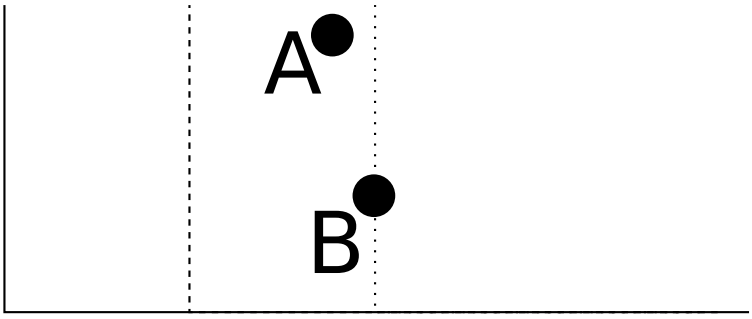


Figure 4.3: Problem with ε-dominance and lexicographic ordering (sorted by x-axis first): As A is always processed earlier, the size of the ε in the x-direction as indicated by the continuous, dashed and dotted lines does not have an impact. Whether B is included or not only depends on the size of the ε in the y-direction.

found earlier. As we deal only with minimization problems, we write $l \prec_\varepsilon l'$ to denote l ε-dominates l'. For a comparison of dominance and ε-dominance see also Fig. 4.2.

The effect of using ε-dominance is similar to using buckets or change the resolution used for each criterion (i.e. switching from 0.1m to 10m to measure the distance) as with this heuristic new routes have to be more than just infinitesimal better than the previously found routes[6].

One problem with ε-dominance is, that it is not transitive (e.g. $\varepsilon = (3,3) : (5,1) \prec_\varepsilon (3,3) \prec_\varepsilon (1,5) \not\Rightarrow (5,1) \prec_\varepsilon (1,5)$), which makes removing elements in the bag infeasible, because we do not know the elements that were previously discarded due to this element. But as we use a Label-Setting algorithm we avoid this problem, because we never remove routes we found previously.

With ε-dominance we do not have the property that labels extracted later cannot ε-dominate labels extracted earlier, but this can occur only if the label extracted later is ε-dominated by the earlier entry as well. On the other hand, it still cannot occur that entries extracted lated dominate those, that are extracted earlier.

As more routes are dominated with ε-dominance, this leads to significantly fewer found routes, as more routes are filtered out by the dominance check before adding a new label is added to the bag of a node. With two routes, which only differ by ε, we can only get one of these in our resulting set.

A problem occurs with lexicographic ordering and ε-dominance: As we extract the labels that are smaller in first criterion earlier the first criterion of the ε is never used. This is illustrated in Figure 4.3.

With ε-dominance we know that the amount of routes in the results is limited by the number of ε-environments that fit on the hypersurface which contains the Pareto set: $P_\varepsilon = \sum_i \prod_{j \neq i} \frac{\Delta l_j}{\varepsilon_j}$, where Δl_i is the difference between the maximum and minimum route length in the criterion i, and ε_i is the value of ε in this criterion. As we do not know the

[6]In theory, this is always the case, because all values on a computer system are truncated at some point

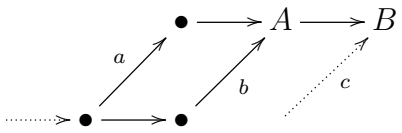


Figure 4.4: Route filter based on small detours: at the node A, if the route a was found earlier, b is discarded. At the node B no check would be done as the paths use the same incoming edge. A route c might be included at B, if it would be dominated by b and not dominated by a.

maximum route length we could only bound it by the sum of the longest edges at each node[7]. Although we do not know the exact value of Δc_i, we can estimate it using the results of our previous Dijkstra searches.

As this is a rather rough limit and we do not know the exact maximum route length in a single criterion, we try several ε based on the potential π (the minimal route lengths in each criterion).

4.3.2 Filtering based on small detours

Another possibility to filter routes might be to ignore routes that only contain a short detour from a previously found route (illustrated at Section 4.3.2). To do this we keep a *history* of n edges at each label and each entry of the priority queue. Every time a label is extracted from the priority queue, we compare the history of this label with the histories of the labels in the bag. If the incoming edge is different and the labels still have a common entry in the history, the new label is discarded. To reduce the number of comparisons we only check if the oldest edge of each history is present in the other history.

With this kind of filtering we would like to improve the quality of the filtered set, if seen as list of alternate routes. When searching alternate routes, we are not interested in small detours, but prefer a small set of routes which are not similar, when compared pairwise. The drawback with this heuristic is that it might introduce routes that are not Pareto, e.g. if in the example seen in Section 4.3.2 a route to B is found that would be discarded due to route b, is is included with this heuristic.

[7] The size of this hypersurface could also estimated really well by computing the convex hull with an approach similar to the on proposed by Sedeño-Noda and Raith [31], but this would likely take much too long.

5 Evaluation

In this chapter we first explain the input data used and how it is created. Then we briefly discuss the different criteria and there impact on the resulting set of routes as well as on the running time. Afterwards we evaluate our approach regarding different combinations of heuristics, priority functions and criteria with regard to the quality and the run time.

This chapter is concluded by case studies, where we review two queries with different heuristics and priority functions. The first query is rather short from Ettlingen to Durlach, where we still compute to Pareto set. The other one is from Baden-Baden to Bruchsal. For the second route the computation of the Pareto set would likely take days or even weeks as it is a rather large distance and it also passes through several cities, which leads to a large Pareto set.

The evaluation was done using the OpenStreetMap from the Region "Regierungsbezirk Karlsruhe"[1] with the data dating to November 25, 2015 at 22:23:00 UTC. For the analysis of the quality only a small segment of this map was cut out as otherwise this would lead to infeasible run times. If not stated otherwise, the evaluation was done on an Intel Xeon E5430 clocked at 2.66 GHz with 32GB RAM and involved 100 random queries. The program was compiled using gcc-4.9 on OpenSUSE 10.3 with the command flags "-O3 -flto".

5.1 Input Data

We get the required data for our route planning from the OpenStreetMap project [4]. The OpenStreetMap Project is a project to collectively map the world. Volunteers gather map data using GPS devices and supply this data on the website of the project. The data can be annotated with a variety of key:value pairs, which are entered as free text, but guidelines how to tag objects are given in the Wiki of the OpenStreetMap project. This user input is not very restricted to allow many special cases to be accounted for. User are also encouraged to create new tags[2] to improve the style of the map or support analyzes that rely on previously unmapped attributes.

As there is no elevation data in OpenStreetMap we additionally use the data from the Shuttle Radar Topology Mission (SRTM) [12, 21]. The SRTM produced elevation data for the most part of the inhabited earth and the areas not covered (mostly near the poles) are likely not interesting to bicyclists. As this data only has a resolution of about 30m, the values for the nodes are usually based on interpolation using this data.

[1] http://download.geofabrik.de/europe/germany/baden-wuerttemberg/karlsruhe-regbez.html
[2] http://wiki.openstreetmap.org/wiki/Map_Features

One of the problems with the input data regarding our approach is, that there can be multiple edges between two nodes, which also are not Pareto comparable. This leads to an increase in the number of Pareto optimal routes. In the worst case two edges between two nodes v and w could double the number of routes, that go through these two nodes, but often some of them are Pareto-dominated by unrelated routes.

5.1.1 Data preprocessing

OpenStreetMap can also contain areas where bicycles are allowed, but the access to this area might be blocked, e.g. we have seen this often with platforms. These areas are typically very small and most of them only contain 1-2 nodes. Therefore we remove all nodes and edges that are not part of the largest strongly connected component. To find the connected components we use Tarjan's algorithm (Section 2.2.3).

As in OpenStreetMap bends are modeled as a sequence of straight lines a large fraction of the nodes are actually used to model bends and do not contribute to additional routes. So we replace series of these nodes with a one arc for each direction (if it is a two way road). If a one-way road and a non-one-way road meet, we do not contract this node and if the nodes are connected by multiple edges, we also do not contract them. If the ways were just tagged differently, this was not treated in a special way, e.g. if there are some stairs on a way, this would lead to a single edge, which just takes longer to use as stairs only permit a very low speed (see Table 3.1).

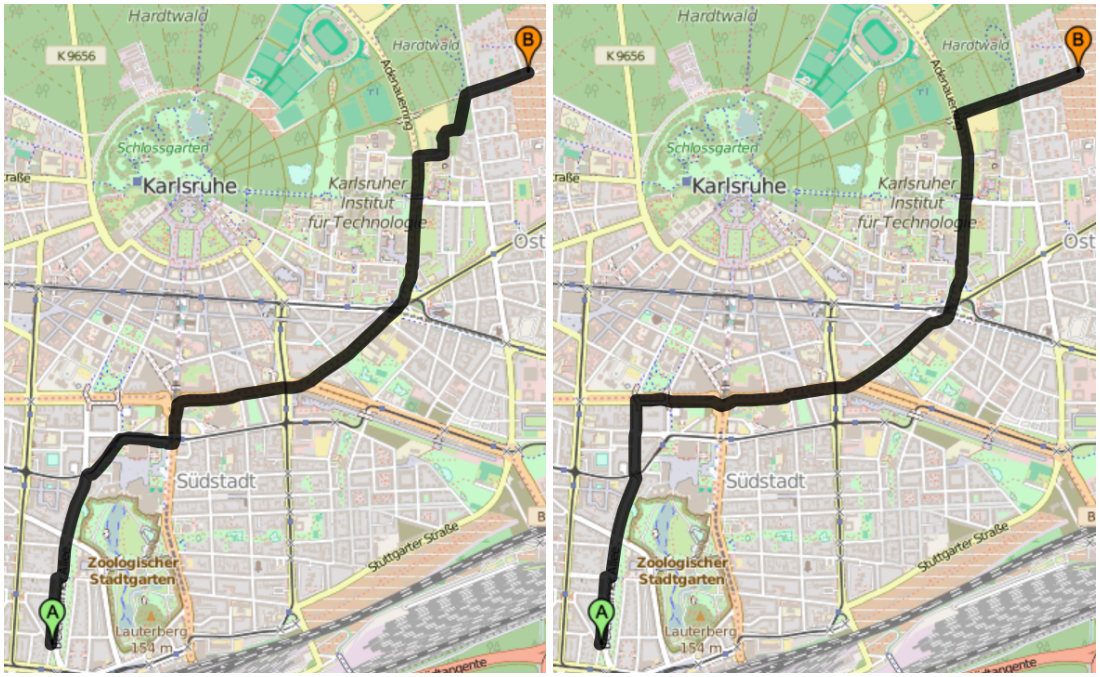
For the graph used in our experiments we therefore give the number of all nodes and edges (which are part of a way) before and after preprocessing. Our main graph ("Regierungsbezirk Karlsruhe") contains 1797052 nodes and 3844808 edges before these two preprocessing steps and 338479 nodes and 935880 edges after. All experiments were the Pareto set was searched were done on rather small segments of this graph.

5.2 Criteria

In this section we evaluate the impact of the different criteria on the number of routes found and the running time. The criteria have a large impact on the number of routes found and therefore on the running time. On the other hand a bad criterion can increase the running time significantly without contributing many new elements to the Pareto set.

The tests that involve computing the Pareto set were done on a small segment of the map: The bounding box used is defined by the two coordinates 48.995N 8.39E and 49.025N 8.435E[3]. This graph has 35702 nodes and 74172 edges before the preprocessing and 3735 nodes and 10692 edges after.

[3]The command used to extract the map segment was: "osmosis –rb karlsruhe-regbez-latest.osm.pbf –bb left=8.39 bottom=48.995 right=8.435 top=49.025 completeRelations=yes –wb karlsruhe-8.39-48.995-8.435-49.025.osm.pbf"



(a) Shortest Route: 4.9 km, 31 min (b) Fastest Route: 5.1 km, 21 min

Figure 5.1: Comparison of two routes with the same Source and Destination with Distance respective Time as sole criterion. In this case the shortest route uses many foot ways and takes 50% longer and is therefore probably unwanted

This graph consist mostly of the central and north eastern part of Karlsruhe. It contains a rather large nice area in the north (see Figure 3.1). As it is an urban area we have more nodes per area than we would have with an rural area.

5.2.1 Distance

Distance is our first criterion we evaluate, as all other criteria are based on it. Therefore it often leads to fewer alternative routes when combined with another criterion. It is important to state that we allow foot ways as well as stairs, but only with a very low speed. Therefore plain minimization of the distance can lead to strange routes, if many of these segments are used, which likely results in routes, that would be ignored by cyclists. But on the other hand, if only few or short segments are used, these routes might still be considered.

In the Figure 5.1 an example is given, where Distance alone gives a bad result, because most people will not put up with the increased travel time in this case.

Table 5.1: Comparison of the different Criteria Combinations and their influence on the number of found routes and running time. The criteria shown as ○ have not been used in the corresponding search. This experiment has been done using only a section of the Map and used lexicographic ordering as priority function. The abbreviations used are: Distance ⟶ D, Time ⟶ T, Energy Consumption ⟶ E, Safety ⟶ S, Niceness ⟶ N, Number of Routes ⟶ $|R|$, Average Query Time ⟶ $\varnothing t$, Label Comparisons ⟶ LCs, and Priority Queue Extractions ⟶ PQ Extr.

D	T	E	S	N	$\lvert R\rvert$	$\varnothing t$ ms	LCs	PQ Extr.
●	○	○	○	○	1.00	3.71	57865.7	6276.9
○	●	○	○	○	1.00	3.74	60394.7	6303.8
○	○	●	○	○	1.00	3.75	58176.7	6311.1
○	○	○	●	○	1.00	3.83	58649.9	6268.6
○	○	○	○	●	1.00	3.91	58259.9	6312.5
●	●	○	○	○	7.71	7.61	137544.5	12696.4
●	○	●	○	○	2.72	7.39	116538.1	12511.7
●	○	○	●	○	13.21	7.76	165226.2	12818.5
●	○	○	○	●	3.11	7.40	118198.1	12535.8
○	●	●	○	○	10.04	7.69	145277.0	12859.4
○	●	○	●	○	32.48	9.54	647294.6	14649.6
○	●	○	○	●	11.98	7.72	157858.1	12935.6
○	○	●	●	○	15.03	8.07	190936.7	13161.6
○	○	●	○	●	4.29	7.47	120881.6	12641.2
○	○	○	●	●	17.09	8.21	224933.6	13349.3
●	●	●	○	○	22.91	11.90	417440.0	19346.7
●	●	○	●	○	152.96	60.63	19933526.9	25414.5
●	●	○	○	●	24.62	12.09	454837.3	19313.1
●	○	●	●	○	24.95	11.92	350533.6	19264.4
●	○	●	○	●	6.99	11.08	186440.7	18806.0
●	○	○	●	●	30.08	12.72	656372.3	19600.4
○	●	●	●	○	169.63	56.37	17294288.3	26386.7
○	●	●	○	●	39.91	13.11	812777.1	20008.0
○	●	○	●	●	231.04	191.58	76627134.4	29428.8
○	○	●	●	●	41.24	13.44	894004.9	20323.3
●	●	●	●	○	282.93	253.30	87728275.8	36754.1
●	●	●	○	●	52.66	19.10	1651192.5	26348.3
●	●	○	●	●	422.60	1433.93	484639002.2	42872.7
●	○	●	●	●	50.35	18.77	1421894.8	26259.3
○	●	●	●	●	582.16	2461.95	636592885.7	46824.5
●	●	●	●	●	748.42	5493.79	1700103364.2	60557.9

5.2.2 Travel Time

This criterion combined with the distance criterion often leads to the same routes or only increases the number of total routes by an insignificant amount, at least compared to the other criteria.

This can accounted to the high correlation between these two criteria. Although with our configuration it is possible that there is a large difference between the fastest and the slowest route (2 - 15 km/h), the slow parts (2 km/h, only occurs on stairs) are usually short (or can be easily avoided) and typical routes consists almost exclusively out of fast parts. The combination with the safety criterion resulted in the most routes among all two criteria combinations. Often this led to more routes than we got with most three criteria combinations.

5.2.3 Energy Consumption

The third criterion is based on elevation difference and the distance between two nodes. At first we tried to use just the elevation difference as criterion, but this mainly increased the running time without a significant increase in the number of interesting results. Using the plain elevation difference the algorithm will often search lower areas before going up. Even with the exact elevation difference known, this will just avoid going up or down unnecessarily, but will still search large areas. Especially if the source and target node are at a low point compared to the nodes in between, the potential will probably not help much.

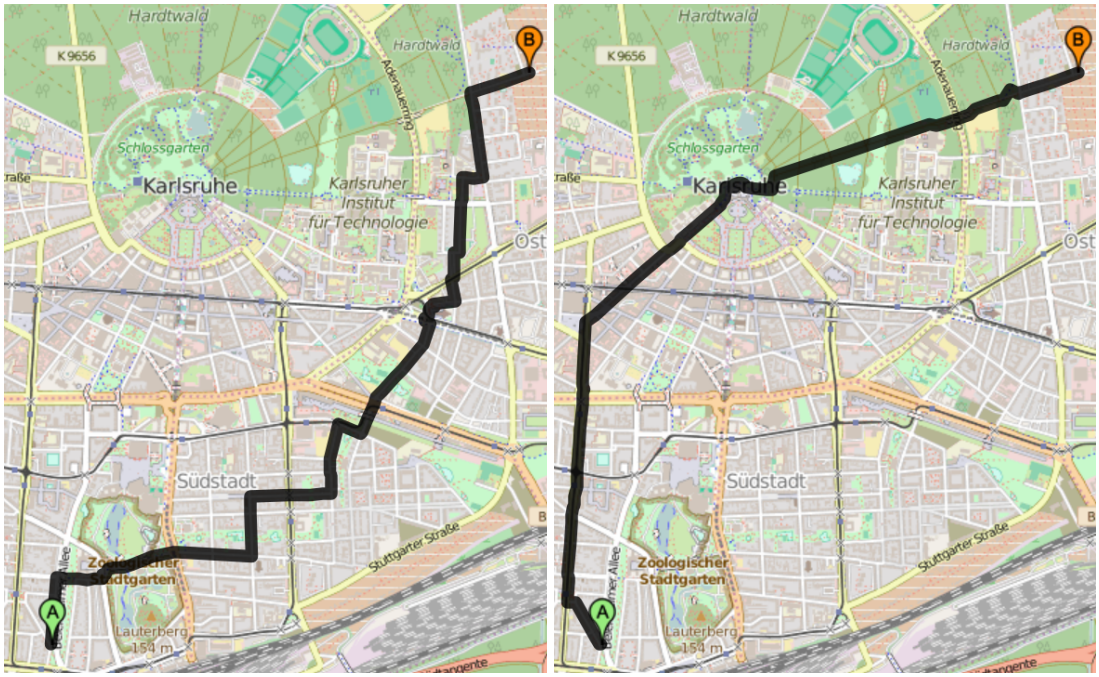
For the reason explained above we use a combination of elevation difference and distance. This still gives the interesting routes with a small elevation difference, but helps a lot in making the search more goal directed.

This criterion combined with the distance led to the fewest routes out of all criteria combinations.

5.2.4 Safety

The safety criterion often leads to a large number of additional routes. Although the safest alternative is just about twice as safe per distance as the unsafest, these alternatives are plenty. There is almost always a safer alternative to the fastest or shortest route and as parts of these can be combined with the already given routes, this leads to a large increase in the number of routes. Especially as the fast route is often also an unsafe one, the combination of these two criteria leads to a huge increase in the number of routes.

As seen in Table 5.1, if this criterion was ignored, it led to a large drop in the number of found routes.



(a) Safest Route: 5.5 km, 22 min (b) Nicest Route: 5.3 km, 24 min

Figure 5.2: Comparison of two routes with the same Source and Destination with Safety respective Niceness as sole criterion.

5.2.5 Niceness of the Surrounding Area

In general only plain distance and energy consumption led to less additional routes. The surrounding area often led to routes similar to those gotten with the safety criterion. Inside cities routes through parks are often also safe as they consist usually of cycleways or minor roads. Still often with the safety considered, we fund much more routes than with this criterion, if combined with other criteria.

5.3 Quality of the Heuristics

As we inspect the output of our heuristics, we want to analyse both the quality of the resulting route set as well as the reduced running time. All queries we evaluate in this section include all proposed metrics. First we introduce the functions use to measure the quality of the heuristic results.

To judge the quality of the heuristics we compare the reduced set of routes to the Pareto set and we assess the missed opportunities in the objective space. To measure the similarity of the set of routes generated by the heuristics compared to the Pareto set, we use the Sørenson-Dice Index [34, 8]. To account for missed opportunities in the objective space we calculate the average Euclidean Distance of the elements in the Pareto set to the closest

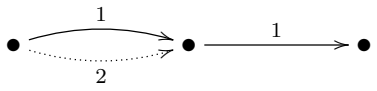


Figure 5.3: Weighted Sørenson-Dice Index: The similarity between these two routes is calculated as $\frac{2\times\text{Common route length}}{\text{Total route length}} = \frac{1+1}{(1+2)+(1+1)} = 0.4$. This is done for each route in the Pareto set and the most similar route in the filtered set.

element in the filtered set. The last quality measure used was the fraction of route in our filtered set that were not part of the Pareto set.

In the following sections we compare the results we get with with ε-dominance and different priority functions. Then we do the same with the heuristic based on small detours. Afterwards we look at possible combinations.

5.3.1 Quality Measurements

To evaluate the quality of the resulting routes we use the Sørenson-Dice index [34, 8] to measure the similarity of the Pareto set and the filtered set weighted with the length of the route segments. We calculate the Sørenson-Dice index for each route of the Pareto set compared the best match in the filtered set according to the Sørenson-Dice index and then take the average of all these Sørenson-Dice indices.

As the Sørenson-Dice index was originally developed to compare the similarity of two samples, we need to define what these two samples should be. If we just count the shared and all routes, we would just get the fraction of size of our reduced set divided by the sum of the sizes of the Pareto and the filtered set. Therefore we compare each route (p) in the Pareto set with the most similar route (p') in the filtered set using the Sørenson-Dice index with each edge weighted with the length of the edge (see also Fig. 5.3) and use the average (SD) as measure:

$$d(p, p') = \frac{2 \times \text{shared route length}}{\text{sum of the route lengths}}$$

$$SD = \frac{1}{|P^*|} \sum_{p \in P^*} d(p, p')$$

To account for the missed opportunities in the objective space, we calculate the average euclidean distance (ED) of the label of each element in Pareto set (c) to the closest label in the reduced set (c'):

$$\Delta p = \sqrt{\sum_i (c_i - c'_i)^2}$$

$$\frac{1}{|P^*|} \sum_{p \in P^*} \Delta p$$

The last measure used was just the fraction of routes in the filtered set, that are in the Pareto set as well:

$$\frac{|\text{Pareto set} \cap \text{Filtered set}|}{|\text{Filtered set}|}$$

5.3.2 ε-Dominance

We first compare the quality of the resulting routes with the Pareto set. We have evaluate three different definitions of ε. If we always use a minimum εm defined as (250, 600, 300, 400, 300). The units we used in our evaluation program are 0.1m, 0.01s, distance in$0.1\text{m} + 8 \times$ elevation difference in 0.1m, $\frac{\text{distance}}{4} \times$ Number found in Table 3.2, and $\frac{\text{distance}}{8} \times$ Number found in Table 3.3. So the minimum distance value of the ε was 25m and the minimum time was 6s.

As seen in Table 5.2 the usage of ε significantly decreases the number of routes found and the running time. It also shows the impact of the chosen priority function. We also see, that even without ε, the priority function has an impact on the running time. As the running time is lower with the Manhattan and Euclidean distance compared to the lexicographic ordering, although the number of priority queue extractions is higher, we assume that entries get extracted faster and the average size of the priority queue is smaller and therefore insertions and extractions are cheaper (both are $O(\log n)$ with n as the current size of the priority queue). The other reason, that could cause this behaviour, is that we add less entries to the target node early. Therefore we need less checks to determine if a route is already dominated by a route found previously. For instance, the lexicographical order first finds the shortest route[4]. Then the second shortest route is found, and so on. It could be beneficial to know these routes early, because more route can than be pruned due the routes we know at the target. On the other hand, if these route only dominate few routes found later, but need to be checked at every node extraction, it might have a detrimental effect. Based on the numbers in Table 5.2 we conclude, that the second effects outweighs the first, if the lexicographical order is used.

5.3.3 Small detours

We use the same metrics to compare the filtered set with the Pareto set as previously. As seen in Table 5.3 with this heuristic we get rather different results. Unlike before now lexicographic ordering gives very nice results, if small detours are prevented. On the other hand it does not improve, if routes with slightly longer detours are discarded. If only few routes are rejected, e.g. if detours shorter than 3 are forbidden and one of the priority functions 1-3 (see Table 5.2b) is used, then we still get are rather large result set. As expected the average quality is also higher if the resulting set is larger.

Compared to the results we get with ε-dominance, the quality we get if the running time and the size of the result set are similar, is much lower. We also get much more routes, that are not in the Pareto set in general. As as single criterion search lasted between 3.68

[4]The first criterion we use is the distance

Table 5.2: Comparison of different εs and priority functions: The Tables (a) and (b) explain the indices used in the first two columns of the Table (c). In addition to the abbreviations used previously we also use: $\varnothing km$ →average route length, SD →Sørenson-Dice index, Euc →Euclidean Distance, and Fr →Fraction of routes that are also in the Pareto set

(a) ε-IDs

#	ε defined by
0	No ε used
1	$\varepsilon_i = max(\pi_i^0.5(s), m_i)$
2	$\varepsilon_i = max(\pi_i(s)^{0.6}, m_i)$
3	$\varepsilon_i = max(\pi_i(s)^{\frac{2}{3}}, m_i)$

(b) Priority Function-IDs

#	Priority Function
0	Lexicographic ordering
1	Volume comparison
2	Manhattan distance comparison
3	Euclidean distance comparison

(c) Impact of different ε and priority functions

ε	PF	$\varnothing$km	$\lvert R\rvert$	$\varnothing t$ms	PQ Extr.	SD	Euc	Fr
0	0	4.077	741.31	5439.04	60265.8	-	-	-
0	1	4.077	741.31	6472.74	63487.8	-	-	-
0	2	4.077	741.31	4463.58	63649.0	-	-	-
0	3	4.077	741.31	3983.51	63497.3	-	-	-
1	0	4.063	41.87	22.03	32711.6	0.938	0.068	0.993
1	1	4.069	27.18	20.55	32926.4	0.930	0.077	0.991
1	2	4.073	18.66	20.62	32521.5	0.904	0.106	0.988
1	3	4.076	18.50	20.57	32505.1	0.903	0.104	0.982
2	0	4.056	19.00	18.98	31780.9	0.916	0.092	0.989
2	1	4.067	13.58	18.95	32089.0	0.908	0.104	0.988
2	2	4.073	8.82	19.18	31845.3	0.880	0.137	0.994
2	3	4.069	8.04	19.10	31801.1	0.873	0.136	0.963
3	0	4.040	8.54	18.36	31425.5	0.871	0.137	0.973
3	1	4.054	6.43	18.46	31677.5	0.863	0.148	0.979
3	2	4.062	3.87	18.59	31505.7	0.823	0.186	0.987
3	3	4.071	3.42	18.58	31478.2	0.810	0.190	0.935

Table 5.3: Evaluation of the heuristic that prevents small detours. The first line is repeated from Table 5.2c as reference. In addition to the abbreviations used in Table 5.2 we use D as the minimal length of a detour.

PF	D	∅km	$\|R\|$	$\varnothing t$ms	PQ Extr.	SD	Euc	Fr
3	0	4.077	741.31	3983.51	63497.3	-	-	-
0	3	4.055	12.73	19.96	32249.6	0.860	0.117	0.743
1	3	4.079	202.84	256.31	40981.7	0.981	0.022	0.871
2	3	4.080	191.23	189.47	40293.1	0.979	0.024	0.891
3	3	4.081	184.36	181.54	40150.4	0.977	0.025	0.878
0	6	4.040	9.83	19.47	32060.3	0.826	0.132	0.774
1	6	4.083	39.45	28.27	33868.0	0.927	0.088	0.768
2	6	4.083	32.22	25.61	33330.9	0.915	0.103	0.811
3	6	4.100	25.85	23.58	33068.2	0.903	0.105	0.749
0	9	4.026	9.17	19.47	32050.0	0.822	0.137	0.796
1	9	4.101	16.90	20.68	32708.3	0.867	0.148	0.685
2	9	4.090	10.85	20.27	32154.9	0.865	0.152	0.758
3	9	4.124	9.35	20.13	32104.7	0.772	0.218	0.901

Table 5.4: Evaluation of some combinations of these two heuristics

ε	PF	D	$\|R\|$	$\varnothing t$ms	PQ Extr.	SD	Euc	Fr
1	0	3	8.19	19.16	31860.2	0.938	0.073	0.993
1	2	3	16.70	20.51	32350.8	0.904	0.106	0.988
3	0	3	4.86	18.48	31463.3	0.869	0.133	0.973
3	2	3	3.77	18.65	31494.5	0.823	0.186	0.987
3	2	9	2.92	18.56	31460.6	0.810	0.201	0.967

and 3.81 ms we concluded that the five Dijkstra searches involved took about 14 ms. With this heuristic it seems difficult to get as close as with the ε-dominance as increasing the length of the prevented detours not only decreases the amount of found routes but also increases the complexity at each step.

5.3.4 Combined heuristics

As seen in Table 5.4 small ε combined with small detours gives surprisingly few, high quality results. No other tried combination resulted in about 8 routes, that were as close to the Pareto set. With an increase ε and preventing longer detours, the difference to just using a larger ε diminishes.

Table 5.5: Evaluation of ε on a larger graph: The average route length was 78.6 km and the reverse Dijkstra searches took about 2 25s

ε	PF	$\lvert R\rvert$	$\varnothing t$ms	PQ Extr.
2	1	911.79	83216.22	6886112.5
2	2	253.14	12504.02	4711110.7
2	3	189.84	8525.13	4220816.4
3	0	144.51	6832.23	4004832.0
3	1	90.75	3881.39	3460879.7
3	2	22.10	2944.09	3134252.7
3	3	14.85	2748.76	3062156.8

5.4 Performance on a large graph

In this section we use the whole graph ("Regierungsbezirk Karlsruhe") to evaluate how well the different heuristics perform. We also restrict ourselves to just evaluate the ε-dominance with different values combined with different priority functions.

The search, which just uses the distance as sole criterion on this graph took 480ms on average and used 581391 priority queue extractions. Therefore we estimate that the five reverse Dijkstra searches take about 2.25s. As seen in Table 5.5 a large ε keeps both the query time as well as the number of resulting routes low.

With this ε and this graph sizes the Dijkstra searches often took a large fraction of the whole query time, especially for short queries and large ε.

We also took a look at the 15 routes between 100 and 120 km. As shown in Figure 5.4 there we could see a rather large variance in the number of routes: With the lexicographic order and the largest ε we got between 52 and 449 routes. Using the Manhattan distance reduced this a range between 9 and 84 routes, but only two sets had more than 40 routes. But not just the variance, but also the median was much smaller than with the lexicographic order. The ordering based on the volume was in between those two. The smallest set contained 68 routes and the largest set 322.

Perhaps even more interesting is the fact that we did not get an outlier with the Euclidean distance and we found at most 40 routes, which took 4 2s. This is a rather nice property as the user might not expect, that query with similar distances vary that much.

5.5 Case Studies

As the Tables in the previous section do not illustrate well how the routes look like, we present some images, which show the Pareto set (for short routes) as well as the filtered set. Unfortunately even on the shortest route the detours are usually too short to be identified individually.

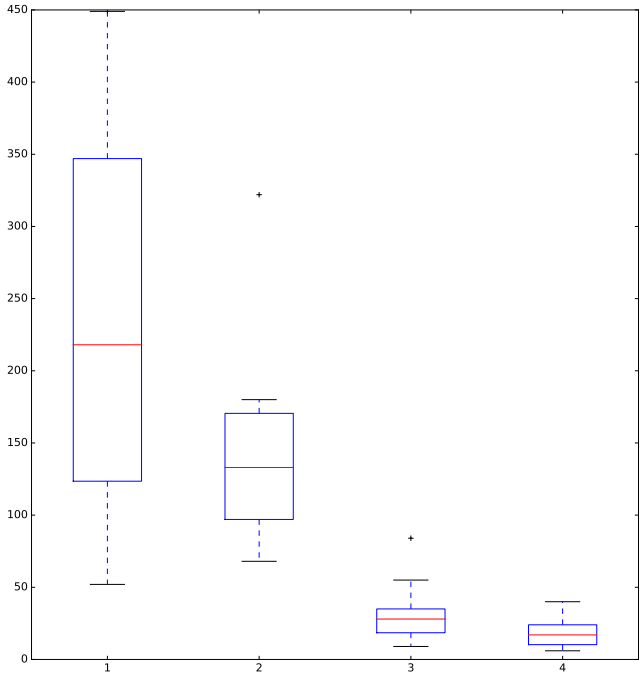


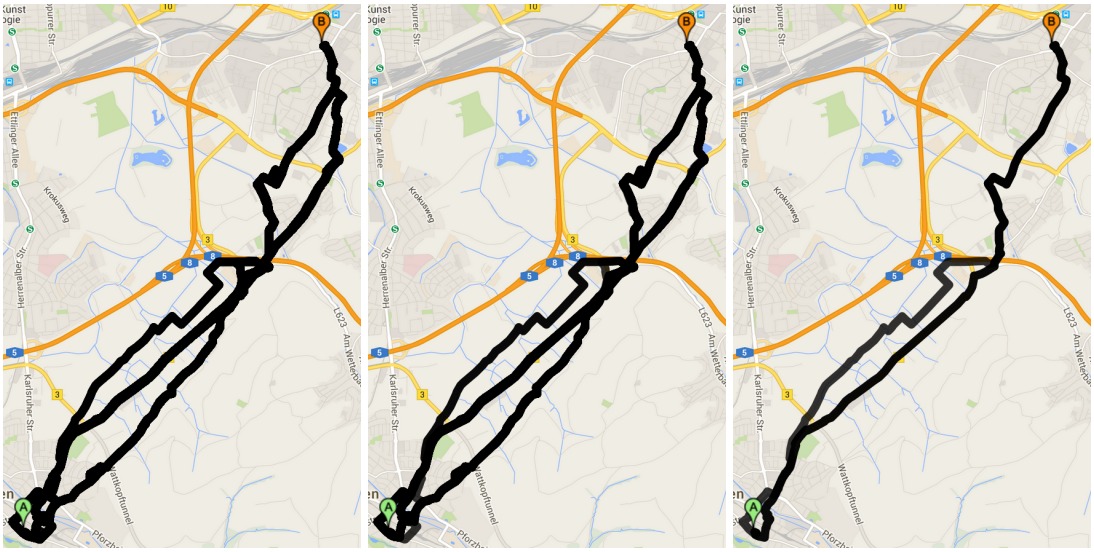
Figure 5.4: Set sizes if the average route length was between 100 and 120 km to show the diversity of the results: From left to right lexicographic ordering, ordering based on the volume, the Manhattan and the Euclidean distance are shown. The red line shows the median value, the boxes show the upper and lower quartile and the whiskers show the minimum and maximum values that are inside $1.5 \times (\text{upper quartile} - \text{lower quartile})$.

The case studies were executed on a Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz with 8 GB RAM. The program was compiled with an gcc-5.2 with the compiler flags "-O3 -flto -march=native" on Arch Linux.

The case studies use the graph "Regierungsbezirk Karlsruhe" in which the reverse Dijkstra searches take between 1.7s and 1.8s on this machine.

We have chosen one rather short query, so that we can still compute the Pareto set and get an impression on such a result. With this query even with an small ε the single criterion Dijkstra searches took longer than the later A* search.

As second query we have chosen a rather long trip. It was chosen to see the routes, that we get if we use a large ε. As computing the Pareto set in the second case would probably take days or even weeks, we cannot compare the filtered and optimal set in this case.



(a) Pareto set (b) ε-ID 2,Volume (c) ε-ID 3,Euclidean

Figure 5.5: Three different settings: The Pareto set on the left took 9.1s to compute and contains 5204 routes. The image in the middle shows 69 routes which are visually indistinguishable and took only 2.0s to compute. The image on the right shows 3 routes and took 2.1s.

5.5.1 From Ettlingen to Durlach

The first case we study is a rather short route from Ettlingen to Durlach. It is about 11km long. Therefore computing the Pareto set only took about 9.1s. Three route sets are shown in Figure 5.5.

As expected (see also Section 5.3.3) using a lexicographic order in our priority queue together with preventing detours of a length 3 and smaller led to a huge drop in the number of routes and the running time. This resulted in 50 routes which were found in only 2.1s. Preventing detours up to a length 9 did not improve the query time (as expected), but led to a further reduction of the number of routes down to 14.

In this example we can see that the Pareto set itself does not necessarily only contains good routes. On the other hand we might probably still be more interested in routes that an optimal in some way. While we cannot guarantee, that our results are in the Pareto set, Figure 5.5b shows that even a small fraction of this set can look like it.

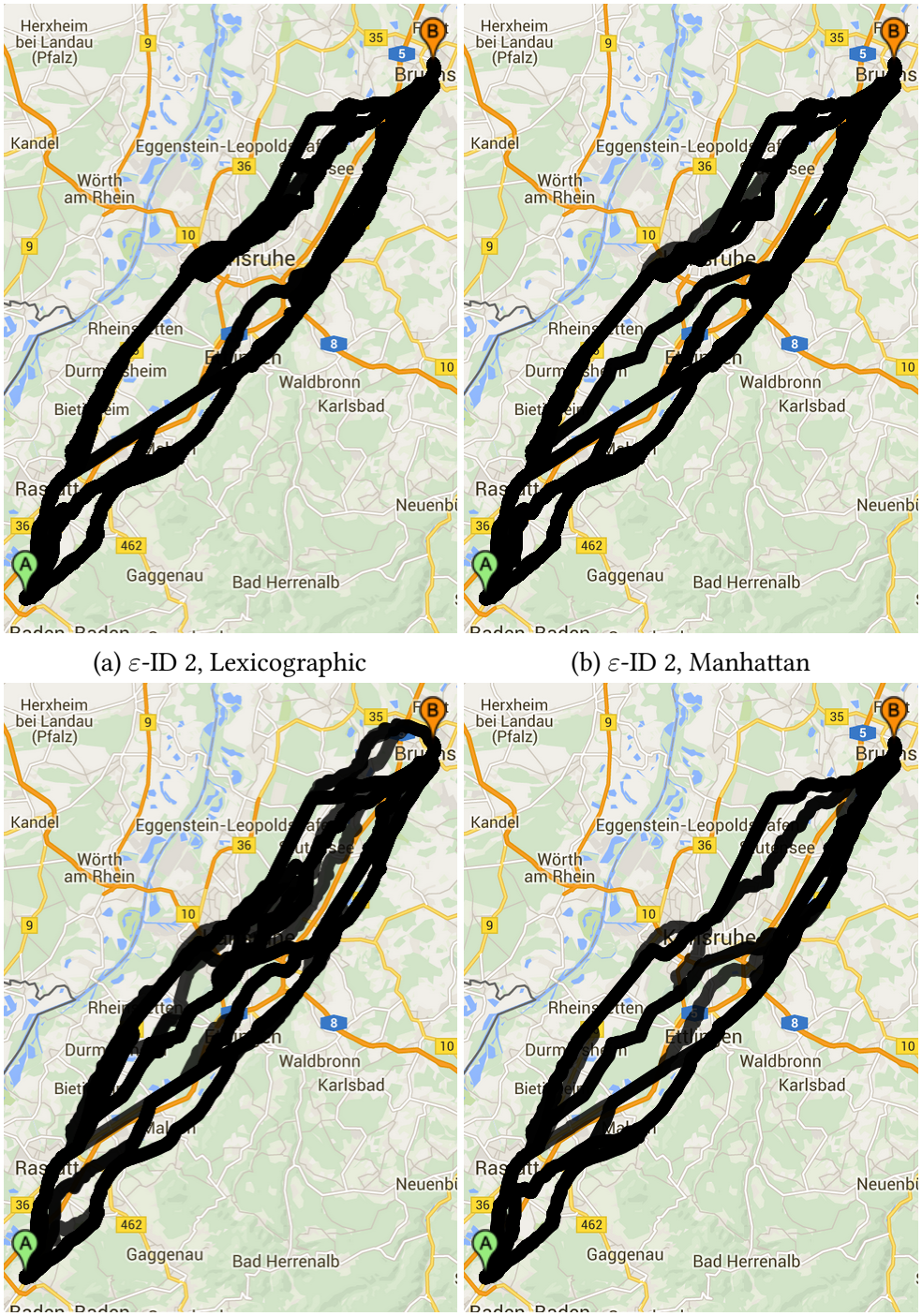
5.5.2 From Baden-Baden to Bruchsal

The next query we look at is from Baden-Baden to Bruchsal. The routes are about 66km long and the first query we tried used the ε with the ID 2. As already described in Figure 5.6 this led to more than 1000 routes and took 43s. Just changing the priority function to the volume of the n-orthotope reduces the number of routes to 600 and the running time to just 11s. With the Manhattan and the Euclidean distance this was further reduced to 256 respective 221 routes. The running time was 6.4s resp. 5.8s. In Figure 5.6c the disadvantages of preventing short detours can be seen: We get many routes that are caused by the effect described in Section 4.3.2.

The Figure 5.6d shows the result of choosing a large ε together with using the Euclidean distance as priority function. We got the fewest routes and it still looks like we have more alternatives than in Figure 5.6a.

5.5.3 Case Study Conclusions

As seen on Figures 5.5 and 5.6 a large proportion of the routes in the results is very similar to each other. Even with a large ε, most routes still consists of different combinations of a rather small set of route segments. To prevent this we would probably need an additional filter.



(a) ε-ID 2, Lexicographic

(b) ε-ID 2, Manhattan

(c) ε-ID 3,Lexicographic, Detours>9

(d) ε-ID 3,Euclidean

Figure 5.6: Four different settings for a rather long route. The first set contains 1276 routes and took 43s. The second contains 256 routes and took 6.4s. The third has 59 routes and took 4.7s. It also looks like it contains many more routes than the other three sets. The last set contains 21 routes and only took 2.4s.

6 Conclusion

In this thesis we examined a way to find attractive routes for cyclists using multiple criteria. We surveyed several existing paper on which criteria are important to cyclists and identified several, that can be gathered from OpenStreetMap.

Afterwards we used a A*-based Algorithm to find all Pareto-optimal routes with prior reverse Dijkstra searches to get a good potential. Then analysed the interaction of the different criteria we used and which combinations led to a large increase in the number of resulting routes. We concluded that the combination of Safety and Travel Time is worst in this regard. Often we got more routes with these two criteria than with four criteria, if one of those two was ignored.

As the set of Pareto optimal routes can become very large even for rather short routes (especially inside cities) some heuristic filtering is required to keep the running time low as well as reduce the resulting routes to a manageable amount. Out of the two filtering techniques we have tried, ε-dominance was the one, which gave better results per speedup. With the introduction of ε-dominance we found, that the choice of the priority function has a big impact on the resulting route set. After evaluating different ε-sizes and different priority functions, we conclude, that we get the least amount of routes if we use the euclidean distance as priority function together with an large ε. With this ε we often found the reverse Dijkstra searches, we used to get the potential for the A* search, to take a large part of the query time. Unfortunately we could not find a reliable correlation between the length of an route, the chosen ε and the number of resulting routes, i.e. with the same route length and the same ε there are still large differences in the number of routes found.

Using the Manhattan or Euclidean distance as priority we got a reduced running time even if the whole Pareto set of routes was searched, if the resulting set was large. That lets us conclude, that these priority functions scale better with the size of the resulting set. With ε-dominance this effect was amplified, as these priority functions also led to a larger decrease of the number of routes found.

Using lexicographic ordering, we got results that are slightly biased towards the first criterion, if ε-dominance is used. As the A* search always extracts the label with a minimal value in this criterion, it finds the best route, that is not already ε-dominated, with regard to this criterion as the next route. With the other priority functions we evaluated, we got less routes and they are not biased towards a particular criterion. Comparing the other three priority functions we got fewer results that are good in a single criterion using the euclidean distance compared to using the Manhattan distance or the volume. On the other hand we get most of those with the volume as priority function.

6.1 Future Work

With a large ε, which is required to keep the number of computed routes to a manageable size, the single-criterion Dijkstra Searches account for a large part of our query time. This especially the case if we if the graph contains many more edges than those required for the A* Search and the result contains few routes. It might be possible the prune the Dijkstra Searches, if it can be guaranteed that the additional nodes searched cannot be part of the later multi-criteria search. To do this we could stop the Dijkstra searches after we reached the source node, calculate the label which dominates all labels found in the individual Dijkstra searches and than continue these searches until we are dominated by this label in our individual searches. Additionally, to further improve the query time, it should be possible to use a speedup technique (e.g. PHAST [7]) to reduce the time required for the computation of the potential.

If the maximal values found in the Dijkstra searches are used as well, we might better estimate the size of the hypersurface, which includes the solutions. This estimation might allow us to choose a better ε, so the sizes of the result set vary less. Combined with improvements to the Dijkstra searches mentioned in the previous paragraph, this should keep the amount of routes found small and enable much shorter running times.

Another simple approach to improve the quality of the results might be, to start with a large ε and decrease it if the found routes are not satisfactory.

Sometimes a bias towards a criterion might be wanted and this could be achieved by weighting the priority functions or using this as first criterion with lexicographic ordering.

Bibliography

[1] Hannah Bast et al. "Route Planning in Transportation Networks". In: *CoRR* abs/1504.05140 (2015). URL: http://arxiv.org/abs/1504.05140.

[2] Moritz Baum et al. "Speed-Consumption Tradeoff for Electric Vehicle Route Planning". In: *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*. Ed. by Stefan Funke and Matúš Mihalák. Vol. 42. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 138–151. ISBN: 9783939897750. DOI: http://dx.doi.org/10.4230/OASIcs.ATMOS.2014.138. URL: http://drops.dagstuhl.de/opus/volltexte/2014/4758.

[3] Joseph Broach, Jennifer Dill, and John Gliebe. "Where do cyclists ride? A route choice model developed with revealed preference {GPS} data". In: *Transportation Research Part A: Policy and Practice* 46.10 (2012), pp. 1730–1740. ISSN: 0965-8564. DOI: http://dx.doi.org/10.1016/j.tra.2012.07.005. URL: http://www.sciencedirect.com/science/article/pii/S0965856412001164.

[4] Community. *OpenStreetMap*. URL: http://www.openstreetmap.org.

[5] Brian C. Dean. *Continuous-Time Dynamic Shortest Path Algorithms*. 1999.

[6] Daniel Delling and Dorothea Wagner. "Pareto Paths with SHARC". English. In: *Experimental Algorithms*. Ed. by Jan Vahrenhold. Vol. 5526. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 125–136. ISBN: 9783642020100. DOI: 10.1007/978-3-642-02011-7_13. URL: http://dx.doi.org/10.1007/978-3-642-02011-7_13.

[7] Daniel Delling et al. "PHAST: Hardware-Accelerated Shortest Path Trees". In: *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*. IPDPS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 921–931. ISBN: 9780769543857. DOI: 10.1109/IPDPS.2011.89. URL: http://dx.doi.org/10.1109/IPDPS.2011.89.

[8] Lee R. Dice. "Measures of the Amount of Ecologic Association Between Species". In: *Ecology* 26.3 (1945), pp. 297–302. ISSN: 00129658. URL: http://www.jstor.org/stable/1932409.

[9] E.W. Dijkstra. "A note on two problems in connexion with graphs". English. In: *Numerische Mathematik* 1.1 (1959), pp. 269–271. ISSN: 0029-599X. DOI: 10.1007/BF01386390. URL: http://dx.doi.org/10.1007/BF01386390.

[10] Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. "Multi-criteria Shortest Paths in Time-Dependent Train Networks". English. In: *Experimental Algorithms*. Ed. by CatherineC. McGeoch. Vol. 5038. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 347–361. DOI: 10.1007/978-3-540-68552-4_26. URL: http://dx.doi.org/10.1007/978-3-540-68552-4_26.

[11] Carlos Dora and Michelle Phillips. *Transport, environment and health*. Tech. rep. WHO Regional Office for Europe, 2000.

[12] Tom G. Farr et al. "The Shuttle Radar Topography Mission". In: *Reviews of Geophysics* 45.2 (2007). RG2004, n/a–n/a. ISSN: 1944-9208. DOI: 10.1029/2005RG000183. URL: http://dx.doi.org/10.1029/2005RG000183.

[13] Michael L. Fredman and R.E. Tarjan. "Fibonacci Heaps And Their Uses In Improved Network Optimization Algorithms". In: *Foundations of Computer Science, 1984. 25th Annual Symposium on*. Oct. 1984, pp. 338–346. DOI: 10.1109/SFCS.1984.715934.

[14] Stefan Funke and Sabine Storandt. "Polynomial-time Construction of Contraction Hierarchies for Multi-criteria Objectives". In: *Proceedings of the Meeting on Algorithm Engineering & Expermiments*. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2013, pp. 41–54. URL: http://dl.acm.org/citation.cfm?id=2790158.2790162.

[15] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979. ISBN: 0716710447.

[16] Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. "Route Planning with Flexible Objective Functions". In: *IN ALENEX*. SIAM, 2010, pp. 124–137.

[17] Robert Geisberger et al. "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks". In: *Proceedings of the 7th International Conference on Experimental Algorithms*. WEA'08. Provincetown, MA, USA: Springer-Verlag, 2008, pp. 319–333. URL: http://dl.acm.org/citation.cfm?id=1788888.1788912.

[18] Pierre Hansen. "Bicriterion Path Problems". English. In: *Multiple Criteria Decision Making Theory and Application*. Ed. by Günter Fandel and Tomas Gal. Vol. 177. Lecture Notes in Economics and Mathematical Systems. Springer Berlin Heidelberg, 1980, pp. 109–127. ISBN: 9783540099635. DOI: 10.1007/978-3-642-48782-8_9. URL: http://dx.doi.org/10.1007/978-3-642-48782-8_9.

[19] P.E. Hart, N.J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (July 1968), pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[20] E. Heinen, B. van Wee, and K. Maat. "Commuting by bicycle: An overview of the literature". In: *Transport Reviews* 30.1 (2010). cited By 106, pp. 59–96. DOI: 10.1080/01441640903187001. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-71149111755&partnerID=40&md5=494b54d29f1c1c7479d844b4506bb65b.

[21] David Hounam and Marian Werner. *The Shuttle Radar Topography Mission (SRTM).* 1999.

[22] Jan Hrncir et al. "Bicycle Route Planning with Route Choice Preferences". In: (2014).

[23] Jan Hrncir et al. "Speedups for Multi-Criteria Urban Bicycle Routing". In: *15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2015).* Ed. by Giuseppe F. Italiano and Marie Schmidt. Vol. 48. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 16–28. ISBN: 9783939897996. DOI: http://dx.doi.org/10.4230/OASIcs.ATMOS.2015.16. URL: http://drops.dagstuhl.de/opus/volltexte/2015/5458.

[24] P. Loridan. "ε-solutions in vector minimization problems". English. In: *Journal of Optimization Theory and Applications* 43.2 (1984), pp. 265–276. ISSN: 0022-3239. DOI: 10.1007/BF00936165. URL: http://dx.doi.org/10.1007/BF00936165.

[25] E. Machuca and L. Mandow. "Multiobjective heuristic search in road maps". In: *Expert Systems with Applications* 39.7 (2012), pp. 6435–6445. ISSN: 0957-4174. DOI: http://dx.doi.org/10.1016/j.eswa.2011.12.022. URL: http://www.sciencedirect.com/science/article/pii/S0957417411016939.

[26] Ernesto Queiros Vieira Martins. "On a multicriteria shortest path problem". In: *European Journal of Operational Research* 16.2 (1984), pp. 236–245. URL: http://EconPapers.repec.org/RePEc:eee:ejores:v:16:y:1984:i:2:p:236-245.

[27] G. Menghini et al. "Route choice of cyclists in Zurich". In: *Transportation Research Part A: Policy and Practice* 44.9 (2010), pp. 754–765. ISSN: 0965-8564. DOI: http://dx.doi.org/10.1016/j.tra.2010.07.008. URL: http://www.sciencedirect.com/science/article/pii/S0965856410001187.

[28] National Imagery and Mapping Agency. *Department of Defense World Geodetic System 1984: its definition and relationships with local geodetic systems.* Tech. rep. TR8350.2. St. Louis, MO, USA: National Imagery and Mapping Agency, Jan. 2000. URL: http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html.

[29] C. H. Papadimitriou and M. Yannakakis. "On the Approximability of Trade-offs and Optimal Access of Web Sources". In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science.* FOCS '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 86–. ISBN: 0769508502. URL: http://dl.acm.org/citation.cfm?id=795666.796569.

[30] Andrea Raith and Matthias Ehrgott. "A Comparison of Solution Strategies for Biobjective Shortest Path Problems". In: *Comput. Oper. Res.* 36.4 (Apr. 2009), pp. 1299–1331. ISSN: 0305-0548. DOI: 10.1016/j.cor.2008.02.002. URL: http://dx.doi.org/10.1016/j.cor.2008.02.002.

[31] Antonio Sedeño-Noda and Andrea Raith. "A Dijkstra-like Method Computing All Extreme Supported Non-dominated Solutions of the Biobjective Shortest Path Problem". In: *Comput. Oper. Res.* 57.C (May 2015), pp. 83–94. ISSN: 0305-0548. DOI: 10.1016/j.cor.2014.11.010. URL: http://dx.doi.org/10.1016/j.cor.2014.11.010.

[32] M. Shimrat. "Algorithm 112: Position of Point Relative to Polygon". In: *Commun. ACM* 5.8 (Aug. 1962), pp. 434–. ISSN: 0001-0782. DOI: 10.1145/368637.368653. URL: http://doi.acm.org/10.1145/368637.368653.

[33] Qing Song et al. "Exploring Pareto Routes in Multi-Criteria Urban Bicycle Routing". In: *IEEE Intelligent Transportation Systems Conference*. 2014.

[34] T. Sørenson. *A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species Content and Its Application to Analyses of the Vegetation on Danish Commons*. Biologiske Skrifter // Det Kongelige Danske Videnskabernes Selskab. I kommission hos E. Munksgaard, 1948. URL: https://books.google.co.in/books?id=rpS8GAAACAAJ.

[35] Sabine Storandt. "Route Planning for Bicycles - Exact Constrained Shortest Paths Made Practical via Contraction Hierarchy." In: *ICAPS*. Ed. by Lee McCluskey et al. AAAI, 2012. ISBN: 9781577355625. URL: http://dblp.uni-trier.de/db/conf/aips/icaps2012.html#Storandt12.

[36] Robert Tarjan. "Depth first search and linear graph algorithms". In: *SIAM Journal on Computing* (1972).

[37] Kay Teschke et al. "Route Infrastructure and the Risk of Injuries to Bicyclists: A Case-Crossover Study". In: *American Journal of Public Health* 102.12 (Dec. 2012), pp. 2336–2343. ISSN: 0090-0036. DOI: 10.2105/AJPH.2012.300762. URL: http://dx.doi.org/10.2105/AJPH.2012.300762.

[38] George Tsaggouris and Christos Zaroliagis. "Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-linear Objectives with Applications". In: *Proceedings of the 17th International Conference on Algorithms and Computation*. ISAAC'06. Kolkata, India: Springer-Verlag, 2006, pp. 389–398. ISBN: 9783540496946. DOI: 10.1007/11940128_40. URL: http://dx.doi.org/10.1007/11940128_40.

[39] Chi Tung Tung and Kim Lin Chew. "A multicriteria Pareto-optimal path algorithm". In: *European Journal of Operational Research* 62.2 (1992), pp. 203–209. URL: http://EconPapers.repec.org/RePEc:eee:ejores:v:62:y:1992:i:2:p:203-209.

[40] T. Vincenty. "Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations". In: *Survey Review* 22.176 (1975), pp. 88–93.

[41] D.J. White. "Epsilon efficiency". English. In: *Journal of Optimization Theory and Applications* 49.2 (1986), pp. 319–337. ISSN: 0022-3239. DOI: 10.1007/BF00940762. URL: http://dx.doi.org/10.1007/BF00940762.

[42] Meghan Winters et al. “Motivators and deterrents of bicycling: comparing influences on decisions to ride”. English. In: *Transportation* 38.1 (2011), pp. 153–168. ISSN: 0049-4488. DOI: 10.1007/s11116-010-9284-y. URL: http://dx.doi.org/10.1007/s11116-010-9284-y.