Karlsruhe Institute of Technology

A Unified Framework for Electric Vehicle Routing

Master Thesis of

Patrick Niklaus

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Prof. Dr. Dorothea Wagner
	Prof. Dr. Peter Sanders
Advisors:	Moritz Baum, M.Sc.
	Tobias Zündorf, M.Sc.

Time Period: 1st June 2017 – 30th November 2017

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, November 30, 2017

Abstract

Route planning is an integral part of our daily lives and industries that enables us to navigate unknown cities, as well as making sure that goods arrive cost-effective and on time. Because we will see a transition from fossil fuel based vehicles to electric vehicles in the coming years, we want to update the models we use for route planning to this development. There are several aspects that make electric vehicles different from fossil fuel based vehicles. A limited number of charging stations, small battery capacities and long charging times make it necessary to incorporate a consumption model when computing optimal travel times. The consumption model of an electric vehicle is also different from a fossil fuel based vehicle, since we need to account for effects like non-linear charging rates of Lithium-Ion batteries and energy recuperation. In this thesis we unify an accurate consumption model that incorporates the effects of driving slower on the consumption, with a realistic charging model for batteries. We propose an algorithm to compute exact solutions and augment it with an A^*-based speedup technique to enable computing solutions on country-sized datasets in a sensible time. Based on that we also propose and evaluate multiple heuristics to further improve the query time. We evaluate the proposed algorithm, heuristics and computed results. Based on these insights, we propose some further ideas on how to improve our algorithm.

Deutsche Zusammenfassung

Routenplanung ist ein essentieller Bestandteil unseres täglichen Lebens und vieler Industrien. Zum einen ermöglicht sie uns in unbekannten Städten zu navigieren, zum anderen ist sie eine wichtige Komponente um sicher zu stellen, das Güter kostengünstig und zur richtigen Zeit am Ziel ankommen. In den nächsten Jahren werden wir einen Wandel von Verbrennungsmotoren zu elektrischen Motoren sehen, weshalb es wichtig ist bestehende Modelle zur Routenplanung dahingehend zu aktualisieren. Es gibt verschiedene Aspekte, die elektrisch betriebene Fahrzeuge anders machen als Fahrzeuge mit Verbrennungsmotoren. Eine begrenzte Anzahl an Ladestationen, geringe Batteriekapazitäten und lange Ladezeiten erfordern es ein Verbrauchsmodell zum Berechnen von optimalen Reisezeiten zu benutzen. Das Verbrauchsmodell eines elektrischen Fahrzeugs unterscheidet sich signifikant von einem Fahrzeug mit Verbrennungsmotor, denn es müssen Effekte wie nicht-lineare Ladezeiten von Lithium-Ion-Batterien und Bremsenergie-Rückgewinnung berücksichtigt werden. In dieser Abschlussarbeit vereinen wir ein realistisches Verbrauchsmodell, das z.B. das Verringern der Geschwindigkeit zum Energie sparen berücksichtigt, mit einem realistisches Lademodell für elektrische Batterien. Wir schlagen einen Algorithmus zur exakten Berechnungen von Pfaden mit optimaler Reisezeit in diesem Modell vor. Anschließend beschleunigen wir die Berechnung mit Hilfe einer A^* basierten Methode um exakte Lösungen auf Straßennetzwerken ganzer Länder in annehmbarer Zeit zu berechnen. Um die Berechnungszeit weiter zu senken, schlagen wir außerdem einige Heuristiken vor. Wir evaluieren den hier vorgeschlagenen Algorithmus, Heuristiken und damit berechneten Resultate. Auf Grundlage dieser Evaluierung schlagen wir weitere Idee zur Verbesserung unseres Algorithmus vor.

Contents

1. Introduction

Navigation applications are ubiquitous to our every day life. Planning routes based on travel time is the most important building block of the logistics industry as we know it today. In the following decades we will see a migration away from fossil fuels to renewable energies, which will have a large impact of how we travel and how we plan transportation. An important part of this change will be the use of electric vehicles, for personal transportation as well as for the transportation of goods and cargo. Countries like Germany and Norway already have strong incentives to buy electric vehicles. PricewaterhouseCoopers estimates that by 2030 at least 30% of car sales will be electric vehicles [PwC16]. It is thus critical to update and validate the models that were traditionally used for computing paths with minimal travel time. Electric vehicle technology has some properties that make it very different from fossil fuel based technology. The central assumption for fossil fuel based vehicles is that the energy consumption does not matter for the minimal arrival time. We will always drive at the fastest possible speed and if we run out of fuel, refueling will only take a short amount of time. These assumptions are valid since the typical range of a fossil fuel based car is very large, much above the typical length of a route, but also because there is a dense network for gas stations that allow fast refueling. While electric vehicle manufactures are working on addressing these issues, small battery capacities and long recharging times are still an important factor that influences the travel time. Thus, a key difference between travel time minimal paths for fossil fuel based vehicles and electric vehicles is the need for an accurate consumption model. When it comes to consumption models for electric vehicles, the most important distinction to the fossil fuel based models is the fact that electric vehicles can regain energy since an electric motor can also act as a generator. This means that in cases like braking or driving downhill, there will not only be no energy used, but the battery will be recharged. Additionally, just like with fossil fuel based vehicles, we have the option of reducing the consumption by driving more slowly. When we exceed the range, even while driving economically, we will need to stop and charge the battery to continue the travel. Filling up the tank for fossil fuel based vehicles is generally quick and only takes a linear amount of time, we can fill up a tank of twice the size in twice the time. For electric vehicles recharging the battery usually takes much longer, and due to constraints of the battery technology is not linear, but has diminishing returns. Charging the battery from a capacity of 60% to 80% will be much faster than charging from 80% to 100%. All of these effects gain importance if we need to use the full battery capacity, which is primarily the case for long distance routes. Traditional algorithms for travel time minimal paths are optimized for computing long distance routes, hence there is a need for incorporating the consumption requirements.

1.1 Related Work

There are several techniques for computing shortest paths without incorporating consumption models. The classic method of finding shortest travel time based paths is *Dijkstra's algorithm* [Dij59] published in 1959. While this algorithm performs well on small or medium sized networks, it is too slow on much larger country or continental sized datasets. To address this issue, multiple approaches have been proposed, most using some form of preprocessing of the road network.

One of the fastest and arguably most popular approaches is *Contraction Hierarchies* [GSSV12]. This approach assigns each node a priority and then inserts shortcuts to keep the invariant that a shortest path can always be found by visiting nodes of strictly increasing priority follwed by nodes of strictly decreasing priority. One requirement for this approach to work effectively is that travel time optimal paths can be computed a priori, meaning without knowing the actual parameters of the query. This requirement is violated when applying this method to models where the travel time optimal path depends on some external parameters. In the case of electric vehicles it is easy to see that the travel time optimal path heavily depends on the state of charge of the battery: Any given sub-path might need to take a detour over a charging station to refuel the battery. There are several approaches that try to integrate the concept of *Contraction Hierarchies* with non-constant travel times. The canonical application for this is the use of traffic data, where the travel time heavily relies on the time of day. Batz et al. [BDSV09] propose a model for generalizing *Contraction Hierarchies* for this case.

In contrast to *Contraction Hierarchies* there are other methods that do not change the structure of the graph but instead use heuristics to guide the search of *Dijkstra's algorithm*. The most well known of these approaches is the A^* algorithm [HNR68] published in 1968. While *Dijkstra's algorithm* only uses the distance to the source node to determine which node to process next, A^* also tries to estimate the distance to the destination node and uses this to estimate the total travel time over a given node. As long as this estimate remains below the actual distance, correctness of the algorithm can be ensured. Since this approach does generally not require excessive preprocessing, it is especially useful if we want to incorporate information that is only available at query time. Another useful property is that we only need to provide a heuristic for the travel time to the destination, which makes it easy to use shortest paths in simplified models as heuristic for the exact search. The simplest case of this would be to disregard fuel consumption and use the travel time based on this model as heuristic.

There have been several attempts of computing both paths that minimize energy consumption and paths that minimize travel time for electric vehicles before. Each approach only incorporates a part of the model that is proposed in this thesis. In the case of energy minimal paths, Sachenbacher et al. [SLAH11] focus on energy-optimal path using *Dijkstra's algorithm* with a shifted node potential to make it label setting on negative edge weights. Hartmann and Funke [HF14] focus on computing the energy-optimal path within a given bound on the maximal travel time. They incorporate the notion of consumption-time trade-offs by computing multiple constrained energy-optimal paths for every discrete maximal speed and return the best solution. Since Hartmann and Funke primarily focus on energy efficient paths for fossil fuel based vehicles, one notable difference is the lack of recuperation and battery constraints in their work. Baum et al. [BDHS$^+$14] explored using a multi-criteria version of *Dijkstra's algorithm* on a multi-graph with edge weights that represented several time-consumption trade-offs. While this publication focuses on computing all solutions that offer interesting trade-offs, it is easy to adapt to only compute the fastest feasible solution. Goodrich et al. [GP14] use a two-phase algorithm of first optimizing for travel time and then for energy consumption, including modeling charging

stations. However, their approach assumes that the battery is always charged to full capacity at charging stations and does not include time-consumption trade-offs. Merting et al. [MSS15] give a good theoretical overview of including charging stations in the shortest path problem, but the proposed approximation algorithms are not shown to be practical on large road networks. Storandt [Sto12] incorporates charging stations with constant edge weights and proposes algorithms for finding both the shortest feasible path and the energy-optimal path. However their model assumes that the battery is always fully charged at each charging stop.

There are also several works that adapt Contraction Hierarchies to graph models for electric vehicles. Eisener et al. [EFS11] demonstrate how to adapt Contraction Hierarchies for computing energy-optimal paths with battery constraints. Storandt [Sto12] additionally incorporates charging stations into the approach from [EFS11]. Baum et al. [BDG$^+$15] apply the model of using charging stations to Contraction Hierarchies, but don't allow for using trade-offs, such as driving more slowly, between charging stations. This is achieved by not contacting the charging stations, keeping a small unprocessed core graph that represents most long distance routes in a graph. In a publication from 2017, Baum et al. [BDWZ17] unify the model of time-consumption trade-offs using a realistic consumption model based on hyperbolic functions and Contraction Hierarchies. They do so by computing trade-off profiles between nodes and ensuring that no shortcut path can violate battery constraints.

In many cases even the most realistic model will have inaccuracies, since external factors such as the state of the vehicle, or the driver behavior will influence the travel time. Hence, it is often sufficient to compute paths that are *"fast enough"*. Interestingly, replacing the goal of optimality with a more pragmatic goal of being *"good enough"* can lead to dramatic speedups. For example for the BSP algorithm [BDHS$^+$14] a heuristic method to discard solutions that offer similar trade-offs yielded speedups by several orders of magnitudes, Hartmann and Funke [HF14] and Baum et al. [BDG$^+$15] [BDWZ17] all propose similar heuristics based on discarding unimportant solutions. Apart from algorithmic heuristics we can also consider simplifications of the graph or consumption model to potentially speed up the query. All approaches presented so far use a simplified model in some aspects. Thus, they are heuristics in a more exact model, that we can evaluate.

1.2 Contribution

In this thesis we make the following contributions to the state of art:

- We propose a model that for the first time integrates realistic time-consumption trade-offs with a realistic charging function model that allows for partially charging the battery. We base our model on the time-consumption model proposed by Baum et al. [BDWZ17] and our charging model based on the work of Baum et al. [BDG$^+$15]. The model allows for a concise representation of charging functions as part of the time-consumption trade-offs.
- We propose a model for computing realistic time and consumption turn costs based on speed changes during a turn maneuver. We consider the effects of energy recuperation when slowing down, the effects of additional energy usage when accelerating and the impact of the speed at which the turn maneuver is executed. The resulting model can be used as a baseline to determine the error of simplified non-turn cost aware models.
- We propose an algorithm that can compute exact shortest paths for electric vehicles with our realistic time-consumption and charging models on country-sized datasets in a reasonable time. We base our work on the *Function Propagation* algorithm proposed in [BDWZ17] and adapt it to our model that includes charging functions.

We apply the A^* algorithm on this extended algorithm and propose and evaluate multiple potential functions.

- We propose several heuristics on how to integrate the realistic turn cost model into the aforementioned algorithm. Incorporating the exact model in our algorithm does not seem feasible since it requires solving a complex optimization problem.
- We propose several heuristics both for our model and our algorithm to speed up the query by several orders of magnitude while still yielding good quality.

This thesis is the base for a unified framework for electric vehicle routing and can serve as exact baseline for simplified models and algorithms. The source code of the software developed for this thesis is publicly available at `github.com/TheMarex/charge` under a permissive software license and we hope it provides a good starting point for further research on the topic.

1.3 Overview

Chapter 1

We give a brief overview of the problem space and realted work.

Chapter 2

We give a brief overview of some mathematical definitions that we use throughout this thesis. In the course of this, we formally define multiple variations of the *Shortest Path* and *Constrained Shortest Path* problem.

Chapter 3

Based on mathematical foundation of Chapter 2 we give a formal definition of the *Electric Vehicle Shortest Path* problem that we try to solve with the algorithms proposed in this thesis. After that we introduce the consumption model based on the time-consumption trade-off function from Baum et al. [BDWZ17] and the charging model based on another paper by Baum et al. [BDG$^+$15]. Finally we try to define turn costs for a graph that uses trade-off functions.

Chapter 4

We define our baseline algorithm based on the function propagation algorithm proposed by Baum et al. in [BDWZ17]. We define a novel operation of composing time-consumption trade-off functions with charging functions by finding optimal points to start charging. In the course of this chapter we proof the optimality of these operations. We extend these operations to not only work on simple trade-off functions but also piecewise trade-off functions.

Chapter 5

We take our base algorithm from Chapter 4 and apply multiple A^* based speedup techniques to it. For this purpose we generalize the Ω-potential proposed by Baum et al. in [BDG$^+$15] to our model that includes trade-off functions. We prove its correctness on time-consumption trade-off function based labels and provide some intuitions on how it works. We also present the potential function based on the shortest unconstrained travel time to the target.

Chapter 6

We consider several heuristics, both for the algorithm and for simplifying the graph model. The algorithmic heuristics are based on employing ϵ-dominance to discard unlikely solutions. We also consider simplifying the graph model by, for example, replacing the hyperbolic trade-off functions with linear functions, limiting the range of trade-offs for some edges or only allowing trade-offs on specific edges. Additionally, we explore heuristics for reducing the set of charging stations.

Chapter 7

We evaluate our algorithm against an adapted version of the BSP algorithm [BDHS$^+$14]. We use datasets from multiple sources, both PTV and OpenStreetMap, and charging stations from ChargeMap and Tesla.

Chapter 8

We conclude this thesis and talk about future work that would improve the ideas presented here.

2. Preliminaries

In this chapter we give some preliminary problem statements and related definitions that lead up to our formal problem statement. In the first Section 2.1 we start by giving some basic definitions related to graphs and paths in graphs and continue with defining the classic *Shortest Path Problem*. We continue in the next Section 2.2 with extending the problem to multiple criteria and introduce the concept of *Pareto-optimal*. Using the concept of *multi-criteria* we define the problem of finding paths under resource constraints, the *Constrained Shortest Path Problem*, in Section 2.3 and discuss how it applied to computing shortest paths for electric vehicles. Afterwards we introduce the concept of variable edge costs in Section 2.4 that depend on a parameter, such as time or speed.

2.1 Shortest Path Problem

In this sections, we give some basic definitions that we will reference throughout this work. A *directed weighted graph* $G = (V, E)$ consists of a set of nodes V and a set of edges $E \subseteq V \times V$. We require an edge cost function $w : E \to W$ that maps each edge $e \in E$ to its corresponding cost $w(e) \in W$, where W denotes the set of all possible cost values. A *path* p in G is an ordered tuple $(v_0, \ldots, v_k)$ with $(v_i, v_{i+1}) \in E$ for all $i \in \{0, \ldots, k-1\}$. We define the set of all possible paths in G as $P(G)$. Given a *path cost function* we can define the cost of a path in G as $w : P(G) \to W$. We define a *shortest path* as a path $p = (s, \ldots, t)$ in G for which $w(p)$ is minimal over all paths starting at s and finishing at t.

Definition 2.1. Shortest Path Problem

Given two nodes $s, t \in V$ find a shortest path $p = (v_0, \ldots v_k) \in P(G)$ for which $v_0 = s$ and $v_k = t$.

Whether this problem is well-defined depends on the cost function w and the graph G. An example is a graph with a cost function that allows for negative values. Certain graphs might contain negative cycles, that is, paths $p^- = (v_0, \ldots, v_0)$ for which $w(p^-) < 0$. In such graphs the question of a shortest path might not be answerable since we could insert many of these cycles in a path to make the total cost arbitrarily low. In reality this will not be a problem if the graph uses a cost based on time: It is generally not possible to go back in time, hence the cost is always positive. However, when considering energy-optimal paths for electric vehicles, the cost might be negative: For example, driving downhill will recharge the battery thanks to recuperation. Negative cycles, however, are not possible since we

use more energy driving uphill then we can regain by driving downhill. For this thesis we can thus assume that there are no negative cycles in the graph. If the notion of a shortest path is well-defined, we can also define a metric $dist_w(u, v) := \min_{(u,\dots,v)\in P(G)} w(u, \dots, v)$ for defining the distance of two nodes u and v in G. If no path exists between u and v we set the distance $dist_w(u, v) := \infty$.

A popular algorithm for solving the *Shortest Path Problem* with a scalar cost function $w : E \to \mathbb{R}_+$ in polynomial time is *Dijkstra's algorithm* [Dij59]. Algorithm 2.1 gives an overview this algorithm. Dijkstra's algorithm needs a priority queue Q that supports updating the key of each entry. The key for each node $v \in V$ is the currently best path cost $k_v := dist_w(s, v)$ from the start node s. The algorithm starts by inserting the start node $s \in V$ into the priority queue with cost 0. In each round of the algorithm we extract a node $u \in V$ from the priority queue. The sorting of the priority queue ensures that this node u has the lowest cost for all nodes in the queue. We compute the new tentative distance $dist'_w(s, v)$ of every neighbor v of u as $dist'_w(s, v) := dist_w(s, u) + w(u, v)$. If $dist'_w$ is smaller then the current tentative distance of s to v we update the key of v to $dist'_w(s, v)$ and record u as the parent node of v. When we extract the target node t from the queue, we can terminate the algorithm, as any following node cannot improve the distance to t anymore. By walking back from t and extracting the parent from each node, the shortest path can be reconstructed. Using a binary heap as priority queue and given graphs with positive edge costs this algorithm has a complexity of $O((m + n) \log n)$ [Cor01]. For graphs with a negative edge cost, Dijkstra's algorithm is not guaranteed to be polynomial, since each node might be processed multiple times. We call an algorithm that only ever processes a node v with a given label l_v once *label setting*, and an algorithm that processes a node and its corresponding label several times *label correcting*.

Algorithm 2.1 Dijkstra's Algorithm

```
 1: Q.Push(s, 0)
 2: while !Q.Empty( ) do
 3:     u ← Q.Pop( )
 4:     if u = t then
 5:         return Path to t
 6:     end if
 7:     for (u, v) ∈ E do                          ▷ Relax out going edges
 8:         dist(s, v) ← min{dist(s, v), dist(s, u) + w(u, v)}
 9:         Q.DecreaseKeyOrPush(v, dist(s, v))
10:     end for
11: end while
```

2.2 Multi-Criteria Optimization

Rather than being interested in solving the *Shortest Path Problem* for a scalar cost function $w(e)$, we might consider extending it to multiple criteria using a weight function $w : E \to \mathbb{R}^n$ that maps edges to a tuple containing multiple costs, such as travel time and energy consumption. In this scenario the notion of a shortest path needs to be re-defined as well, as there can be multiple paths that are optimal with regard to a specific criterion or offer good trade-offs between criteria. For example, given the two criteria of time and energy consumption, we might not only be interested in the path that is the shortest or the most energy efficient, but also in the path that is 5 minutes slower but saves 1kWh of the energy consumption. We say a value $w = (w_0, \dots, w_{n-1}) \in W$ is *dominated* by another value $w' = (w'_0, \dots, w'_{n-1})$ if each value w'_i is better or equal than the corresponding value w_i and strictly better in one criterion.

A common way to generalize the notion of a shortest path to the multi-metric case is the concept of Pareto sets. A *Pareto set* is a set of points $P \subset W$ where for every $p \in P$ it holds that there is no other point in $p' \in P$ that dominates p. We call a path with the path cost $w(p) = (w_0, \ldots, w_n)$ *Pareto-optimal* if there is no other path p' whose path cost $w(p')$ dominates $w(p)$. A Pareto-optimal path from node s to node v would thus be part of the Pareto set of all paths from s to v.

Definition 2.2. Pareto-optimal Path Problem

Given $s, t \in V$, find all paths $p = (s, \ldots, t)$ for which $w(p) = (w_0, \ldots, w_n)$ is not dominated by any other path from the start s to the target t.

An adapted version of Dijkstra's Algorithm can solve the *Pareto-optimal Path Problem*. Instead of using at most one label for every node $v \in V$ in the priority queue Q, a label for every undominated path from s to v needs to be inserted. Instead terminating as soon as the target node t is taken out of the queue Q, the algorithm needs to continue until the full Pareto set of t is found. To sort the labels in the priority queue Q, multiple approaches can be used. For example, the cost tuple $w_v \in \mathbb{R}^n$ in combination with lexicographical sorting can be used as key directly. Another option is to use a scalar key computed by a weighted sum of the criteria, or even just a single criteria. While we can adapt Dijkstra's algorithm to this problem the general polynomial running time guarantees do not hold for this problem as the size of each Pareto set can be exponentially large. In general we do not have any guarantees that the running times to solve the *Pareto-optimal Path Problem* will be polynomial.

2.3 Constrained Shortest Path Problem

In practice, we might not be interested in finding all possible trade-offs between multiple criteria, but only solutions that obey certain constraints. For example, while the Pareto set might include all Pareto-optimal time-consumption trade-offs, we might only be interested in finding the fastest solution that does exceed the maximum energy of the battery. We could still compute the whole Pareto set and only select the solution that fits our constraints after the fact, but this would mean we potentially spend a lot of time computing solutions that we will not use. Hence, we could incorporate the idea of splitting the criteria into costs that will be optimized and resource usage that will be limited. In the context of electric vehicles we primarily are concerned with two cases. The first case being computing energy-optimal paths that stay below a certain time limit (e.g. at most 20% longer then the shortest path). The second one being computing paths with minimal travel time that due not exceed the battery capacity of the vehicle.

To define the idea of a limited resource mathematically, we will formulate it as a function that transforms a bi-criteria cost tuple $(w_0, w_1) \in \mathbb{R}^2$. The first component w_0 contains the cost we want to minimize, and the second component w_1 contains the resource usage. If the resource usage exceeds the limit that we want to enforce, we set the cost to ∞ marking this solution as invalid. In addition to enforcing a maximal resource usage we can enforce a minimal resource usage, which is important in the case of electric vehicles since the consumption of certain edges can be negative. Of course a battery cannot hold more charge than its capacity, so we need to limit the consumption of paths to be a least 0. We capture this idea in the constrain function $con : \mathbb{R}^2 \to \mathbb{R}^2$ that enforces a minimum W_{min} and a maximum W_{max} on the resource w_1.

$$con(w_0, w_1) := \begin{cases} (\infty, \infty), & w_1 > W_{max} \\ (w_0, W_{min}), & w_1 < W_{min} \\ (w_0, w_1), & otherwise \end{cases}$$

Since we need to apply these constraints at every step of our path the definition of the path cost function then becomes recursive:

$$w(v_0, \ldots, v_n) := con(w(v_0, \ldots, v_{n-1}) + w(v_{n-1}, v_n))$$

Note that in this definition the constraint function is applied to the path cost after adding each individual edge cost. It is easy to see that if the cost of every edge in a path is positive in each criteria, it holds that $w(v_0, \ldots, v_n) = con(\sum_i w(v_i, v_{i+1}))$, the same is true if all values are negative. This can be exploited in cases where we want to delay the application of the constraint function, because we do not yet know all parameters. In fact Baum et al. [BDWZ17] use this idea to pre-compute the consumption profile of paths, without knowing the state of charge at the start node (yet).

Definition 2.3. Constrained Shortest Path Problem

Given $s, t \in V$ with constraint function $con : \mathbb{R}^2 \to \mathbb{R}^2$, find a path $p = (s, \ldots, t)$ with $w(p) = (w_0, w_1)$ for which w_0 is minimal and $w_1 < \infty$.

Again, we can use an adapted version of the multi-criteria version of Dijkstra's Algorithm to solve the *Constrained Shortest Path* problem. After computing the tentative cost w_v for a node $v \in V$, we apply the constraint function *con* before updating the label. If the resource consumption $w_{v,1}$ of $con(w_v)$ is not ∞, the path is feasible and the label can be updated, otherwise we discard the path. If the first criterion $w_{v,0}$ is non-negative, using it as the key for the priority queue Q will ensure that the key for each node $v \in V$ is monotonically increasing. As soon as the target node $t \in V$ is extracted from the queue for the first time, we have found the first feasible solution, since $w_{t,0} < \infty$ that satisfies our resource constraints. As a result we can terminate the search and return that solution as the shortest feasible path.

2.4 Variable Edge Costs

So far we have only considered constant cost functions, but in practical applications the cost of paths in the road network might depend on certain parameters. The canonical example is the travel time in road networks. During peek traffic hours streets will be filled with slow moving traffic affecting the travel time. As a result the average travel time during this peak hours will be much higher compared to times where we can drive at the free flow speed. Thus, we might consider modeling the cost of an edge as a function depending on the arrival time: $w : E \to [\mathbb{R}_+, \mathbb{R}_+]$. Since the cost is now denoted by a function and not scalar values, the operator $\oplus$ to combine path and edge costs needs to be re-defined. For example, in the time-dependent case we might define the operator as a function that maps arrival time to departure time, using the variable travel time $f(x)$. To calculate the arrival time at a node $v \in V$ if we can reach it over an edge $(u, v) \in V$ with variable edge cost $f : \mathbb{R}_+ \to \mathbb{R}_+$, we add the arrival time x at u to the travel time of (u, v) at point x, which can be calculate by $f(x)$:

$$\oplus : \mathbb{R}_+ \times [\mathbb{R}_+ \to \mathbb{R}_+] \to \mathbb{R}_+, (x, f) \mapsto x + f(x)$$

Using this operator on functions we can adapt the definition of the cost of a path seen in previous sections by replacing the normal addition on scalar values with the $\oplus$ operator:

$$w((v_0, \ldots, v_n), x_0) := w((v_0, \ldots, v_{n-1}), x_0) \oplus w(v_{n-1}, v_n), \text{ with } w((v_0, v_1), x_0) := w(v_0, v_1)(x_0)$$

Note that in this case the cost of a path is still a scalar value, but it requires knowledge about the arrival time x_0 at node v_0.

Definition 2.4. Time-dependent Shortest Path Problem

Given $s, t \in V$ and a departure time x_0 find a path $p = (s, \ldots, t)$ with minimal path cost $w(p, x_0)$.

Since the departure time x_0 is given and hence the path cost is scalar, it is simple to adapt Dijkstra's algorithm to this problem. Instead of starting with a cost of 0 at s, the initial cost is set to x_0. By applying the $\oplus$ operator defined above, the algorithm computes the arrival time at every node. Instead of using the distance to the start node as key for the priority queue Q, the arrival time is used.

The departure time at the start s might not always be known, so we can generalize this problem by treating the path cost as a function itself, which makes defining an operator $\oplus$ in a closed form hard. Computing the optimal arrival time function $f \oplus g$ in a closed form for which $(f \oplus g)(x_0) := x_0 + f(x_0) + g(x_0 + f(x_0))$ typically requires knowledge about the specific shape of f and g, see for reference Batz et al. [BDSV09]. We call the result of such an operation *path profile* of a path $p \in P(G)$, because it represents the arrival time for every possible departure time x_0. Note that for a node $v \in V$ the path with earliest arrival time given a departure time x_0 might not be the same for all $x_0 \in \mathbb{R}_+$ [FHS14]. Similar to multi-criteria optimization, we thus need to consider multiple solutions implied by different paths that each offer unique trade-offs. Thus, there might be no single path profile that is best for all departure times. To account for this we generalize the *Time-dependent Shortest Path Problem* to not only ask for a specific path, but for a function that maps each departure time to the best solution, possibly using different paths for different departure times x_0.

Definition 2.5. Time-dependent Shortest Path Profile Problem

Given $s, t \in V$ find a function $f : \mathbb{R}_+ \to \mathbb{R}_+$ for which $f(x_0)$ denotes the earliest arrival time at t with departure time x_0 at s.

To solve this problem we can again adapt Dijkstra's algorithm. Instead of using $dist_w(s, v)$ as the key for a node v in the queue, we use the earliest arrival time at v as the key. When processing the node v, we need to relax all outgoing edges (v, u), but instead of adding the distance values we need to use the operator $\oplus$ to compute $dist_w(s, v) \oplus dist_w(v, u)$. If the resulting arrival time profile improves the arrival time at u for some departure times at v, we need to insert it into the Pareto set of the node u. We can terminate the search once the earliest arrival time in the queue exceeds the maximal arrival time at t.

3. Problem Statement and Models

In this chapter we want to use the definitions from the previous Chapter 2 to formally define the problem statement we try to address in this thesis. Part of this problem statement is supplying realistic models for time-consumption trade-off functions and charging functions that we use in this work. We will also briefly dive into the topic of realistic turn costs. Unfortunately we conclude that an exact model for turn costs is infeasible in practice.

3.1 Electric Vehicle Shortest Path Problem

Computing shortest paths for electric vehicles incorporates aspects of all aforementioned problems:

- Like in the *Shortest Path Problem* we are interested in finding a path with shortest travel time.
- Like in the *Pareto-optimal Path Problem* we do not only consider a single criteria but include both travel time and consumption.
- Like in the *Constrained Shortest Path Problem* the resource usage on the electric consumption is limited by the battery of the vehicle.
- Like in the *Time-dependent Shortest Path Profile Problem* the edge cost is not static but variable.

In the context of electric vehicles, the cost of an edge is defined by two criteria: The time it takes to traverse the edge, and the consumption of energy. These two criteria are actually not independent: The consumption of energy very much depends on the speed of the electric vehicle. To include this in the model, we can use an edge cost function $w : E \to [\mathbb{R}_+ \to \mathbb{R}]$ that models the cost of an edge as a function itself. The total path cost depends on two parameters, the total travel time of the path x that will determine the consumption, and the initial consumption $y_0 \in \mathbb{R}$ at the start node v_0. For a path $p = (v_0, \ldots, v_n)$, we can represent this using a path cost function $w_p : (\mathbb{R}_+, \mathbb{R}) \to \mathbb{R}$ that maps time, initial consumption and travel time to the consumption at v_n. This function allows us to answer the question how slow we need to drive in order to reach v_n from v_k given an initial state of charge c, by finding the first time x for which the consumption $f(x, M - c)$ is below the battery capacity M. As a result we define the path cost function $w(p)$ to map a path to such a function.

$$w : P(V) \to [(\mathbb{R}_+, \mathbb{R}) \to \mathbb{R}], e \mapsto [f(x, y_0) \mapsto y]$$

Similar to the time-dependent case we need to define an operator $\oplus$ that defines how to combine the path cost with the edge cost.

$$\oplus : [(\mathbb{R}_+, \mathbb{R}) \to \mathbb{R}] \times [\mathbb{R}_+ \to \mathbb{R}] \to [(\mathbb{R}_+, \mathbb{R}) \to \mathbb{R}]$$

Given two consumption functions f and g the total consumption for a given travel time x depends on how we split that travel time between f and g. We might save more energy by driving slower on f than we would save driving slower on g. Mathematically we can define finding this optimal split of the travel time between f and g as solving a minimization problem as follows:

$$(f \oplus g)(x, y_0) := \min_{\Delta} f(\Delta, y_0) + g(x - \Delta) \text{ for every } x \in \mathbb{R}_+$$

This function represents the complete consumption profile for a path, that depends on the travel time x and initial consumption y_0. If we assume a fixed consumption of $y_0 = 0$ at the start node s we can simplify this path cost function to only depend on the travel time x.

So far the model assumes that the travel time of an edge and subsequently the travel time of the path can be chosen freely. However, due to physical constraints and legal limits, the speed on each edge is limited, which induces minimal and maximal travel times $[\underline{\tau}_e, \overline{\tau}_e)$ for every edge e. To incorporate these additional restrictions into the consumption function, we use a piecewise function that will map to ∞ for all travel times smaller then $\underline{\tau}_e$ and map to $f(\overline{\tau}_e)$ for all travel times that exceed the maximal travel time. This cost model ensures that no path exceeds the maximal speed, and driving slower than the minimum speed will not improve the consumption.

$$f_e(x) := \begin{cases} \infty, & x \leq \underline{\tau}_e \\ f_e(x), & \underline{\tau}_e < x \leq \overline{\tau}_e \\ f_e(\overline{\tau}_e), & x > \overline{\tau}_e \end{cases}$$

See Figure 3.1 for an example of an edge cost function of this form. This also induces a interval $[\underline{\tau}, \overline{\tau})$ of minimal and maximal travel time on the cost of the overall path. While this consumption function definition takes care of constraints of the minimal and maximal speed, we need to ensure that the consumption never exceeds the battery capacity and never over-charges the battery. To incorporate these battery constraints into this model we define a constraint function con_{bat} like in Section 2.3 using $W_{min} = 0$ and $W_{max} = M$, where M is the capacity of the battery of the electric vehicle. Applying these constraints will modify the minimal and maximal travel time of the consumption function, to ensure that maximum consumption $f(\underline{\tau})$ is below the battery capacity and the minimal consumption $f(\overline{\tau})$ is non-negative. See Figure 3.1 for an example of how the battery constraint is applied to a path cost function. Using this consumption function model we can give a formal problem statement:

Definition 3.1. Electric Vehicle Shortest Path Problem

Given $s, t \in V$, the initial battery consumption y_0 and the battery capacity M, find a path $p = (s, \ldots, t) \in P(G)$ that minimizes the travel time $\underline{\tau}$, for which the consumption $w(p)(\underline{\tau}, y_0)$ does not exceed the battery capacity M.

In other words we are looking for a path with minimal travel time that arrives at the target without running out of electricity. In Chapter 4 we describe an algorithm that solves this problem. It should be immediately clear that this problem is a variation of the *Constrained Shortest Path Problem* and thus NP-complete as well.

While the problem statement above incorporates the notion of conserving energy to reach the target, electric vehicles generally also have the option to recharge their battery. Recharging

is typically only possible at dedicated charging stations that we represent by a set of nodes $S \subseteq V$. Each node $v \in S$ has a charging function $g_v : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$ that maps charging time and arrival consumption to departure consumption with $g(0, y) = y$. How fast an electric vehicle can recharge its battery heavily depends on the type of charger and the current state of charge of the battery. As a result where and for how long an electric vehicle charges its battery heavily influences the shortest paths. When arriving at a charging node we would thus like to compute the optimal charging time, based on the current state of charge. Since the state of charge depends on the consumption profile of the path leading to the charging station, this amounts to finding optimal splits between driving slower or charging longer. Mathematically we can incorporate the notion of charging by introducing another operation $\odot$ that composes a charging function g with a trade-off function f. For every charging station $v \in S$ we insert a self-loop $(v, v) \in E$, all other nodes in the graph do not have self-loops. This enables us to compute the path cost in a concise manner, by applying the compose operator $\odot$ when we find a self-loop, and otherwise the combine operator $\oplus$.

$$w((v_0, \ldots, v_{n-1}), y_0) := \begin{cases} con_{bat}(g_{v_{n-1}} \odot w((v_0, \ldots, v_{n-2}), y_0)), & v_{n-1} = v_{n-2} \\ con_{bat}(w((v_0, \ldots, v_{n-2}), y_0) \oplus f_{v_{n-2}, v_{n-1}}), & \text{otherwise} \end{cases}$$

Using this mathematical model of a charging station we extend the problem statement from above to include charging stops.

Definition 3.2. Electric Vehicle Shortest Path Problem With Charging

Given $s, t \in V$, the initial battery consumption y_0, the battery capacity M and a set of charging stations $S \subseteq V$, find a path $p = (s, \ldots, t) \in P(G)$ that minimizes the travel time $\underline{\tau}$ for which the consumption $w(p)(\underline{\tau}, y_0)$ does not exceed the battery capacity M.

Merting et al. [MSS15] showed that including charging stations into the modeling already yields an *NP-complete* problem, even for constant edge costs, hence this problem is NP-complete too. Note that the *Electric Vehicle Shortest Path* problem is a special case of the previous problem statement for an empty set of charging stations $S = \emptyset$. In the remainder of this chapter we discuss the specific modeling of time-consumption trade-off functions and charging functions.

3.2 Trade-Off and Charging Models

In this section we introduce the basic model we need for discussing the properties of an electric vehicle and how these affect the minimal travel time of routes. The first building block is a mathematical way of representing the notion of driving slower to save energy. Energy consumption depends on three types of parameters, parameters intrinsic to the vehicle (e.g. friction losses, drag coefficient, motor efficiency), parameters depending on the environment (e.g. slope of the street, wind speed, traffic conditions), and parameters depending on the actual driving behavior (e.g. speed, acceleration). Note that these parameters are not necessarily independent. For example, the motor efficiency of a vehicle depends on the speed and torque, while the impact of the drag coefficient depends not only on the vehicle and speed, but also the wind conditions. We are going to use the model introduced by Baum et al. [BDWZ17] that models consumption with respect to speed (or travel time):

$$c(v) = \lambda_1 v^2 + \lambda_2 v + \lambda_3, \text{ where } \lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}$$

The parameters λ_1, λ_2, λ_3 are depend on the electric vehicle and the environment. These parameters can be determined empirically, for example based on real consumption profiles of vehicles combined with a curve fitting algorithm that extract the best parameters values.

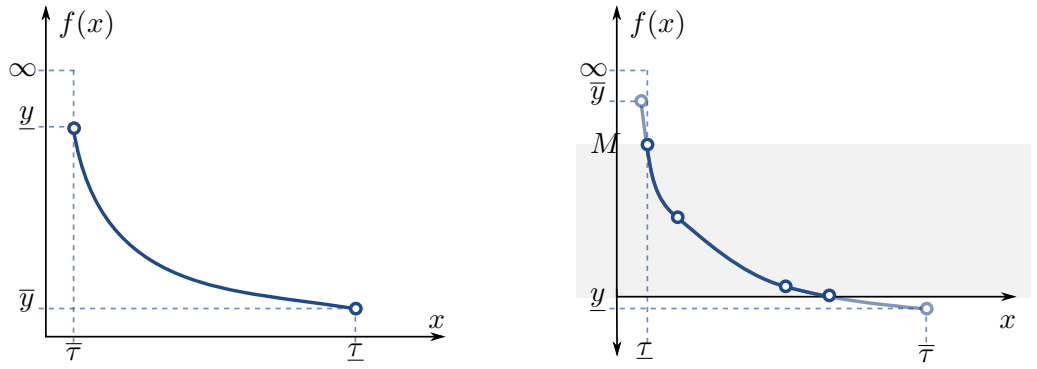

Figure 3.1: Left: An example consumption function f with minimal and maximal feasible times $[\underline{\tau}, \overline{\tau})$. Right: Clipping of a piecewise consumption function f to battery constraints with capacity M by adjusting $\underline{\tau}$ and $\overline{\tau}$.

Since we are interested in a model that captures the consumption of an edge in the graph for a given travel time x, we are going to apply the transformation from [BDWZ17] to the model:

$$f(x) = \frac{a}{(x-b)^2} + c, \text{ where } a > 0, b \geq 0, c \in \mathbb{R}$$

The parameter c can be both negative or positive, depending on the slope of the street, while the parameter b makes it easy to shift the function along the x-axis, which is a common operation in the algorithm we introduce in Chapter 4. The optimal combination of two hyperbolic trade-off functions, will in general not be a single hyperbolic function, but multiple piecewise functions. As a result we represent the consumption of a path using a piecewise function consisting of multiple hyperbolic functions.

Our goal is to represent charging functions in the same consumption function model as trade-offs for driving slower. For this we need to consider how charging is represented mathematically. The charging function model introduced by Baum et al. [BDG$^+$15] represents charging using a charging function cf that maps the charging time and state of charge at arrival to the state of charge after charging.

$$cf : [0, M] \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$$

We believe this model is superior to simpler models that only allow to fully charge the battery [GP14] [Sto12], since realistic charging functions are highly non-linear. It is thus much faster to only charge the battery up to a specific state of charge, in fact the Tesla Super-Charger network is designed to exploit this effect by only charging up to 80% of the battery capacity. See Figure 3.2 for examples of charging functions with different charging rates.

If we charge at a charging station for τ_1 seconds and then charge again for τ_2 seconds, we would expect the resulting charge to be the same as if we charged for the full time of $\tau_1 + \tau_2$. We can capture this requirement in the *shifting property*:

$$cf(cf(\beta, \tau_1), \tau_2) = cf(\beta, \tau_1 + \tau_2)$$

Baum et al. showed [BDG$^+$15] that all charging functions with the shifting property can be represented using a function $cf : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ that is independent of the state of charge:

$$cf(\beta, \tau) = cf(\tau + cf^{-1}(\beta))$$

To incorporate charging functions in the model of trade-off functions, we reformulate them as functions that given a consumption y and time x return a consumption $g(x, y) \leq y$.

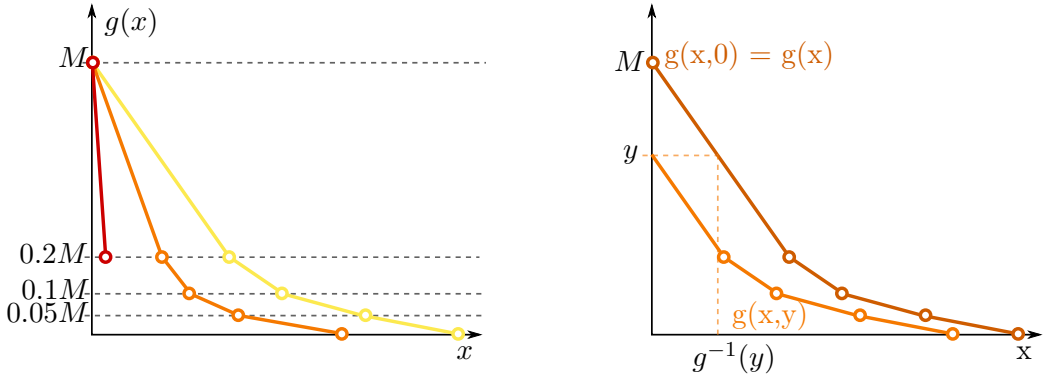

Figure 3.2: Left: Three charging functions with different charging rates. In red a super-charger that only charges up to 80% of the SoC (or here: reduces consumption to 20% of the battery capacity M). Right: The relation between a charging function $g(x, y)$ (light orange) that has the shifting-property and the corresponding function $g(x)$ (dark orange)

Please note that the original trade-off model already allows for trade-off functions to have a negative consumption, e.g. for downhill edges. Similar to the original formulation of a charging function cf, we can also represent the consumption-based charging function $g(x, y)$ using a single function $g(x) := M - cf(x)$ where M is the capacity of the battery:

$$g(x, y) = g(x + g^{-1}(y)) = M - cf(x + cf^{-1}(M - y))$$

Unlike Baum et al. we explicitly assume in this thesis that cf is a piecewise-linear function, which means for all times x in an interval $[\underline{\tau_i}, \overline{\tau_i})$ we can represent the function $g(x)$ as linear function where $g(x) = g_i(x) = d_i(x - b_i) + c_i$, see Figure 3.2 for an example.

When we arrive at a charging station, we will spend some time to exit the vehicle, plug it in the charger and pay. This overhead makes frequent charging stops impractical, so we want to include it in our model by adding a time penalty to the resulting time-consumption trade-off function. This is equivalent to a graph model where the actual charging station node $v \in S$ is only connected to the road network over two edges, that do not add any additional consumption but a constant time cost that is equal to the charging penalty. Note we do not need to use this explicit modeling of a charging penalty in the graph, but can incorporate this directly in the algorithm.

Using this charging model we see that the consumption function of a path can also contain linear components if a charging station is used. To account for this in the trade-off model we extend the trade-off function by an additional linear component with parameter d:

$$f(x) = \frac{a}{(x - b)^2} + c + d(x - b)$$

In Section 4.2 we show that this function always either has a hyperbolic component, or only a linear component, so either the parameter a is zero or the parameter d is zero. Using this model we define operations that compute optimal trade-off functions that represent the consumption profile of paths in a graph, potentially with charging along the way.

3.3 Turn Cost Models

In the models we have discussed so far, we assumed that the cost of an edge $e \in E$ is independent of which edge is the next edge in the path. In reality that assumption is not necessarily true: Consider for example an intersection with a traffic light for the left-turn but no traffic light for going straight. In general, turning left will take much longer than

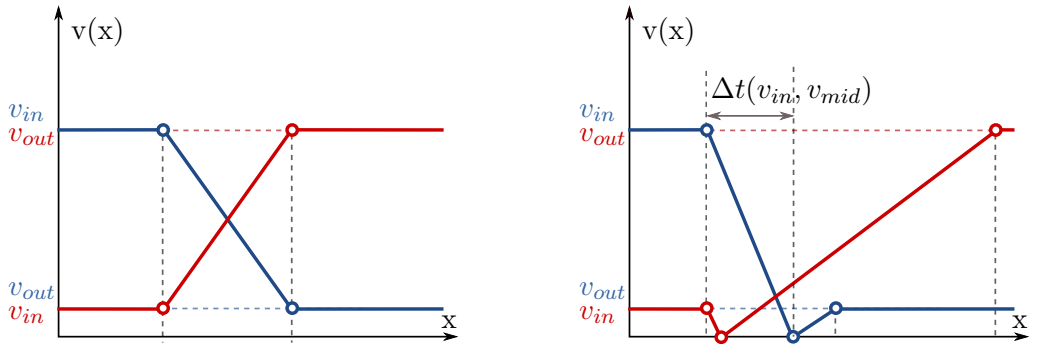

Figure 3.3: Left: Speed change without stopping, the blue curve shows a vehicle slowing down, the red curve a vehicle accelerating. Right: Speed change with stopping at the intersection.

driving straight through the intersection. However, in the current cost model there is no direct way to express that concept. Instead, the cost of the edge leading up the intersection is the same for both the left turn as well as going through the intersection. To account for this model inaccuracy we can use a different graph model, that treats turns as edges between edges in our original graph. We defined the *expanded graph* $E(G) := (T, E)$ where the set of turns $T \subset E \times E$, and for every turn $t = (e_1, e_2) \in T$ we require that both edges $e_1 = (u, v)$ and $e_2 = (v, w)$ share a node $v \in V$. By defining a cost function on the turns in this graph, instead of the edges in the original graph G, we are able to model different cost for every turn at an intersection. Using a turn cost function $w(t) = w(e_1, e_2) := w(e_1)$ we get a weighted graph that is equivalent to its unexpanded graph. Since $E(G)$ is also a graph, all definitions regarding edge and path costs apply directly to this graph as well.

Turn costs for an electric vehicle can be expressed in two ways, costs in time and costs in consumption. For example, when turning from a slower street to a faster street, e.g. from an motorway ramp on the motorway, we need to accelerate from a lower speed to a higher speed. Since the acceleration is limited it will take some time until we have reached the target speed. When we have reached the target speed the electric vehicle will have gained some kinetic energy, which means it consumed additional energy. Similarly, when decelerating we need some time to reach the lower target speed by braking, and electric vehicle looses some kinetic energy. Contrary to traditional fossil fuel based vehicles, this lost kinetic energy can be used to recharge the battery over a process called recuperation. If the physical parameters of the vehicle are known, we can calculate the loss or gain of energy and how long it takes to accelerate or decelerate. The kinetic energy of a vehicle moving with speed v (in m/s) and mass m (in kg) can be calculated as [Sin09]:

$$E_{kin} = \frac{1}{2}mv^2$$

To calculate the energy we gain or loose due to (de-)acceleration we calculate the difference in kinetic energy:

$$\Delta E(v_1, v_2) := E_{kin}(v_2) - E_{kin}(v_1) = \frac{1}{2}m(v_2^2 - v_1^2)$$

This energy difference does not translate into electric energy consumption directly: The conversion is governed by the efficiency of the process that is responsible for converting electric energy into kinetic energy, the motor/drive-shaft system of a vehicle. Since this is a mechanical system, it is subject to friction losses and other inefficiencies. As a result electric vehicles generally only convert a fraction of the battery energy into kinetic energy. We will refer to the efficiency of this system using the constant η_{motor}. When converting

kinetic energy to electric energy the recuperation system also has losses, we refer to its efficiency as η_{recup}. Given these constraints we can define the consumption based on a change in speed:

$$c(v_{in}, v_{out}) = \begin{cases} \Delta E(v_{in}, v_{out})/\eta_{motor}, & v_{out} \geq v_{in} \\ \Delta E(v_{in}, v_{out}) \cdot \eta_{recup}, & v_{out} < v_{in} \end{cases}$$

This however assumes that we can transition from speed v_{in} to speed v_{out} directly without first slowing down. This assumption is true for turns that do not involve any maneuvers, as for example speed limit changes on the same street. When maneuvers are involved we generally slow down to an intermediate speed for the actual turn and then accelerate again to the outgoing speed. See Figure 3.3 for an example of different speed profiles. We take this into account in our turn cost function, by defining a cost function with an additional intermediate speed v_{mid}:

$$c(v_{in}, v_{mid}, v_{out}) := c(v_{in}, v_{mid}) + c(v_{mid}, v_{out})$$

For turns that require a maneuver we set $v_{mid} = 0$, for turns not requiring a maneuver we set $v_{mid} = v_{in}$. To calculate the time needed to speed up or slow down we need the maximal and minimal acceleration of the vehicle. This highly depends on the vehicle, for example an electric sedan like the *Tesla Model S* can accelerate from $0km/h$ to $100km/h$ in about 2.6 seconds, which would amount to a net acceleration of $10.68m/s^2$. In comparison to that a compact car like the Peugeot Ion needs about 14 seconds for the same and has thus a maximal acceleration of $1.98m/s^2$. The minimal deceleration also depends on the type of the vehicle and the type of the ground surface. For example, on dry asphalt the typical car has a acceleration of about $-8m/s^2$, but on snow only $-4m/s^2$ [WAB16]. Given this maximal/minimal acceleration we can compute the minimal time Δt that it takes to change the speed:

$$\Delta t := \begin{cases} (v_{out} - v_{in})/a_{up}, & v_{in} < v_{out} \\ (v_{out} - v_{in})/a_{down}, & v_{in} \geq v_{out} \end{cases}$$

Using this minimal transition time, we can also compute the minimal transition length Δs as the integral of the speed profile:

$$\Delta s := \Delta t \cdot (v_{in} + 0.5 \cdot (v_{out} - v_{in}))$$

To define the time of a turn we set $\Delta t(v_{in}, v_{mid}, v_{out}) = \Delta t(v_{in}, v_{mid}) + \Delta t(v_{mid}, v_{out})$ and $\Delta s(v_{in}, v_{mid}, v_{out}) := \Delta s(v_{in}, v_{mid}) + \Delta s(v_{mid}, v_{out})$.

Incorporating this turn cost model into a constant edge cost model with a cost function $w : E \to (\mathbb{R}_+ \times \mathbb{R})$ on travel time and consumption is straight forward. Since we know the length of each segment we can compute the speed of an edge $e \in E$ as $v_e := w_0(e)/l_e$. For a turn $(e_1, e_2) \in T$ we first determine the fraction of $l_{e,1}$ that we need to accelerate/decelerate to the new target speed:

$$\alpha := \frac{\Delta s(v_{e,1}, v_{mid}) + \Delta s(v_{mid}, v_{e,1})}{l_{e,1}}$$

Using α we can scale the original cost of u, v proportionally and add both the transition time and the change in consumption due to an increase/decrease in kinetic energy.

$$w(e_1, e_2) := (1 - \alpha) \cdot w(e_1) + (\Delta t(v_{e,1}, v_{mid}, v_{e,2}), c(v_{e,1}, v_{mid}, v_{e,2})) \in \mathbb{R}^2$$

Extending this to a variable edge cost model is however more complicated: If the travel time for each edge is not fixed, the speeds cannot be derived statically. This means the

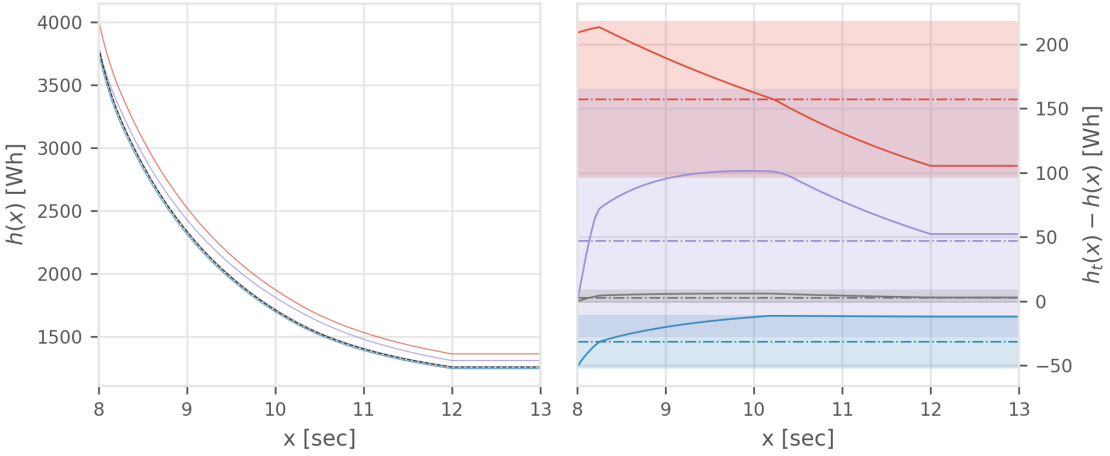

Figure 3.4: Left: Four different speed profiles speedup (red), brake (blue), fast (purple), slow (gray) and the resulting function $h(x) := (f \oplus_t g)(x)$ with additional turn consumption due to speed changes. As reference the $f \oplus g$ without additional consumption is included as black dotted line. Right: The error to the non-consumption cost aware $\oplus$.

cost of a turn does not only depend on the variable speed of the starting edge but also on the variable speed of the target edge. However, the speed of the target edge are not fixed yet, it depends on the following edge, whose speed is not fixed yet, and so forth. As a result we can only determine the actual turn cost of the turn when we can evaluate the speed of both the start and target edge which would requires knowledge of the full path taken. This makes using an expanded graph model unsuited for computing the optimal turn cost, since we can only base the turn cost on the start and target edge.

However, maybe we can incorporate the turn cost in the trade-off model itself. To simplify the calculation, we are going to assume for the moment that speed changes occur instantaneous, that means $\Delta t(v_{in}, v_{out}) = 0$ and $\Delta s(v_{in}, v_{out}) = 0$. In this case we can incorporate the consumption penalty in the $\oplus$ operator for combining trade-off functions of turns:

$$(f \oplus_t g)(x) := \min_{\Delta} f(\Delta) + c(\frac{\Delta}{l_{u,v}}, v_{mid}, \frac{x-\Delta}{l_{v,w}}) + g(x-\Delta)$$

If we assume $v_{mid} = v_{in}$ we can further simplify this equation using the helper function $v_{u,v}(x) := x/l_{u,v}$ and $v_{v,w}(x) := x/l_{v,w}$:

$$(f \oplus_t g)(x) = \min_{\Delta} f(\Delta) + g(x-\Delta) + \frac{1}{2} m \cdot \eta(v_{u,v}, v_{w,v}) \cdot (v_{v,w}(x-\Delta)^2 - v_{u,v}(\Delta)^2)$$

Where $\eta(v_{u,v}(x), v_{w,v}(x))$ is either η_{motor}^{-1} or η_{recup} depending on which speed is higher. For the case of $v_{u,v}(x) < v_{w,v}(x)$ we set $\alpha := \frac{1}{2} m \cdot \eta_{motor}^{-1}$ and we set the function:

$$h(x, \Delta) := f(\Delta) + g(x-\Delta) + \alpha(v_{v,w}(x-\Delta)^2 - v_{u,v}(\Delta)^2)$$

The derivation of this function in Δ yields:

$$h'_x(\Delta) = f'(\Delta) - g'(x-\Delta) + \frac{2 l_{v,w} \alpha}{x-\Delta} + \frac{2 l_{u,v} \alpha}{\Delta}$$

Finding the local minima of this function requires computing the roots of a polynomial of at least degree 5, for which there is no general closed form solution. Again, we see that incorporating exact turn costs on edge weights with trade-offs is infeasible, even for simplified models, for a shortest path algorithm since we cannot incorporate turn costs in

the graph model, nor can we efficiently extend the $\oplus$ operation to compute optimal path costs.

However, we can compute the optimal delta values numerically and give a comparison to omitting the turn cost, see Figure 3.4 for an example that compares the speed-aware $\oplus_t$ operator with the normal $\oplus$ operator. We can see that the error depends on the maximum speed difference between the two functions f and g. If both f and g have a low maximum speed the error of using $\oplus$ over $\oplus_t$ is very low, if both have a high maximum speed the difference is much bigger. The most extreme cases are when the maximum speed of f and g are very different, e.g. slowing down from a motorway edge to a side-road edge, or accelerating when turning on the motorway from a ramp. This suggests we might be able to derive a heuristic turn cost that tries to minimize the error of $\oplus$ and $\oplus_t$ only from the minimum and maximum speed of trade-off functions.

In this work we evaluate the use of three heuristics. The first heuristic adds a time penalty for every turn (u, v, w) where $deg(v) > 2$ in a route corresponding to the angle of the turn. For this we use the empirically determined cost function that is used in the open source routing engine OSRM [OSR17]: Given the maximal time penalty τ_{max} of a turn and the turn angle α between -180° and 180°, the cost function uses a sigmoid function to progressively apply a higher penalty the sharper the turn is:

$$\frac{\tau_{max}}{1 + e^{-13 \cdot |180 - \alpha| / 180. - 6.5}}$$

The second heuristic uses the insight we gained above that the average consumption error between $\oplus_t$ and $\oplus$ can be approximated by taking the average of the maximum and minimal consumption error, which can be easily computed. The third heuristic combines both approaches to calculate consumption and time penalties. Since both of these models are static, we can apply them during preprocessing and add them to the cost of a turn in the edge-expanded model. For U-turns we assume an additional penalty of 20 seconds in all turn cost models.

4. Base Algorithm

In this chapter we describe our base algorithm for solving the *Electric Vehicle Shortest Path Problem with Charging*. Our algorithm is based on a generalization of Dijkstra's algorithm, the *Function Propagation* (FP) Algorithm described by Baum et al. in [BDWZ17]. We extend the FP algorithm to work with charging stations using the model proposed in the previous chapter and call the resulting algorithm *Function Propagation With Charging* (FPC). The FP algorithm (and FPC algorithm) uses convex piecewise functions to represent the cost of a path $(s, \dots, v)$ from the start node s to a node $v \in V$. Since there might be multiple paths from s to v that offer unique trade-offs, the label $L(v)$ at the node $v \in V$ will be a *set* of multiple label entries $l_v = (f_{s,v}, \Delta_{s,v}, u)$, where $u \in V$ is the node over which we reached v, the function $f_{s,v} : \mathbb{R}_+ \to \mathbb{R}$ is the convex piecewise function that denotes all time-consumption trade-offs from s to v for this specific path, and the function $\Delta_{s,v} : \mathbb{R}_+ \to \mathbb{R}_+$ lets us determine the split between time spent on the previous segment $f_{s,u}$ and the current segment $f_{u,v}$.

As part of the Dijkstra search we create new label entries by visiting adjacent nodes and computing the optimal time-consumption trade-off to reach that node over a given edge. We call that operation *combining* two trade-off functions. In Section 4.1 we describe how to compute optimal combinations efficiently.

Our extended algorithm also needs to determine optimal charging trade-offs when arriving at a charging station, we call this operation *composing* a charging function with a trade-off function. In Section 4.2 we describe how to compute optimal compositions of charging functions and trade-off functions.

Not all paths from s to v represent sensible trade-offs, so we only include a label entry l if it offers a better time-consumption trade-off than all other labels. For this we need to check if for the label entry l if there exists a time x such that the trade-off function $f_{s,v}$ is better then all other trade-off functions f in the label set, that is $f_{s,v}(x) < f(x)$. We call a label entry that fulfils this criteria *undominated* by $L(v)$. In Section 4.3 we describe how we compute the dominance checks efficiently for piecewise trade-off functions.

In the last Section 4.4 we discuss some important implementation details for a practical use of these algorithms, such as considerations about floating point precision, function representation, and label storage.

Algorithm 4.1 Function Propagation with Charging Stations (FPC) Dijkstra

```
1: Q.Push(s, 0)
2: μ ← ∞
3: while !Q.Empty( ) and Q.MinKey( ) ≤ μ do
4:     v ← Q.pop( )
5:     (f_{s,v}, Δ_{s,v}, u) ← L.pop(v)
6:     k ← L.MinKey(v)
7:     Q.IncreaseKey(v, k)
8:     if v = t then                                   ▷ Update upper bound
9:         μ ← min{μ, Q.MinKey(t)}
10:    end if
11:    if v ∈ S then          ▷ Composing current label with charging function
12:        S_{s,v} ← g_v ⊙ f_{s,v}
13:        for all (f^j_{s,v}, Δ^j_{s,v}) ∈ S_{s,v} do
14:            L.Push(v, (f^j_{s,v}, Δ^j_{s,v}, v))
15:        end for
16:        k ← L.MinKey(v)
17:        Q.DecreaseKey(v, k)
18:    end if
19:    for (v, w) ∈ E do                               ▷ Relax outgoing edges
20:        (f_{s,w}, Δ_{s,w}) ← f_{s,v} ⊕ f_{v,w}
21:        L.Push(w, (f_{s,w}, Δ_{s,w}, v))
22:        k ← L.MinKey(w)
23:        Q.DecreaseKeyOrPush(v, k)
24:    end for
25: end while
```

4.1 Optimally Combining Trade-Off Functions

Every time we relax a node v over a node u in Algorithm 4.1 we need to compute the new tentative cost to reach the node v from the start s using the given edge (u, v). For fixed values this is rather simple, we just need to add the cost of reaching the node u from the source s to the cost of the edge (u, v) to form the new tentative cost from s to v. However, the FPC algorithm does not use fixed values but represents costs as functions, that each represent all possible time-consumption trade-offs. This brings us to the problem of computing how to combine these trade-off functions to represent the trade-offs of the new path we just found to v. In other words, we are trying to decide how much time to spend on the path that leads to u vs. the time to spend on the path from u to v. This split is generally not a constant value: If we can spent a lot of time to get to v, we can use the more energy-efficient trade-off for both the path to u and the path to v. However, if we only want to invest a short amount of time we need to consider for which sub-path driving slower would save more energy. Thus, for every amount of time x we can spent on the overall path from s to v, we need to compute the optimal time split between the path leading to u and the path from u to v. If we denote the time-consumption trade-off function of the path to the node u as f, and the trade-off function of the path from u to v as g we can formally define the following optimization problem:

$$h(x) := \min_{\Delta} f(\Delta) + g(x - \Delta), \text{ where } \Delta \in [\underline{\tau}_f, \overline{\tau}_f), x - \Delta \in [\underline{\tau}_g, \overline{\tau}_g)$$

For convenience we will define the function $h(x, \Delta) := f(\Delta) + g(x - \Delta)$ to refer to a specific trade-off between f and g. It is easy to see that the minimal travel time $\underline{\tau}$ of the whole path is $\underline{\tau}_f + \underline{\tau}_g$ and the maximal travel time $\overline{\tau}$ is $\overline{\tau}_f + \overline{\tau}_g$. Due to the shape of cost functions we

defined in Section 3.1, it is easy to see that the minimal value $h(\overline{\tau})$ is equal to $f(\overline{\tau}_f) + g(\overline{\tau_g})$, and the maximal value $h(\underline{\tau}) = f(\underline{\tau}_f) + g(\underline{\tau}_g)$.

The definition of $h(x)$ as minimization problem is equivalent to finding a function $opt_\Delta(x)$ such that $h(x, opt_\Delta(x)) = h(x)$. The first step for computing opt_Δ is to derive the local minimum in Δ of $h(x, \Delta)$. We do this by using the classic analytical approach of finding the roots of the derivative $\frac{\delta}{\delta\Delta} h(x, \Delta) = 0$.

$$\begin{array}{rcl} h'_x(\Delta) & = & f'(\Delta) + g'(x - \Delta) \cdot (-1) \\ 0 & = & f'(\Delta) - g'(x - \Delta) \\ f'(\Delta) & = & g'(x - \Delta) \end{array} \tag{4.1}$$

This means for very x we need to find a value Δ^* for which both $f(\Delta^*)$ and $g(x - \Delta^*)$ have the same slope. Baum et al. use this insight to derive a closed form solution for the function $opt_\Delta(x)$. In case both functions f and g do not have a linear component ($d = 0$) this solution is also valid for our trade-off function model. An extensive description of this can be found in [BDWZ17], we will omit the details for these cases here. However, in the case that either f or g have a linear component, this solution is not valid. Section 4.2 shows that in this case we are either dealing with a linear or a hyperbolic function. It is easy to see that the operator $\oplus$ is commutative, so without a loss in generality we can restrict ourselves to finding the optimum for two cases: f and g both being linear functions, and the function f being a hyperbolic while g is a linear function. For each case we will prove a lemma that specifies how to compute the optimal value of opt_Δ from which $h(x)$ can be computed. Lemma 4.1 proves an optimal choice for opt_Δ for the case of two linear functions, Lemma 4.2 does the same for the case of a hyperbolic and a linear function. In Figure 4.1 we show, for both cases, an example of two optimal trade-off combinations.

Lemma 4.1. *If f and g are both linear functions, the optimal solution $opt_\Delta(x)$ dependents on their slope $f'(x) = d_f$ and $g'(x) = d_g$. For the case of $f'(x) = d_f < d_g = g'(x)$ the optimal Δ is computed by:*

$$opt_\Delta(x) = \begin{cases} x - \underline{\tau}_g, & \text{if} \quad \underline{\tau} \leq x < \overline{\tau}_f + \underline{\tau}_g \\ \overline{\tau}_f, & \text{if} \quad \overline{\tau}_f + \underline{\tau}_g \leq x \leq \overline{\tau} \end{cases}$$

Otherwise the optimal Δ is computed by:

$$opt_\Delta(x) = \begin{cases} \underline{\tau}_f, & \text{if} \quad \underline{\tau} \leq x < \underline{\tau}_f + \overline{\tau}_g \\ x - \overline{\tau}_g, & \text{if} \quad \underline{\tau}_f + \overline{\tau}_g \leq x \leq \overline{\tau} \end{cases}$$

Proof. In the case of two linear functions the general form of $h(x, \Delta)$ is:

$$\begin{array}{rcl} h(x, \Delta) & = & d_f(\Delta - b_f) + c_f + d_g(x - \Delta - b_g) + c_g \\ & = & (d_f - d_g)\Delta + d_g x + c_f + c_g - d_f b_f - d_g b_g \end{array}$$

If $d_f = d_g$ the function h is independent of Δ, in that case any trade-off between f and g is optimal, so we set the function $opt_\Delta$to interpolate between f and g on $[\underline{\tau}, \overline{\tau})$:

$$\Delta(x) := \frac{\overline{\tau}_f - \underline{\tau}_f}{\overline{\tau} - \underline{\tau}} \cdot (x - \underline{\tau}) + \underline{\tau}_f$$

In all other cases the minimum of this function in Δ depends on the sign of $d_f - d_g$. If $d_f < d_g$ we minimize h by setting the largest feasible value for Δ. If $d_f > d_g$ we minimize h by setting the smallest feasible value for Δ.

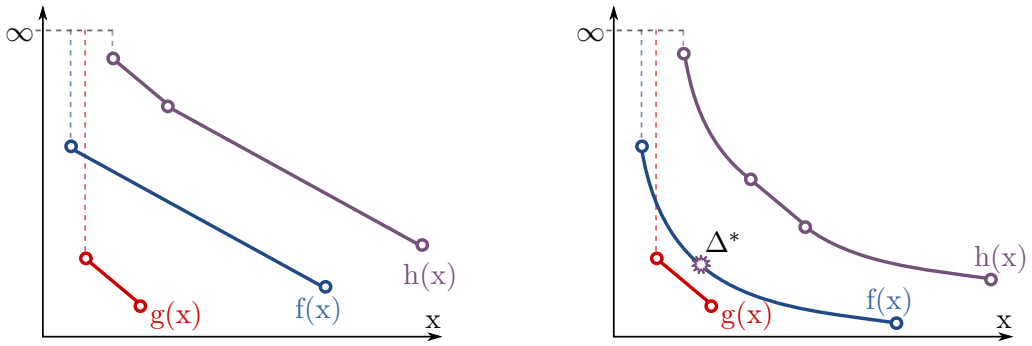

Figure 4.1: Left: Example of combining two linear functions f and g to the optimal trade-off $h = f \oplus g$. Right: Example of combining a hyperbolic function f with a linear function g to the corresponding optimal trade-off $h = f \oplus g$. Δ^* denotes the point where f and g have the same slope.

Case 1: The slope of f is strictly smaller then the slope of g.

In this case it holds that $d_f < d_g$. For $x \in \left[\underline{\tau}_f + \underline{\tau}_g, \overline{\tau}_f + \underline{\tau}_g\right]$ using the constraint $\Delta \leq x$ we can compute the maximum value for Δ as: $\Delta(x) = x - \underline{\tau}_g$ since $x - \Delta(x)$ needs to be at least $\underline{\tau}_g$ to be feasible. For $x \in \left[\overline{\tau}_f + \underline{\tau}_g, \overline{\tau}_f + \overline{\tau}_g\right]$ the constraint $\Delta \leq \overline{\tau}_f$ sets the maximum feasible value for Δ to $\overline{\tau}_f$.

Case 2: The slope of f is bigger or equal to the slope of g.

In this case it holds that $d_f \geq d_g$. For $x \in \left[\underline{\tau}_f + \underline{\tau}_g, \underline{\tau}_f + \overline{\tau}_g\right]$ using the constraint $\Delta \geq \underline{\tau}_f$ we can compute the minimum value for Δ as: $\Delta(x) = \underline{\tau}_f$. For $x \in \left[\underline{\tau}_f + \overline{\tau}_g, \overline{\tau}_f + \overline{\tau}_g\right]$ the constraints $\Delta \leq \overline{\tau}_g$ and $\Delta \leq x$ sets the minimum feasible value for Δ is $x - \overline{\tau}_g$.

□

Lemma 4.2. *If f is a hyperbolic function and g is a linear function, there exists a point Δ^* for which $f'(\Delta^*) = g'(x) = d_g$. With the value $\underline{\tau}^* := max\{\underline{\tau}, \Delta^* + \underline{\tau}_g\}$ and the value $\overline{\tau}^* := min\{\overline{\tau}, \Delta^* + \overline{\tau}_g\}$ the function opt_Δ can be computed in the following way:*

$$opt_\Delta(x) = \begin{cases} x - \underline{\tau}_g, & \text{if} \quad \underline{\tau} \leq x \leq \underline{\tau}^* \\ \Delta^*, & \text{if} \quad \underline{\tau}^* < x < \overline{\tau}^* \\ x - \overline{\tau}_g, & \text{if} \quad \overline{\tau}^* \leq x \leq \overline{\tau} \end{cases}$$

Some of these cases might degenerate if the value $\underline{\tau}^*$ is equal to $\underline{\tau}$ or the value $\overline{\tau}^*$ is equal to $\overline{\tau}$.

Proof. We start by computing the local minimum of $h(x, \Delta)$ using Equation 4.1.

$$f'(\Delta) = g'(x - \Delta) = d_g$$

Using the inverse function of the derivative $(f')^{-1}(\alpha) = b_f + \sqrt[3]{-2 \cdot a_f/\alpha}$, we can compute the value $\Delta^* := (f')^{-1}(d_g)$. The derivative $g'(x - \Delta^*) = d_g$ is constant for $\underline{\tau}^* \leq x \leq \overline{\tau}^*$, so there is a local minimum of $h_x(\Delta)$ and the optimal solution of $opt_\Delta = \Delta^*$.

The function f is a strictly monotonically decreasing function, for every $x \in [\underline{\tau}, \underline{\tau^*})$ we thus see that $f'(x - \underline{\tau}_g) \leq f'(\Delta^*) = d_g$. With $h'_x(\Delta) = f'(\Delta) - d_g \leq 0$ we see that $h(x, \Delta)$ is

monotonically decreasing in Δ for $\underline{\tau}_f \leq \Delta \leq x - \underline{\tau}_g$, hence the optimal value for Δ is the maximal feasible value $x - \underline{\tau}_g$.

Similarly, we can again use the fact that f is strictly monotonically decreasing to see that $d_g = f'(\Delta^*) \leq f'(x - \overline{\tau}_g)$ for every $x \in [\underline{\tau}, \underline{\tau^*})$. With $h'_x(\Delta) = f'(\Delta) - d_g \geq 0$ we see that $h(x, \Delta)$ is monotone increasing in Δ for $x - \overline{\tau}_g \leq \Delta \leq \overline{\tau}_f$, hence the optimal value for Δ is the minimal feasible value $x - \overline{\tau}_g$.

□

Optimally Combining Piecewise Functions

Using these lemmas for combining single trade-off functions, we define an algorithm that takes a convex piecewise trade-off function f together with the cost of an edge represented by a single trade-off function g, and produces a convex piecewise linear function $f \oplus g$ that represents the optimal trade-off between the two functions. Algorithm 4.2 gives an overview of this algorithm. The optimal piecewise trade-off function $f \oplus g$ is computed in a linear sweep over the convex piecewise function f. For every sub-function f_i we need to find out if it is worth to first decrease our speed on f_i before we decrease the speed on g. This is only the case if the function f_i has a steeper sloper (or higher rate), so increasing the travel time by the same amount on f, as opposed to g would yield higher energy savings. The algorithm determines the first sub-function f_{i+1} where it does not pay off to first drive slower on f.

For this we iteratively combine every sub-function f_i with g yielding at most $k \in \mathbb{N}$ sub-functions $h_0, \ldots, h_{k-1}$. We can use the last sub-function h_{k-1} to determine whether we can improve the solution by first driving slower on the next sub-function f_{i+1} as opposed to g. For this we compute the minimal slope α of the last solution h_{k-1}. If the minimal slope α is bigger than the maximal slope of g, there can be no value $\Delta \geq \overline{\tau}_{f,i}$ for which the slope $f'(\Delta)$ is smaller or equal to the slope $g'(x - \Delta)$, hence it is always better to decrease the speed on g first. Since f is a convex function, the slope of all following sub-functions will also be strictly bigger than the maximal slope of g. Thus, the optimal solution will be $opt_\Delta(x) = x - \overline{\tau}_g$ and we can compute all following sub-functions of h using this $\Delta(x)$ function, which amount to shifting all sub-function by $\overline{\tau}_g$ to the right, and adding the minimal consumption of g, $g(\overline{\tau}_g)$.

If the minimal slope α is smaller than the maximal slope of f_i it holds that $f'(\overline{\tau}_{f,i}) = g'(\overline{\tau}_g)$, since the last segment must be a shifted part of f_i. Since f is convex, we see that the slope of any follwong sub-function $f'_{i+1}(\Delta)$ will be bigger then $g'(\overline{\tau}_g)$ for all $\Delta \geq \overline{\tau}_{i,f}$. Again, the optimal solution is $opt_\Delta(x) = x - \overline{\tau}_g$.

In any other case we determine the minimal value for $x - \Delta$ for which $g'(x - \Delta) \geq f'(\Delta)$. In this case we only need to add $h_0, \ldots, h_{k-2}$ since we can potentially get a better trade-off for h_{k-1} by combining f_{i+1} with g.

4.2 Optimally Composing Charging and Trade-Off Functions

Given a charging station $v \in S$, the charging function model represents charging at this node using a piecewise-linear function $g(x, y)$ that depends on the charging time x and consumption y. We can decrease the consumption at v by increasing the time spend on charging. In the degenerate case of no charging we require that $g(0, y) = y$, see Figure 3.2 for an example. In this section we define the operation of composing the charging function g with time-consumption trade-off function f. Contrary to the operator $\oplus$ on trade-off functions, we do not minimize the sum of two trade-off functions. Instead, we define the

Algorithm 4.2 Combining a piecewise trade-off function f with a trade-off function g

$h \leftarrow \{\}$
$i \leftarrow 0$
finished $\leftarrow$ False
while !*finished* **do**
    Compute $f_i \oplus g$ as piecewise function h_i consisting of sub-functions $h_0, \ldots, h_{k-1}$
    Append $h_0, \ldots, h_{k-2}$ to h
    Minimal slope of last sub-function $\alpha := h'_{k-1}(\underline{\tau}_{k-1})$
    if $\alpha \geq g'(\overline{\tau}_g)$ or $\alpha < f'_i(\overline{\tau}_{f,i})$ **then**
        finished $\leftarrow$ True
    else
        $\underline{\tau}^*_g := (g')^{-1}(\alpha)$
        $\underline{\tau}_g \leftarrow \max\{\underline{\tau}_g, \underline{\tau}^*_g\}$
    end if
    $i \leftarrow i + 1$
    finished $\leftarrow$ *finished* $\vee (i = n)$
    if *finished* **then**
        Append h_{k-1} to h
    end if
end while
$\underline{g} := g(\overline{\tau}_g)$
Append $f_{i+1}, \ldots, f_{n-1}$ to h, shifted by $\overline{\tau}_g$ to the right and with the added consumption of $\underline{g}$.
return h

operator $\odot$ that maps a charging function and a trade-off function to a trade-off function that represents the optimal trade-off between driving slower and charging at this node.

$$\odot : [\mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}] \times [\mathbb{R}_+ \to \mathbb{R}] \to [\mathbb{R}_+ \to \mathbb{R}]$$

We call the $\odot$ operator composition to differentiate it from the combination operator $\oplus$. Intuitively that means we need to find all points in time where it would pay off to start charging instead of driving slower. Due to the non-linear nature of the realistic charging function model proposed in Section 3.2, it heavily depends on the current state of charge of the battery whether it pays off to start charging.

Mathematically we can define finding the optimal charging point $\Delta \in \mathbb{R}_+$ for a trade-off function f and a charging function g as:

$$(g \odot f)(x) := \min_{\Delta} g(x - \Delta, f(\Delta)), \text{ where } \Delta \in [\underline{\tau}_f, \overline{\tau}_f), x - \Delta \in [\underline{\tau}_g, \overline{\tau}_g)$$

This is equivalent to finding a function $opt_\Delta(x)$ for which $g(x - opt_\Delta(x), f(opt_\Delta(x)))$ is minimal for every $x \in \mathbb{R}$. We now define and prove Lemma 4.3 that will be useful to give an efficient algorithm to compute the optimal composition of trade-off and charging functions and derive some useful corollaries from it.

Lemma 4.3. *Given a charging function $g(x, y)$ and a trade-off function $f(x)$, all local minima $\Delta \in [\underline{\tau}_f, \overline{\tau}_f)$ of the function $g(x-\Delta, f(\Delta))$ for which g is differentiable in $g^{-1}(f(\Delta))$ and f differentiable in Δ satisfy the following equation:*

$$g'(g^{-1}(f(\Delta))) = f'(\Delta)$$

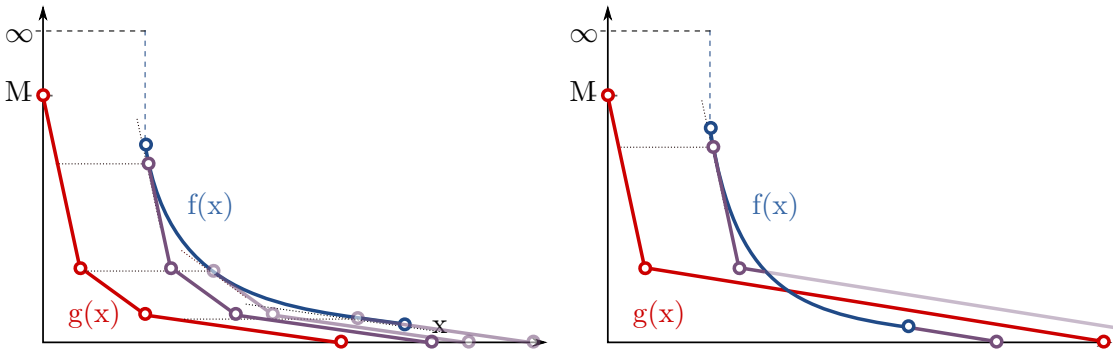

Figure 4.2: Left: Composing a charging function g with a consumption function f yields multiple solutions that are dominated by the first one. Right: The optimal solution of combining f and g is not convex and consists of disjoint parts of f and g.

Proof. As per definition of a charging function in Section 3.2 $g(x, y)$ has the shifting property. That means $g(x, y) = g(x + g^{-1}(y))$ for a univariant function $g(x)$. Using this we see that find the alternative form of $g(x - \Delta, f(\Delta))$:

$$g(x - \Delta + g^{-1}(f(\Delta)))$$

For convenience we are going to bundle all terms that depend on Δ in a helper function w:

$$w(\Delta) := \Delta - g^{-1}(f(\Delta))$$

With this we can rewrite $g(x - \Delta, f(\Delta))$ as $g(x - w(\Delta))$, and derive the partial differential of g in Δ as:

$$\frac{\delta}{\delta \Delta} g(x - \Delta, f(\Delta)) = g'(x - w(\Delta)) \cdot (-w'(\Delta))$$

The partial differential is zero if and only if either $g'(x - w(\Delta))$ is zero, or $w'(\Delta)$ is zero.

Case 1: The slope of g is zero.

In this case we know that $g'(x - w(\Delta)) = 0$. As per our definition of $g(x)$ as a monotonically decreasing function this is only true for two cases. The first case is $x - w(\Delta) > \overline{\tau}_g$ for which the composed function $g(x - \Delta, f(\Delta)) = g(\overline{\tau}_g, f(\Delta))$ is constant and an optimal choice for Δ would be x (meaning no charging at all). The second case is $x - w(\Delta) < \underline{\tau}_g$, which would imply that $x < \Delta$. So the first case will never yield a local minimum.

Case 2: The slope of w is zero.

In this case we know that $w'(\Delta) = 0$. Using the derivative $w'(\Delta) = 1 - (g^{-1})'(f(\Delta))f'(\Delta)$ we see that $w'(\Delta) = 0$ if and only if $f'(\Delta) = 1/(g^{-1})'(f(\Delta))$. Since g is a continuous, differentiable function we use the identity:

$$(g^{-1})'(y) = \frac{1}{g'(g^{-1}(y))}$$

From this it follows that $w'(\Delta) = 0$ if and only if $g'(g^{-1}(f(\Delta))) = f'(\Delta)$. □

Intuitively Lemma 4.3 tells us that we can find optimal points to start charging where the slope of f corresponds with the slope of g, for the same consumption value. Using Lemma 4.3 we can prove a few useful properties if we require that $g(x)$ is a piecewise-linear function, and f is a convex function. First of all, we can derive Corollary 4.4, that derives the general shape of all optimal charging points Δ.

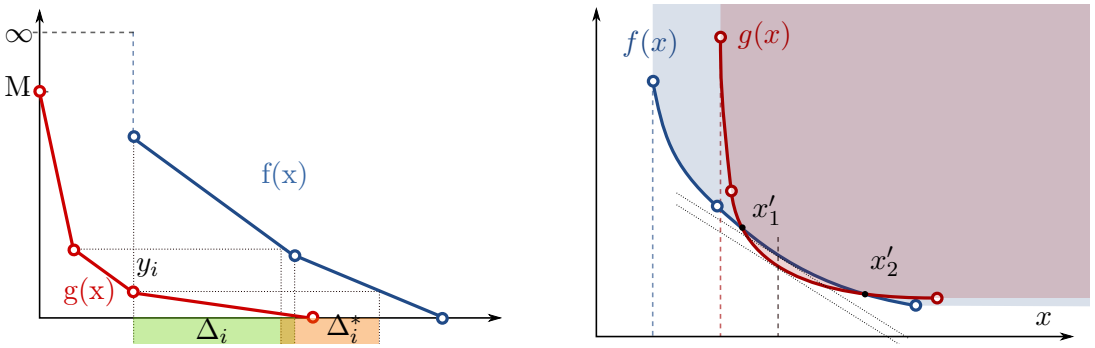

Figure 4.3: Left: Compose a charging function with a piecewise linear consumption function. Illustrates the intervals used in the prove for Corollary 4.4. Right: Function g (red) intersections function f (blue) twice at x_1' and x_2' and is thus not dominated by f on the second segment. The dotted lines represent the slope at the critical point.

Corollary 4.4. *Given a piecewise-linear function $g(x)$ with $k \in \mathbb{N}$ segments, the corresponding charging function $g(x, y)$, and a convex trade-off function $f(x)$, there are most k intervals $[\underline{\Delta}_i, \overline{\Delta}_i)$ for which all $\Delta \in [\underline{\Delta}_i, \overline{\Delta}_i)$ are a local minimum of $g(x - \Delta, f(\Delta))$. For all values $\Delta \in [\underline{\Delta}_i, \overline{\Delta}_i)$ the function $g(x - \Delta, f(\Delta))$ has the same value.*

Note that we explicitly allow degenerated intervals $[\Delta, \Delta]$ that only contain a single point. In fact, this will always be the case if the slope of f is strictly monotonically increasing, which is the case for hyperbolic functions.

Proof. Since the function g is a piecewise-linear function with $k \in \mathbb{N}$ segments, there are at most k distinct values for $g'(x) = \alpha_i$. The function f is convex, which means that $x_1 < x_2 \Rightarrow f'(x_1) \geq f'(x_2)$.

By identifying the minimal and maximal values $[\underline{\Delta}_i, \overline{\Delta}_i)$ for which each value $\Delta \in [\underline{\Delta}_i, \overline{\Delta}_i)$ has the property that the slope of $f'(\Delta)$ is equal to α_i, we form at most k intervals. Since $g^{-1}(y)$ is a monotonically decreasing function, we can also find a minimal values $\underline{y}_i$ and maximal values $\overline{y}_i$, for which each value $y \in [\underline{y}_i, \overline{y}_i)$ has the property $g'(g^{-1}(y)) = \alpha_i$.

The function $f(x)$ is convex and monotonically decreasing, and thus invertible. The inverse $f^{-1}(y)$ is also a continuous function and maps intervals $[\underline{y}_i, \overline{y}_i)$ to intervals $[\underline{\Delta^*}_i, \overline{\Delta^*}_i)$. Intersecting $[\underline{\Delta^*}_i, \overline{\Delta^*}_i)$ and $[\underline{\Delta}_i, \overline{\Delta}_i)$ yields at most k intervals for which every value Δ implies $g'(g^{-1}(f(\Delta))) = f'(\Delta)$, and thus a local minimum of $g(x - \Delta, f(\Delta))$. What is left to show is that all Δ values in the same interval yield the same local minimum. This is easy to see if you consider that both f and g have the same slope for all Δ in each interval, the resulting function $g \odot f$ would be the same. See Figure 4.3 for an illustration on how these intervals correspond. □

We can prove another corollary from this that shows that the operator $\odot$ is well defined:

Corollary 4.5. *The optimal function $opt_\Delta(x)$ is a piecewise function consisting of either constant sub-functions or the identity function $y = x$.*

Proof. From Corollary 4.4 it is easy to see that all optimal values for $\Delta < x$ are constant values. Where a local minimum is implied by a non-degenerated interval $[\underline{\Delta}, \overline{\Delta})$ we can choose any of those values. The remaining case of $\Delta = x$ implies an identity function as the solution for $opt_\Delta(x)$. □

With Corollary 4.5 we see that $g \odot f$ is either a portion of the charging function, and thus a linear function, or a portion of the consumption function f. Since linear functions are included in the set of all possible consumption functions in our model, we see that the $\odot$ operator is well defined.

We are now use these insights to define an algorithm for efficiently composing piecewise trade-off functions with charging functions. The optimal charging trade-off $g \odot f$ is in general not be a convex function, see Figure 4.2 for an example. However, to efficiently combine trade-off functions using Algorithm 4.2, all trade-off functions need to be convex. We solve this problem by representing the optimal non-convex solution through multiple convex partial solutions. Using Corollary 4.5, it is easy to see that the optimal solution can be represented using at most k partial solutions of the form:

$$opt^k_\Delta(x) := \begin{cases} x, & x \leq \Delta^*_k \\ \Delta^*_k, & else \end{cases}$$

The lower bound of all partial solutions is the non-convex optimal solution of $g \odot f$. We discared an earlier approach of computing this non-convex lower bound explicitly and then splitting it up into convex sub-functions, since it required an excessive amount of intersection tests, and for the same reason also had problems with numeric stability.

Before we can use these Lemmas to define an algorithm for composing piecewise trade-off functions, we first need to consider an important feature of piecewise trade-off functions. Since sub-functions of are not required to have the same derivative at segment endpoints, they are generally not differentiable at segement endpoints. Thus the requirement of Lemma 4.3 that both functions need to be differentiable in the local minimum, might not always be true. In this case Lemma 4.3 does not apply, so we need to check all of these points for optimal charging points as well. We do this by keeping track of the left and right derivative of every segment endpoint of f. If we encounter a point where the left and right derivative of f is not the same, we evaluate if charging at this point would lower the current consumption. We can easily verify this by checking the slope of g at the point with the same consumption value. If the slope of g is smaller at this point (the charging rate is higher), we accept the point as an optimal charging point. The special case where g is not differentiable can be easily resolved by always using the right derivative, since starting to charge at this point would use the charging rate of the next sub-function of g not the previous one.

In Algorithm 4.3, we show how we can compute all partial solutions opt^k_Δ by iterating over sub-functions of the piecewise convex trade-off function f and the piecewise-linear convex charging function g at the same time. For this we exploit the fact that the function g consists of $k \in \mathbb{N}$ linear functions with at most k different slopes. Since f is a convex function, there is also at most one sub-function that has the same slope as a sub-function of g, and the sub-functions are sorted increasing in slope (decreasing in rate). We can also exploit that g is convex as well, to simultaneously iterate over f and g. For every sub-functions g_j of g we find the first sub-function f_i of f where the left derivative at the segment start is smaller or equal to the slope of g_i. For the first sub-function f_0 we define the left-derivative as $-\infty$. If the slope of g_j is bigger then the right derivative at the segment start of f_i, we have found a point where the left derivative $\overrightarrow{f}'(\tau)$ and right derivative $\overleftarrow{f}'(\tau)$ of f are not the same, since $\overrightarrow{f}'(\tau) \leq g'_j < \overleftarrow{f}'(\tau)$. This case is not covered by Lemma 4.3 and needs to be checked. For this, we compute the consumption of f_i at its segment start and the x value of the point on the charging function g with the same consumption. If this x value is within the bounds of g_i, we know that the slope of $g(x - \Delta, f(\Delta)$ with $\Delta = \underline{\tau}_{f,i}$ is the slope of g_i. Hence, starting to charge at Δ would charge with the charging rate of g_i, which is better than the rate of f_i. Since there can

Algorithm 4.3 Composing a piecewise trade-off function f with a charging function g

If f is a constant function return $h \leftarrow g(x - \underline{\tau}_f, f(\underline{\tau}))$, otherwise continue.
$h \leftarrow \{\}$
$i \leftarrow 0$
$j \leftarrow 0$
while $i \leq n-1$ and $j \leq k-1$ **do**
  while $g'_j < \overleftarrow{f}'_i(\underline{\tau}_{f,i})$ **do**
    if $g'_j > \overrightarrow{f}'_i(\underline{\tau}_{f,i})$ **then**
      $y \leftarrow f_i(\underline{\tau}_{f,i})$
      $x \leftarrow g^{-1}(y)$
      if $x \geq \underline{\tau}_{g,j}$ and $x < \overline{\tau}_{g,j}$ **then**
        Output $h \leftarrow g(x - \underline{\tau}_{f,i}, f(\underline{\tau}_{f,i}))$
      end if
    end if
    $j \leftarrow j + 1$
  end while
  if $g'_j \leq \overrightarrow{f}'_i(\overline{\tau}_{f,i})$ **then**
    $\Delta^* \leftarrow (f')^{-1}(g'_j)$
    Output $h \leftarrow g(x - \Delta^*, f(\Delta^*))$
    $j \leftarrow j + 1$
  else
    $i \leftarrow i + 1$
  end if
end while
$y_{min} \leftarrow f(\overline{\tau}_f)$
$x_{max} \leftarrow g^{-1}(y_{min})$
while $j \leq k-1$ and $g'_j < \overrightarrow{f}'(\overline{\tau}_f)$ **do**
  if $g'_j > \overleftarrow{f}'(\overline{\tau}_f)$ **then**
    if $x_{max} \geq \underline{\tau}_{g,j}$ and $x_{max} < \overline{\tau}_{g,j}$ **then**
      Output $h \leftarrow g(x - x_{max}, f(x_{max}))$
    end if
  end if
  $j \leftarrow j + 1$
end while

be no other sub-function of f with the same slope as g_i, we can continue with the next sub-function. In the case where we find a sub-function f_i where the slope of g_i is smaller or equal to the left derivative of the segment endpoint, there must be a point on f_i that has the same slope as g_i. We compute this point over the inverse of the derivate and check if the consumption value at this point maps to g_i. If this is the case, we know that Lemma 4.3 applies (both points are differentiable) and we need to return this point as optimal charging point. If we have processed all sub-functions of f, we need to check if the last segment end-point is an optimal charging point. For this we continue to iterate over g until we find the sub-function that maps to the minimal consumption of f. If it exists, we output this point as optimal charging point as well. Algorithm 4.3 uses two nested loops, which might create the appearance that this algorithm is quadratic. However in every step of the inner or the outer loop we either process a sub-function of g or a sub-function of f. Since if f has n sub-functions and g has k sub-functions, there are only $n + k$ sub-functions that can be processed and the algorithm terminates in linear time. For the charging functions we used in this work k is at most 5.

4.3 Dominance Check of Trade-Off Functions

After computing a new tentative trade-off function h, the algorithm needs to determine if this solution can improve the result and thus needs to be inserted in the label set. To determine this, the trade-off functions needs to be checked against the trade-off functions that are already in the label set of a node $v \in V$. We say the piecewise trade-off function f *dominates* the piecewise trade-off function g if:

$$f \preceq g := \forall x \in \mathbb{R}_+ : f(x) \leq g(x)$$

By checking whether $\underline{\tau}_g < \underline{\tau}_f$ we can quickly determine if it is even worthwhile to do a dominance check, since $f(x) = \infty$ for all $x < \underline{\tau}_f$. Since the trade-off functions are piecewise functions, the complexity of this operation depends on the number of sub-functions of trade-off functions. A naive version of the dominance check for two piecewise functions f and g could be to do pairwise dominance checks between all sub-functions of f and g. This algorithm is obviously quadratic in the number of sub-functions of f and g and could miss cases where sub-function of g are only partially dominated by sub-functions of f. However, we can use the fact that all functions are convex monotonically decreasing functions to derive an algorithm that can check whether f dominates g using a coordinated linear sweep over f and g. We visit all interval endpoints x_f and x_g in increasing order, and check for every endpoint x' whether $f(x') \leq g(x')$. If this is not the case f cannot dominate g. If f and g have at most one intersection, it is easy to see that checking segment endpoints is sufficient, since for at least one endpoint x' we would see that $f(x') > g(x')$ However, in the case of two intersections, both intersection points x'_1 and x'_2 could be between two successive segment endpoints. For this case we will require Lemma 4.6 to see which points we need to check as well.

Lemma 4.6. *If two trade-off functions f and g intersect in two points $x'_1 < x'_2$ there exists a point $x^* \in [x'_1, x'_2]$ called the* critical point *for which $f'(x^*) = g'(x^*)$.*

Proof. Since f and g are both monotonically decreasing convex functions, we know that for their derivatives it holds that $f'(x'_1) \leq f'(x'_2)$ and $g'(x'_1) \leq g'(x_2)$. The function f is monotonically decreasing, so we can apply that the slope of a line going through x'_1 and x'_2 gives a bound on the slope at the intersection points:

$$f'(x'_1) \leq \frac{f(x'_2) - f(x'_1)}{x'_2 - x'_1} \leq f'(x'_2)$$

The same is true for the function g. The points x'_1 and x'_2 are intersection points and we see that the slope of the function passing through both endpoints is the same:

$$\frac{f(x'_2) - f(x'_1)}{x'_2 - x'_1} = \frac{g(x'_2) - g(x'_1)}{x'_2 - x'_1}$$

Case 1: f and g are linear functions.

For since f is a linear function it holds that the slope is a constant value $f'(x) = \frac{f(x'_2)-f(x'_1)}{x'_2-x'_1}$, the same is true for the linear function g. By applying the identity we just derived above, we see that we can chose any point as x^*.

Case 2: f is a linear function and g is a hyperbolic function

We set $\alpha := f'(x)$ to the constant slope of the linear function f. Since g is a hyperbolic function, its derivative is continuous. From $g'(x'_1) \leq \alpha \leq g'(x'_2)$ we see that there exists a value $x^* \in [x'_1, x'_2]$ so that $g'(x^*) = \alpha = f'(x^*)$. Using the inverse of the derivate of g we can compute the point τ^* as:

$$x^* = b_g + \sqrt{a_g/(\alpha - c_g)}$$

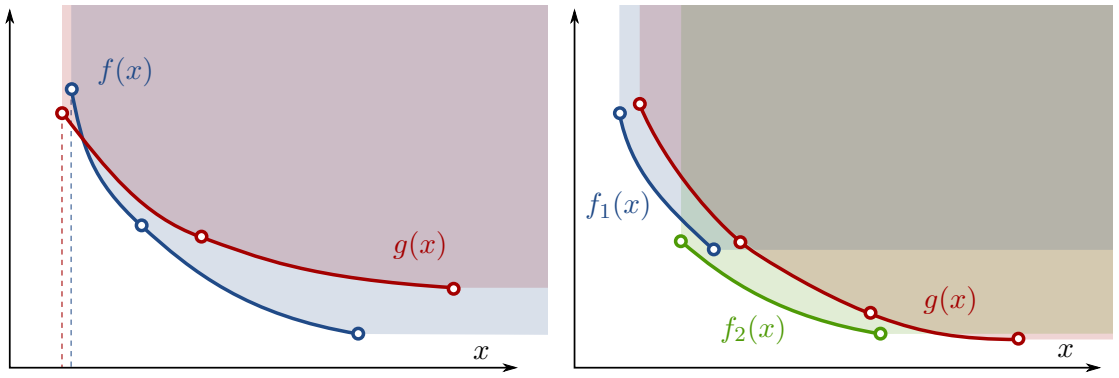

Figure 4.4: Left: Function f (blue) dominates the second segment of function g (red), but no the first. Right: Shows an example where the order of the partial dominance checks matters. The function g does not have a prefix or suffix that is dominated by f_2, however if we first removed the suffix that is dominated by f_1, then f_2 dominates the second segment of g.

Case 3:f is a hyperbolic function and g is a linear function.

This is symmetric to case 2.

Case 4:f is a hyperbolic function and g is a hyperbolic function.

$f'(x) = g'(x)$ if and only if $\frac{-2a_f}{(x-b_f)^3} = \frac{-2a_g}{(x-b_g)^3}$ which is a unique point x^*. We now need to check if $x^* \in [x'_1, x'_2]$. Since x'_1 and x'_2 are the only intersection points of f and g we know for all $x < x'_1$ that $f(x)$ does not equal $g(x)$, and similarly for all $x > x'_2$. If the function f is strictly bigger then g for all $x < x'_1$, then it follows that for all $x > x'_2$: the same is true, otherwise x'_2 would not be an intersection point. From this follows that $f'(x'_1) < g'(x'_1)$ and $f'(x'_2) > g'(x'_2)$, which already shows that the intersection point of both derivatives needs to be in $[x'_1, x'_2]$ and since it is unique, it is the point x^*. We can make an analogous argument for the case that f is strictly smaller then g all $x < x'_1$. See Figure 4.3 for an example of the point x^*, denoted by the vertical slashed black line. □

To maintain correctness in the case of two intersections, we are thus going to compute the critical point x^* and apply Lemma 4.6. If the critical point is not in the shared domain of f and g, we know that both functions cannot have two intersections. Otherwise, we also need to check if the function f dominates g at the critical point $f(x^*) < g(x^*)$.

Partial Dominance

If f is a piecewise trade-off function with $f_0, \ldots, f_n$ being its sub-functions, for $i \leq n$ we call $f_0, \ldots, f_i$ a *dominated prefix* if the piecewise function implied by these sub-functions is dominated another function g. Similarly, for all $0 \leq i \leq n$ we call $f_i, \ldots, f_n$ a *dominated suffix* if the piecewise function implied by these sub-functions is dominated by another function. A *maximal dominated prefix* is the prefix of maximal length that is dominated, a *maximal dominated suffix* is the suffix with maximal length that is dominated. We can compute the maximal dominated prefix using the same algorithm described above, but instead of terminating as soon as we find a dominated sub-function of f, we terminate if we find the first undominated sub-function. To compute the maximal suffix, we need to adapt the algorithm to do the linear sweep by decreasing x coordinate. We can use these prefixes and suffixes to remove unneeded sub-functions, and only keep the undominated parts of f.

Set Dominance

To check whether a piecewise function g is dominated by a whole set of functions $F := f_1, \ldots, f_k$, we implemented a simple heuristic based on the algorithm for determining maximal prefixes and suffixes. For every function f_i we compute the maximal dominated prefix of g and remove it before checking f_{i+1}, analogous we remove the maximal suffix. If g is dominated by F the maximal prefix and suffix will be all of g, otherwise g is partially dominated and we can discard the sub-functions of the dominated prefix and suffix. Note that the order in which the functions in the set f_i are checked matters, for an example see Figure 4.4. If function f_2 is checked before f_1 we would only remove the first sub-function of g. Generally we will check the functions in the order in which their corresponding labels were inserted in the label set, which roughly corresponds to a sorting by the minimal feasible time of each function (depending on the key of the queue). For this sorting we almost never encounter the case that we do not remove dominated parts of g. The problem could be solved by iterating over all endpoints of all functions in the set F at the same time, but we opted for the simpler approach described here.

4.4 Implementation

In this section we want to give some details about the implementation of this algorithm and the relevant data structures. We chose C++17 as implementation language, numbers given here for data type sizes reference these of the compiler GCC 7.1 on an x86 64bit Linux system, if not stated otherwise.

Function Representation

We represent each piecewise function as a vector of sub-functions. Each sub-function stores the minimal and maximal x value on which it is defined. All sub-function are sorted by minimal x-value for efficiently finding the sub-function that is defined for an x value. Since we generally only represent piecewise functions where the sub-function share interval endpoints, it would be possible to only store the minimal x value for each sub-function and infer the maximal x value from the next sub-function. We store both values to simplify the implementation of some algorithms.

Every trade-off function has four parameters a, b, c, d that need to be stored. However, we can use the fact that the values for a and d have disjoint ranges: $a \geq 0$ and $d \leq 0$. Since we know that $a \neq 0 \Rightarrow d = 0$, we can use this fact to encode a and d in the same value and use the sign to determine if the function is hyperbolic or linear. Using `double` values with each 8 bytes for each parameter and the domain endpoints, this yields 40 bytes for each sub-function. Removing the second interval endpoint this could be optimized to 32 bytes.

For continuous piecewise-linear functions we use a more compact storage. For every interval endpoint we store the x and y value of the function and linearly interpolate between them. This reduces the storage for k linear sub-functions from $k \cdot 40$ bytes to $(k+1) \cdot 16$ bytes, since we only need to store two `double` values for each pair of x and y values, but at least two pairs.

Label Sets

To represent paths that offer different trade-offs, every node has multiple label entries that each represent one path. For this purpose, we need a data structure that efficiently represents each of this labels sets. Every label entry contains four values: A piecewise function that represents the trade-off of the path, a piecewise-linear function that represents the Δ values used to construct this trade-off function, the id of the parent node, and the id

of the parent's label entry that was used to generate this path. This amounts to 8 byte for the parent id and parent label entry id, the size needed for the piecewise functions depends on the implementation of the `STL` container `std::vector`, for our compiler that is 24 bytes each for the trade-off function and the Δ function. Note this does not include the memory needed for each sub-function of the two piecewise functions, only data necessary for the implementation of `std::vector`. The total label entry size thus amounts to a minimal size of 56 bytes per label entry, for which 32 bytes are only needed if the path needs to be reconstructed later on. In addition to that, for every trade-off function with k sub-functions we need a total of $k \cdot 40$ bytes and $(k+1) \cdot 16$ bytes for the corresponding Δ function. Since we need to sort each label entry by a given key, we also need 4 bytes per label entry to store that key for efficient sorting.

We divide the label set for every node into settled and unsettled labels (as proposed in [BDG+15]). We keep all unsettled labels in a heap sorted by the same key used for the priority queue Q. When a new label entry is inserted in the heap, we check if the minimal key changes, and if so we check if the new minimum label entry is dominated by the set of settled labels for the node. If the new minimum is dominated, we remove label entries from the set of unsettled labels until the new minimum is not dominated anymore, or there are no unsettled labels anymore. Similarly, when removing the minimum from the heap to settle it, we need to ensure the new minimum is not dominated. In the case the minimum is partially dominated, we remove the maximal dominated prefix and suffix as described in Section 4.3.

Floating Point Precision

A large part of the needed storage for labels is determined by the size of each trade-off function. Since all parameters are stored as floating point values, the precision of the floating point type matters. Using 32 bit floating point values instead of 64 bit floating point values could save up to 50% of the memory needed to store each hyperbolic/linear function. Unfortunately, using 32 bit floating point values quickly lead to large numeric errors when computing the combination two hyperbolic trade-off functions. Especially the computation of domain boundaries for each sub-function of $f \oplus g$ was a major source of numeric instability. If the computation of $\underline{\tau}$ and $\overline{\tau}$ each have a numeric error $\underline{\epsilon}$ and $\overline{\epsilon}$, the size of the domain could increase by up to $|\overline{\epsilon}| + |\underline{\epsilon}|$. When using single precision floating point values, this lead to the introduction of "phantom" solutions with tiny domains. This not only lead to an increase in the length of each piecewise trade-off function, but the effects of this numeric instability amplified when the "phantom" solutions were subsequently combined with other trade-off functions. This could potentially creating trade-off functions that again consist of three sub-functions, two of which are degenerated numerical noise. By changing the slope near domain boundaries, a small error in the domain boundary can also lead to the whole piecewise function to not be convex anymore, which invalidates assumptions in multiple algorithms leading to non-optimal solutions. Hence we settled on using `double` precision floating point numbers for all values.

5. Speedup Techniques

In this chapter we talk about speedup techniques for the base algorithm described in Chapter 4. We limit ourselves only to speedup methods that do not require preprocessing. First, we are going to look into some general properties and requirements of the A^* algorithm. Then, we are going to define a very simple potential that is based on the fastest path to the target t. In some circumstances this potential can underestimate the travel time severely, especially when using charging stations. As a result, we proposed another potential π_Ω in Section 5.3 that incorporates the notion of using charging stations in the lower bound on the travel time. Both potentials require us to compute the distance between a node and the target in various scalar metrics, for example doing a full backwards search from the target node. While this approach is feasible on small graphs, it quickly becomes the bottle neck on larger graphs, like the road networks of Germany, so we discuss in the last Section 5.4 how to compute these potentials on demand, whenever we reach a new node in the network.

5.1 A^* Algorithm

One of the most popular speedup methods for Dijkstra's algorithm is the A^* algorithm, which introduces a potential function to change the order of nodes in the queue. If the potential function $\pi : V \to \mathbb{R}_+$ never over-estimates the distance to the target node $t \in V$, that is $\pi(t) \leq dist(v, t)$, we call it *admissible*. With an admissible potential function π, using $dist(s, v) + \pi(v)$ as key in the priority queue Q yields a correct algorithm. Intuitively the algorithm first relaxes nodes that seem to be on shorter paths to the target node t. In the special case where for all nodes $v \in V$ the potential $\pi(v)$ is equal to the exact distance to the target $dist(v, t)$, the algorithm would only process nodes that are on a shortest path to t. If the potential function fulfills the additional criteria of $\pi(v) - \pi(u) \leq dist(u, v)$ for all edges $(u, v) \in E$ we call it *consistent*. Using an admissible and consistent potential function the A^* algorithm is label setting.

In the context of electric vehicles the travel time to the target depends heavily on the current state of charge, or in our case the current consumption. For this reason, we extend the basic definition of a potential function to a consumption-dependent potential function $\pi(v, y)$. We call a consumption-dependent potential function *admissible* if $\pi(v, y)$ is always a lower bound for $dist(v, t)$ given an initial consumption of y at v. Similarly, we call a consumption-dependent potential function *consistent* if for all edges $(u, v) \in E$ the difference $\pi(u, y) - \pi(v, y + f_{u,v}(\tau)) \leq \tau$ for all feasible travel times $\tau \in [\underline{\tau}_f, \overline{\tau}_f)$.

Since our algorithm that we introduced in Chapter 4 uses multiple labels per node, which each represent an own path to v, we need to adapt the definition of a potential for labels. For this we use a similar construction as Baum et al. [BDG+15] and introduce a label key function $\kappa : L \to \mathbb{R}$ that maps every label $l \in L(v)$ to a key that will be used in the priority queue of our algorithm. For a potential π we also define a trade-off key function $\kappa(f_{s,v}, \tau)$ that maps a trade-off function and a feasible travel time $\tau \in [\underline{\tau}_f, \overline{\tau}_f)$ to a key value.

$$\kappa(f_{s,v}, \tau) := \tau + \pi(v, f_{s,v}(\tau))$$

We call a label key function *consistent* if for all nodes $v \in V$ and all labels $l \in L(v)$, the label key function $\kappa(l)$ is smaller than all trade-off keys $\kappa(f, \tau)$ for its corresponding trade-off function f. For the case of using a zero potential function, the key function $\kappa(l) := \kappa(f, \underline{\tau}_f) = \underline{\tau}_f$ is always consistent. If a consistent label key function κ is used, the A^* algorithm is label setting, since its value is a valid lower bound for all trade-offs it represents.

5.2 Fastest-Path Potential Function

Given a label $l \in L(v)$ we have so far used the minimal feasible travel time $\underline{\tau}_{s,v}$ of the label's trade-off function $f_{s,v}$ as key for the priority queue in our base algorithm. The natural extension of this would be to use the unconstrained travel time as potential function $\pi_t(v, y) := dist_t(v, t)$. It is easy to see that the unconstrained travel time to the target t is always a lower bound, since every constrained shortest path is also valid without constraints, so this potential is admissible. Using the fact that $dist_t$ fulfills the triangle inequality, is also easy to see that the potential is consistent for all $\tau \in [\underline{\tau}_f, \overline{\tau}_f)$:

$$\pi_t(u, y) - \pi_t(v, y + f_{u,v}(\tau)) = dist_t(u, t) - dist_t(v, t) \leq dist_t(u, v) \leq \tau$$

To compute this potential function we run a one-to-all backwards search from the target t using the unconstrained minimal travel time as cost before starting the trade-off search. In Section 5.4 we discuss how we can circumvent running a full backwards search for every query.

The potential π_t leads to great speedups if the constrained shortest path mostly coincides with the shortest path. However, this potential breaks down when we reach the edge of the battery capacity where the battery constraints begin to limit the travel time of all labels on fastest paths to t. The unsettled labels from charging stations are only considered very late in the search, since their minimal travel time to the target is much higher than for other non-charging labels. This is amplified by the fact that charging stations are not generally on the fastest path, but require a detour, for example leaving the motorway and driving to a parking lot and back. In some cases these detours would even form a cycle in the route, which is never the case for an unconstrained shortest path.

5.3 Ω-Potential Function

To solve the problem of paths via charging stations being found very late in the search, we adapted an approach introduced by Baum et al. in [BDG+15]. The basic idea is to incorporate the time needed to lower the consumption until a path becomes feasible. Lowering the consumption in our model means we need to either drive slower, or use a charging station. If we assume that we can recharge at every node, and $\alpha_{min} < 0$ is the best charging rate over all charging stations, we can compute a lower bound for lowering the consumption by Δy as $-\Delta y / \alpha_{min}$. However, we also have the option to arrive later by driving slower, reducing the overall consumption as well, which might be faster than stopping to charge. By computing the first time τ^* where it would make sense to start

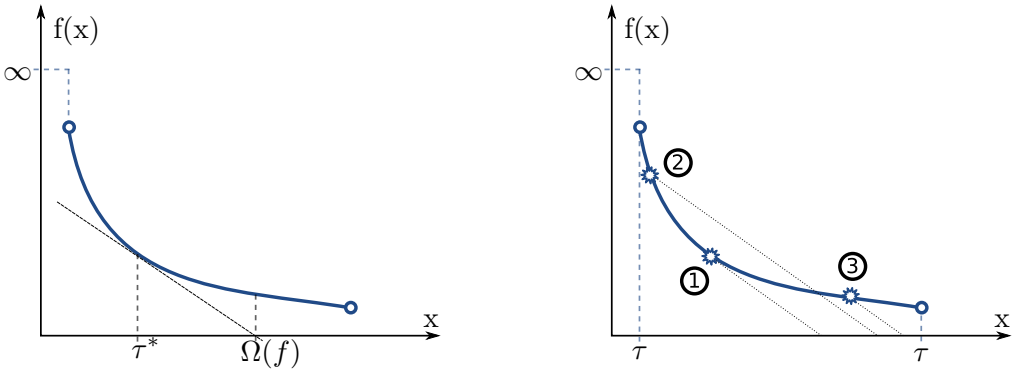

Figure 5.1: Left: Visualizes the value $\Omega(f)$ of a consumption function f, the time τ^* is the time where f has the same slope as the minimal charging rate α_{min} denoted by the dotted line. Right: Visualizes the time needed to charge at best charging rate α_{min} until the battery is full for several points on the function f, it is clear that the point 1 is the value that minimizes this.

charging instead of driving slower, that is the time-consumption trade-off rate $f'(\tau^*)$ is greater or equal to the best charging rate α_{min}, we can compute a valid lower bound on the time needed to arrive at the node with a consumption of y. For this we need to consider two cases, the first one being that it is overall faster to drive slower. This seems paradoxical at first, but the lowered consumption might increase the range just enough to reach the target, or avoid a detour to the nearest charging station. In this case the time needed to reach the consumption y by driving slower $f^{-1}(y)$ will be smaller than the time τ^* where it would pay off to start charging. If $y > f(\tau^*)$ using the charging station would yield the best rate, so we can compute the time needed to charge to y as:

$$\tau^* - \frac{y - f(\tau^*)}{\alpha_{min}}$$

See Figure 5.1 for an illustration of multiple trade-offs between driving slower and charging. The point `1` at τ^* is the optimal point to start charging, any earlier point (e.g. point `2`), or later point (e.g. point `3`), will yield a higher overall time.

In this section we will use these insights to adapt the π_Ω potential used in Baum et al. [BDG+15] to trade-off functions. To do this we first define a function that maps a trade-off function to the best trade-off between driving slower and charging the remaining consumption. We call this function Ω function, similar to the ω weight on scalar values introduced in [BDG+15].

$$\Omega(f) := \min_{\tau} \tau - \frac{f(\tau)}{\alpha_{min}}, \text{ with } \tau \in [\underline{\tau}_f, \overline{\tau}_f)$$

We then use this function to define a scalar cost for every edge $(u, v) \in E$:

$$dist_\Omega(u, v) := \min_{\tau} \tau - \frac{f_{u,v}(\tau)}{\alpha_{min}}, \text{ with } \tau \in [\underline{\tau}_{f_{u,v}}, \overline{\tau}_{f_{u,v}})$$

We turn this into metric for all nodes by setting $dist_\Omega(v, w)$ to the value of the shortest path between v and w using $dist_\Omega$ as cost. We use this metric to get a lower bound on the time needed to reach the target t with full battery. To show that $dist_\Omega$ is a lower bound we going to use an inductive argument. We know by definition that $dist_\Omega(u, v) \leq \Omega(f_{u,v})$ for very edge $(u, v) \in E$, if we can prove that this is true for all trade-off functions computed by applying the operators $\oplus$ and $\odot$ it is true for trade-offs.

For all trade-off functions f and charging functions g the composition operator $\odot$ does not change the Ω value, since the optimal charging point τ^* is precisely defined to be the first point where charging would make sense. That means the value $\Omega(g \odot f)$ is equal to $\Omega(f)$, since the optimal combination $(g \odot f)(x)$ is equal to $f(x)$ for all times $x \leq \tau^*$. However, proving how the function Ω works with the combination operator $\oplus$ is not trivial, hence we are going to prove the following Lemma 5.1.

Lemma 5.1. *For all trade-off functions f and g it holds that $\Omega(f \oplus g) = \Omega(f) + \Omega(g)$.*

Proof. If $f'(x) < \alpha_{min}$ for all feasible times $x \in [\underline{\tau}_f, \overline{\tau}_f)$, we see that the value $\Omega(f)$ is equal to $\overline{\tau}_f - \frac{f(\overline{\tau}_f)}{\alpha_{min}}$. Similarly, if $f'(x) > \alpha_{min}$ for all feasible times $x \in [\underline{\tau}_f, \overline{\tau}_f)$, we see that the value $\Omega(f)$ is equal to $\underline{\tau}_f - \frac{f(\underline{\tau}_f)}{\alpha_{min}}$. The same is true for the function g.

If both $f'(x) < \alpha_{min}$ and $g'(x) < \alpha_{min}$ for all respectively feasible x, it is easy to see that $\Omega(f \oplus g) = \underline{\tau}_f + \underline{\tau}_g$ since the slope of $f \oplus g$ will also be strictly smaller than α_{min}, which proves the lemma for this case. We can apply the same argument for the case where both slopes of the function f and the function g are strictly bigger then α_{min}.

Lets assume there exists $\tau_f^* \in [\underline{\tau}_f, \overline{\tau}_f)$ for which the slope at that point $f'(\tau^*)$ is equal to α_{min} and the slope of the function g is strictly smaller then α_{min}. As per definition of the operator $\oplus$ and $g'(x) < f'(\Delta)$ for all feasible times $x \in [\underline{\tau}_g, \overline{\tau}_g)$ and times $\Delta \geq \tau^*$, we see that $opt_\Delta(x) = x - \overline{\tau}_g$ for all times $x \geq \underline{\tau}_g + \tau^*$. As a result, optimal combination $(f \oplus g)(x)$, has the value $f(x - \overline{\tau}_g) + g(\overline{\tau}_g)$ for all $x \geq \underline{\tau}_g + \tau^*$. From this follows that the slope of the function $f \oplus g$ is $f'(x - \overline{\tau}_g)$ for all $x \geq \underline{\tau}_g + \tau_f^*$. Consequently, the slope of this function at the point $x = \overline{\tau}_g + \tau_f^*$ is α_{min}, which proves the lemma for this case. We can apply the symmetric argument for the function g by swapping f and g. Similarly, we can adapt the same reasoning to the cases where the slope of g is strictly smaller.

What is left to show is the case where we can find values τ_f^* and τ_g^* that are the first feasible times where $f'(\tau_f^*) = \alpha_{min}$ and $g'(\tau_g^*) = \alpha_{min}$. For $\Delta = \tau_f^*$ and $x = \tau_g^* + \tau_g^*$ we can see that:

$$f'(\Delta) = \alpha_{min} = g'(x - \Delta)$$

As per Section 4.1 that means the point (Δ, x) is a local minimum of $f \oplus g$ and $opt_\Delta(x) = \Delta$. Consequently, the slope of $(f \oplus g)(x)$ is equal to $g'(x - \Delta) = \alpha_{min}$. □

It is easy to see that applying battery constraints to a trade-off function f can only increase the value $\Omega(f)$, since the feasible points of the constrained funciton $con_{bat}(f)$ are also feasible on the unconstrained function f. From this follows that $dist_\Omega(v, t)$ is a lower bound to $\Omega(f_{v,t})$ where $f_{v,t}$ is the optimal trade-off for all paths from v to t and computed by successively applying $\oplus$, $\odot$ and con_{bat} operations.

Similarly to $dist_t$ we could try to use $dist_\Omega(v, t)$ as potential for $l_v \in L(v)$. However, if we can reach t from v on the fastest path $p \in P(G)$ without violating the battery constraints, then $dist_\Omega(v, t)$ is not admissible:

$$dist_\Omega(v,t) = \sum_{(u,v)\in p} \min_\tau \tau - \frac{f_{u,v}(\tau)}{\alpha_{min}} > \sum_{(u,v)\in p} \min_\tau \tau = \sum_{(u,v)\in p} \underline{\tau}_{u,v} = dist_t(v,t) = dist(v,t)$$

Let the value $\underline{y}_{v,t}$ be the minimal consumption of a path from $v \in V$ to the target t, and $y_{s,v}$ the consumption of a path from s to v. If the minimal consumption $\underline{y}_{v,t}$ is bigger then the remaining battery charge $M - y_{s,t}$, at some point we will have to spend time charging the surplus consumption $y_{s,v} + \underline{y}_{v,t} - M$. To regain the energy $y_{s,v} + \underline{y}_{v,t} - M$ we can reduce the energy consumption $y_{s,v}$ on the path from s to v, regain it through charging

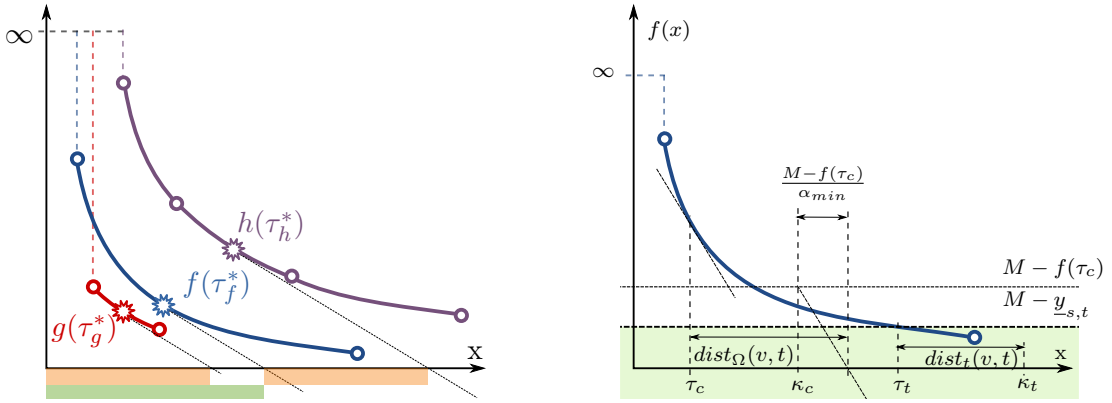

Figure 5.2: Left: Omega value of two consumption functions f and g, the omega value of the combined function $h := f \oplus g$ is the sum of both. Right: Relationship between the key of the first feasible point k_1 and the key of the optimal charging point k_2.

somewhere on the path from v to t, or potentially a combination of both. Given the best charging rate of $\alpha_{min} < 0$ we can compute the minimal time required for charging the remaining energy as $(y_{s,v} + \underline{y}_{v,t} - M)/\alpha_{min}$.

We can use these insights to define our consumption-dependent potential $\pi(v, y)_\Omega$:

$$\pi_\Omega(v, y) := \begin{cases} \pi_t(v), & \text{if } y \leq M - \underline{y}_{v,t} \\ dist_\Omega(v, t) + \frac{M-y}{\alpha_{min}}, & otherwise \end{cases}$$

If we can reach the target t with a consumption y without using charging, we assume we can get there on the fastest path. Obviously this is very optimistic, since a lower bound $\pi_t(v)$ on the travel time of the time-optimal path might be much lower than the travel time of the energy-optimal path with a consumption of $\underline{y}_{v,t}$. If the minimal overall consumption to the target t would exceed the battery capacity, it is clear that we need to charge somewhere on the path. In that case we can use the $dist_\Omega$ metric to compute a lower bound for the time to reach the target and fully charge the battery. Since the assumption of fully charging the battery is too pessimistic, we need to subtract the time we already "saved" by not having to charge the remaining battery capacity $M - y$ at node v. What is left to show it that π_Ω is actually an admissible and consistent consumption-dependent potential, which we will do in the following Lemma 5.2 and Lemma 5.3.

Lemma 5.2. *π_Ω is an admissible label potential, that means for all $v \in V$ and all consumptions $0 < y \leq M$ the potential $\pi_\Omega(v, y)$ is a lower bound for the actual travel time $dist(v, t)$ with initial consumption y at v.*

Proof. Let $v \in V$ be any node and y any valid consumption value.

If the consumption y is smaller or equal to the maximal consumption $M - \underline{y}_{v,t}$ without charging, then the potential $\pi_\Omega(v, y)$ is equal to $\pi_t(v)$ We have already seen that this is an admissible potential, so the lemma is true for this case.

If the consumption y is bigger than the maximal consumption without charging $M = \underline{y}_{v,t}$, then we can prove that we need to charge on the path to the target t. In this case the potential has the following form:

$$\pi_\Omega(v, y) = dist_\Omega(v, t) + \frac{M - y}{\alpha_{min}}$$

With $dist_\Omega(v,t) \leq \Omega(f_{v,t})$ for the optimal trade-off $f_{v,t}$ for all paths from v to t we see that:

$$\pi_\Omega(v,y) \leq \Omega(f_{v,t}) + \frac{M-y}{\alpha_{min}} \leq \underline{\tau}_{v,t} - \frac{f_{v,t}(\underline{\tau}_{v,t})}{\alpha_{min}} + \frac{M-y}{\alpha_{min}}$$

Since $f_{s,t}(\underline{\tau}_{s,t}) < \underline{y}_{s,t}$, $\alpha_{min} < 0$ and $\underline{y}_{s,t} - M + y > 0$:

$$\pi_\Omega(v,y) \leq \underline{\tau}_{v,t} - \frac{\underline{y}_{s,t} - M + y}{\alpha_{min}} \leq \underline{\tau}_{v,t}$$

Which concludes the proof for this case.

□

Lemma 5.3. *The potential π_Ω is a consistent. For every edge $(u,v) \in E$ the difference of potentials $\pi_\Omega(u,y) - \pi_\Omega(v, y + f_{u,v}(\tau))$ never over-estimates the actual travel time $\tau \in [\underline{\tau}_f, \overline{\tau}_f)$.*

Proof. Let $(u,v) \in E$ be any edge, $f_{u,v}$ its corresponding trade-off function and $\tau \in [\underline{\tau}_f, \overline{\tau}_f)$ any feasible travel time. If there is a feasible path from u to t the consumption y is smaller than $M - \underline{y}_{u,t}$. In this case the potential $\pi_\Omega(u,y)$ is equal to $\pi_t(u)$ and we can easily derive an upper bound using the fact that π_t is consistent:

$$\pi_\Omega(u,y) - \pi_\Omega(v, y + f_{u,v}(\tau)) = \pi_t(u) - \pi_\Omega(v, y + f_{u,v}(\tau)) \leq \pi_t(u) - \pi_t(v) \leq \underline{\tau}_f \leq \tau$$

If there is no feasible path from u to t, we need to charge on the path to t. In this case the potential $\pi_\Omega(u,y)$ is equal to:

$$dist_\Omega(u,t) + \frac{M-y}{\alpha_{min}}$$

Case 1: There is a feasible path from v to t.

In this case the consumption $y + f_{u,v}(\tau)$ is strictly smaller than $M - \underline{y}_{v,t}$. From $M - y < \underline{y}_{u,t}$ we see that:

$$\underline{y}_{v,t} < M - y - f_{u,v}(\tau) < \underline{y}_{u,t} - f_{u,v}(\tau)$$

Since $\underline{y}_{v,t} + f_{u,v}(\tau) < \underline{y}_{u,t}$ is a contradiction to $\underline{y}_{u,t}$ being minimal, this case can never occur.

Case 2: There is no feasible path from v to t.

In this case the consumption $y + f_{u,v}(\tau)$ is strictly greater than $M - \underline{y}_{v,t}$ and we see that $\pi_\Omega(u, y - f_{u,v}(\tau))$ is equal to:

$$dist_\Omega(v,t) + \frac{M - y - f_{u,v}(\tau)}{\alpha_{min}}$$

Together with $\pi_\Omega(u,y)$ this yields:

$$dist_\Omega(u,t) + \frac{M-y}{\alpha_{min}} - dist_\Omega(v,t) - \frac{M - y - f_{u,v}(\tau)}{\alpha_{min}} = dist_\Omega(u,t) - dist_\Omega(v,t) + \frac{f_{u,v}(\tau)}{\alpha_{min}}$$

Since $dist_\Omega$ is a metric we see that $dist_\Omega(u,t) - dist_\Omega(v,t) \leq dist_\Omega(u,v)$, which we can use to derive another upper bound.

$$dist_\Omega(u,v) \leq \Omega(f_{u,v}) \leq \tau - \frac{f_{u,v}(\tau)}{\alpha_{min}}$$

Which finally yields $\pi_\Omega(u,y) - \pi_\Omega(v, y - f_{u,v}(\tau)) \leq \tau - \frac{f_{u,v}(\tau)}{\alpha_{min}} + \frac{f_{u,v}(\tau)}{\alpha_{min}} = \tau$.

□

Using the consumption-dependent potential $\pi_\Omega(v, y)$ we now define a consistent key function:

$$\kappa(l_v) := \min_\tau \kappa(f_{s,v}, \tau), \text{ where } \tau \in [\underline{\tau}_f, \overline{\tau}_f)$$

This key function is consistent by definition, but quite useless for practically computing the actual consistent key. For this purpose we are going to prove the next Lemma 5.4.

Lemma 5.4. *Let τ_t to be the first time with a feasible consumption $f^{-1}(M - \underline{y}_{v,t})$, and τ_c be the first time charging would pay off $(f')^{-1}(\alpha_{min})$. We set $\tau'_t := \max\{\underline{\tau}_f, \min\{\overline{\tau}_f, \tau'_t\}\}$, and $\tau'_c := \max\{\underline{\tau}_f, \min\{\overline{\tau}_f, \tau_c\}\}$. The consistent key function is equal to the minimum of the keys $\kappa_t := \kappa(f_{s,v}\tau'_t)$ and $\kappa_c := \kappa(f_{s,v}, \tau'_c)$.*

Proof. If $f_{s,v}(\tau) \leq M - \underline{y}_{v,t}$ then the trade-off key function evaluates to:

$$\kappa(f_{s,v}, \tau) = \tau + dist_t(v, t)$$

It is easy to see that for this case the minimal value τ for which $f_{s,v}(\tau) \leq M - \underline{y}_{v,t}$ minimizes $\kappa(f_{s,v}, \tau)$, which is τ_t.

If $f_{s,v}(\tau) > M - \underline{y}_{v,t}$ then the trade-off key function evaluates to:

$$\kappa(f_{s,v}, \tau) = \tau + dist_\Omega(v, t) + \frac{M - f_{s,v}(\tau)}{\alpha_{min}} = \tau - \frac{f_{s,v}(\tau)}{\alpha_{min}} + \frac{M}{\alpha_{min}} + dist_\Omega(v, t)$$

Per definition of $\Omega(f_{s,v})$ this is minimized by $\tau^* = \tau_c$. □

An illustration on how this key function computes a consistent key for a trade-off function f can be found in Figure 5.2 (right). We first determine the first feasible point τ_t and the optimal charging point τ_c and evaluate π_Ω at these positions. We illustrate the case where the consumption at τ_c is too hight to reach the target, in which case the potential π_Ω evaluates to the Ω-distance to t, minus the time needed to charge the remaining battery charge $M - f(\tau_c)$ at charging rate α_{min}. Per definition of τ_t the potential π_Ω evaluates to the unconstrained shortest path distance $dist_t$. Adding both values yields κ_c and κ_t respectively, in this case κ_c is smaller and thus the consistent key for the trade-off function.

Depending on the distribution of charging stations this potential can be very optimistic. Not only do we assume that we can charge at every node along the path, but also at the best charging rate. In most scenarios the best charging rate will be that of a super charger with about $120kW$ charging power. These charging stations are relatively sparse, as such we will often encounter the case where the actual available charging power of all charging stations within reach is much lower.

5.4 Lazy Potential Generation

In the previous two sections we assumed that we can run a full one-to-all backwards search on the whole graph to determine our potentials. This can be extremely wasteful on larger graphs, such as Germany where a single (unaccelerated) backwards search can take up to 500 milliseconds. This becomes even less feasible in the case of the Ω-Potential since we would need to run three full backwards searches from the target for every query. One approach to circumvent the need to re-run this backward search for every query would be to use an approach that lets us pre-compute the potential. Naively pre-computing the exact potential for every possible node quickly becomes unfeasible since it would require $|V|^2$ memory to store, which already would be more then 3 Terrabyte just for a graph with one million nodes. If it suffices to have a lower bound for the metric, we could use

a preprocessing based methods like ALT [GH05] to compute pre-compute an admissible potential. However, this could lead to the quality of each potential decreasing even more, so it would be interesting to find method that does not decrease the potential quality, but computes less values.

For this purpose we adapted the idea from Baum et al. [BDG+15] to compute each potential in a *lazy* manner: For every distance value $dist(v, t)$ we need to compute, we start by running Dijkstra's Algorithm to find a path from t to s on the reverse graph. During that search we keep track of which nodes have already been settled and thus have an accurate value $dist(v, t)$. When we reach s from t, we do not clear the queue, but preserve its state for the duration of the query. Every time we need to know the value $dist(v, t)$ for an unsettled node, we just resume the Dijkstra's Algorithm using the state preserved through the queue. Again, we terminate once this node has been settled, preserving the state of the queue.

This approach works well for computing the lower bound on the travel time to the target $dist_t$, but for $dist_\Omega$ and $dist_c$ Dijkstra's Algorithm is not label setting since they include negative edge costs. For this purpose we adapted the potential shifting approach from Sachenbacher et al. [SLAH11] based on the potential energy of each node in the graph, that ensures we will always use more energy to drive uphill then we could save while driving downhill. First we need to compute a value $\alpha \in \mathbb{R}_+$ for which the scaled height difference of every edge $(u, v) \in E$ is a lower bound of the edge cost:

$$\alpha(h(v) - h(u)) \leq dist(u, v)$$

By setting $dist'(u, v) := dist(u, v) + \alpha h(u) - \alpha(v)$ we obtain a strictly positive cost function that we can use for the lazy backward search. Given the path cost $dist'(v, t)$ we can extract the path cost in the original metric using $dist'(v, t) - \alpha h(u) + \alpha h(v)$.

6. Heuristics

Since computing approximations does in many cases dramatically reduce the running time of algorithms, we will investigate the computation of paths that are not optimal but "good enough" in this chapter. To formalize this criteria, we call a path $p \in P(G)$ an *ϵ-approximation* of the shortest path $p' \in P(G)$ if and only if the path cost $w(p)$ is lower than $(1+\epsilon)w(p')$. In the case of solving the *Constrained Shortest Path* problem this would mean the approximate solution has a minimal feasible travel time that is at most $(1+\epsilon)$-times longer then the optimal solution. Note that we do not require any quality constraints on the consumption of the approximate path (other than it is a feasible path), so we might find paths that are only slightly slower but have a much higher consumption.

In this chapter we will investigate multiple approaches to incorporate heuristics in our algorithm. In Section 6.1 we introduce the concept of approximate dominance checks. In Section 6.2 we touch on the possibility to simplify the model, rather then our algorithms. For this purpose we consider various ways to modify the consumption model of the graph, or modify the set of charging stations.

6.1 Epsilon-Dominance

A natural way to implement the notion of ϵ-approximate paths is to redefine what it means to be better or equally as good as another path. In the scenario of trade-off functions we use dominance checks to decide if a path could offer a better trade-off than another path. By changing the dominance check to be biased towards rejecting trade-offs as dominated, we can potentially reduce the number of labels for each node. For this purpose we define the values ϵ_x and ϵ_y that determine the "slack" of the dominance function. We define a function f to (ϵ_x, ϵ_y)-*dominate* a function g if and only if for all times x the function $f(x)$ has a lower consumption then the function $g(x-\epsilon_x)+\epsilon_y$. This is analogous to shifting the function time-consumption trade-off function g to the right by a time ϵ_x and up by a consumption ϵ_y before doing the dominance test.

Using two ϵ parameters has the advantage the we can tune the amount of slack we want to allow for both duration and consumption independently. Since the time x (in seconds) and consumption y (in Wh) typically have very different magnitudes the ϵ values for each axis will be very different too. It is easy to see that given a value ϵ_y and a battery capacity M, we can only have at most M/ϵ_y settled labels in each node set. For large values of M that can still be much larger then the typical number of labels per node, but it ensures that

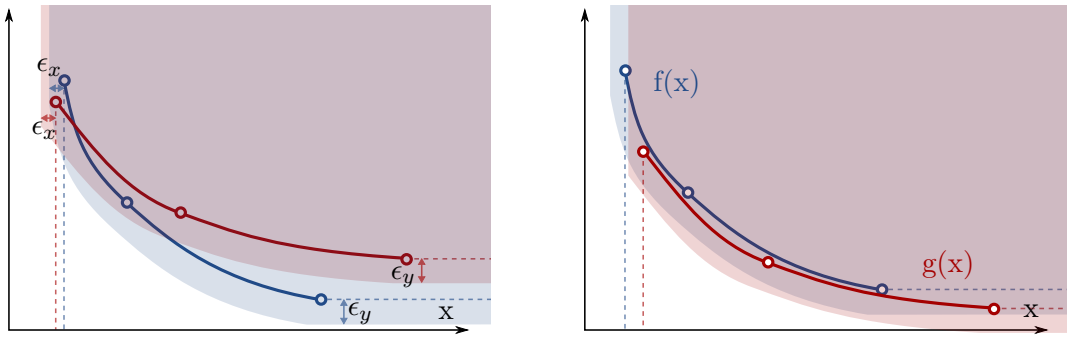

Figure 6.1: Left: Function f (blue) (ϵ_x, ϵ_y)-dominates function g (red) even though $\tau_g < \tau_f$. Right: The function f dominates the function g only using the ϵ_y-value.

outliers do not explode in size. Limiting the amount of settled labels per node is beneficial to the running time of our algorithm, since we perform a dominance test with every settled label, before we allow an unsettled label to be settled. Figure 6.1 visualizes the impact of the values ϵ_x and ϵ_y on the dominance check of two functions f and g. The function f would not dominate g in the exact case since the minimal feasible travel time $\underline{\tau}_g$ of the functions g is lower then the minimal feasible travel time of f. However, in the case of (ϵ_x, ϵ_y)-dominance f can dominate g, possibly reducing the size of the label set.

6.2 Simplified Models

Since our general model aims to be as realistic as possible, it is interesting to investigate which aspects can be simplified without sacrificing a lot of accuracy. For the first simplification we can consider that in our model we allow for driving slower to increase the range of the electric vehicle. Previous works generally assumed a fixed consumption and travel times for every edge. Combining constant consumption functions with hyperbolic time-consumption trade-off functions indeed is a cheap operation that amounts to a few addition operations, since it only shifts the trade-off function to the right and adds a constant consumption. Hence, modeling more edges as constant functions could improve the running time of our algorithm. Since we are interested in routes with minimal travel time, an easy way to convert a hyperbolic function f to a constant function is to simply use the minimal travel time $\underline{\tau}_f$ and associated maximal consumption $f(\underline{\tau}_f)$ as parameters.

Limiting Trade-Offs

The critical aspect is which trade-offs should be converted to constant values. For this we consider when using trade-offs is necessary. Since the battery of a typical electric vehicle is sufficient for all short-range routes, we will only use trade-offs for long-distance routes. Road networks have the property that long-distance routes typically use certain roads, the arterial network, for the biggest portion of the route. In Germany for example, all fastest paths from Stuttgart to Berlin will almost entirely consist of motorways, except for a few exceptions at the beginning and end of the route. Hence, a sensible simplification of the model would be to only allow trade-offs on the arterial network, but not on the roads leading there. These roads have the property that the maximum speed is generally very high, above 100 km/h, so we will only allow time-consumption trade-offs above 100 km/h. In Chapter 3 we saw that the consumption of an electric vehicle is quadratic in the speed, so we would still expect to see gains for lowering the speed from 130 km/h to 100 km/h.

Since our trade-off model uses hyperbolic functions, there are strong effects of diminishing returns: After a certain point reducing the consumption only marginally will result in a large increase in travel time. Trade-off functions have generally a negative slope, so we

are interested in limiting the maximal slope (or minimal rate). We also note that the combination of two trade-off functions will be a function whose maximal slope is the same is the maximal slope of both functions that were combined. Thus, we can hope to limit the overall maximal slope by enforcing it on all edge cost functions in the graph. However, we can also introduce linear segments through charging functions with a potentially higher slope. If assume that we would rather use a charging function than driving slower, we can use the worst maximal charging rate of all charging stations to limit the slope of trade-off functions for each edge.

Simplifying Trade-Off Function

As part of our efforts to incorporate charging functions into the trade-off function model, we extended all operations to handle linear functions. Operations involving linear functions generally have a lot less constant overhead. For example, combining them optimally only requires us to compare the slope, and dominance checks can be done just based on doing a linear sweep over the segment endpoints. We want to constrain ourselves to only use a single linear function to approximate a hyperbolic function, since our algorithm assumes the graph cost is a single trade-off function. A simple method to approximate the hyperbolic function using a single linear function is to use the linear function that is the secant of the minimal travel time trade-off $(\underline{\tau}, f(\underline{\tau}))$ and the maximal travel time trade-off $(\overline{\tau}, f(\overline{\tau}))$. Since this is an upper bound on the consumption, routes that we calculate using this consumption model will still be correct in the sense that they do not exceed the battery capacity. However, this simplification might be a very pessimistic estimate on the consumption, leading to some paths not being found.

Removing Bad Charging Stations

Our charging station dataset includes various charging stations that have a charging rate below 22 kW, which means it can take over two hours just to re-charge to 80% of the battery capacity. This suggests that it almost always pays off to either charge more often at other charging stations with higher charging rates, or to drive slower. However, depending on the state of charge of the battery these slow charging stations can still be important to reach the target, so we need to include them in the exact algorithm. For most queries, this will introduce a lot of useless solutions that will never be used, or even settled. As a heuristic approach to decrease the number of labels, we can thus try to remove all charging stations with a charging rate below 22 kW.

7. Evaluation

In this chapter we will present the empirical evaluation of our model and our algorithms. In Section 7.1 we outline the general experimental setup and datasets that were used to obtain these results. In Section 7.2 we verify the general analytical operations that we need for our base algorithm using a numeric approach. Then we evaluate the correctness by empirically comparing it to an extended version of the BSP [BDHS$^+$14] algorithm. Following the verification, we will dissect the performance of our algorithm on various datasets and settings in Section 7.3. After that we evaluate the quality and performance trade-offs of the heuristics we proposed in Chapter 6. In Section 7.5 we then investigate the quality of the routes that our algorithm generates based on several metrics such as the number of speed changes. In the last Section 7.6 we show-case a few example queries and related metrics, for example the search space of an example query.

7.1 Setup

All measurements were done on two identical systems, each using an `Intel Xeon E5-1630v3` processor with 4 cores clocked at 3.7 GHz and 128 GB of memory. All numbers we report here were obtained by only using one core at the time. We used several datasets from multiple sources for our evaluation.

PTV AG Germany

Our base dataset was the German road network provided by PTV AG with about 4 692 091 nodes and 10 805 429 edges (2 448 306 non-constant, 1 454 845 negative). For various smaller measurements we use a much smaller road network of Luxembourg that only has 36 457 nodes and 81 950 edges (36 680 non-constant, 17 233 negative). The datasets only contain few degree 2 nodes for visualization and are strongly-connected. The maximum speed is derived from the free flow speed that is present in the data. The minimum speed is derived from the road class.

OpenStreetMap

We use a smaller datasets based on data from OpenStreetMap, an extract of Switzerland. Our Switzerland road network has 2 279 877 nodes and 4 951 699 edges (576 284 non-constant, 1 948 041 negative). Since the OpenStreetMap dataset contains a very high number of degree two nodes due to fine road geometry modeling, we preprocessed the

dataset to remove nodes of degree two. Contracting a node can change the slope of the adjacent edges, so we only contracted nodes $v \in V$ of degree two where both neighbor edges (u, v) and (v, w) have either a decreasing slope (both lead uphill) or have a decreasing slope (both lead downhill). Whenever possible we use the appropriate fields `maxspeed` and `minspeed` for minimum and maximum speed of the road, but in cases where this data was absent we inferred the speeds from the road class. Our dataset does not adjust the height profile for tunnels or other underground roads, which makes it significantly harder for some queries on Switzerland, since some tunnels have an excessive consumption. With a more correct modeling of tunnels we expect significantly faster query times, preliminary results show that contracting the tunnels improves the running time by about factor three. Correct elevation data modeling on tunnels also reduces the size of the road network in Switzerland significantly, since we are able to contract more degree two nodes that do not have an elevation profile.

Consumption Data

The consumption data was obtained from the PHEM (Passenger Car and Heavy Duty Emission Model) datasets [HRZL09] provided by the Graz University of Technology, that was already used in [BDG+15] and [BDWZ17]. We used the same consumption model parameters based on a Peugeot Ion, derived from this dataset, that were used in [BDWZ17].

Elevation Data

The elevation data was derived from the SRTM (Shuttle Radar Topology Mission) v3.1 1-arc second void-filled dataset kindly provided by OpenTopo (`opentopo.sdsc.edu`). We determined the elevation for each node from its geographical location using bilinear interpolation of the raster-based elevation profiles. Using the height h_v of every node v we calculated the slope of every edge segment (u, v) as $(h_v - h_u)/l_{u,v}$ where $l_{u,v}$ is the distance of u and v computed using the Haversine formula. Since the elevation data can be quite noisy, we limited the minimum slope to -10% to avoid unrealistic energy gains for driving downhill.

Charging Stations

We use two charging station datasets. The first dataset was obtained from ChargeMap and only contains the coordinate and a coarse classification for charging stations into `slow`, `accelerated` and `quick`. We assume a charging rate of 3.7 kW for the slow stations, a charging rate of 22 kW for the accelerated stations and a charging rate of 44 kW for the quick charging stations. This dataset is the same set of charging stations used in [BDG+15] but we refrained from assigning charging rates randomly based on a pre-defined distribution, preserving the original properties of the dataset. For Germany the dataset contains 290 charging stations with a charging rate of 44 kW, 1076 charging stations with a charging rate of 22 kW, and 749 charging stations with a charging rate of 3.7 kW. Note that there are no super-charger stations for this set of charging stations. To compensate for not introducing artificial super-chargers, we used a second dataset of charging stations that contains all Tesla-compatible charging stations in Europe, including super-chargers. This dataset can be obtained from `tesla.com` as a JSON object and contains the charging rate for each charging station in kW. For Germany it contains 59 charging stations with a charging rate of 120 kWh, the super-chargers, 262 charging station with a rate of 22 kW, and 154 charging stations with mixed charging rates below 17 kWh.

7.2 Correctness

We used two approaches to verify the correctness of our proposed algorithm. The first approach uses a numerical method to check whether we really determine the optimal

values when combining trade-off functions, or composing charging functions with trade-off functions. The second approach uses a simple sampling based implementation that serves as reference for comparing the travel times calculated by our algorithm.

Numerical Verification

To numerically verify our algorithms, we first implemented them in Python using the `NumPy` library. Since both the combine operation $\oplus$ and the compose operation $\odot$ are essentially minimization problems, we only had to provide a definition of the minimization problem as function $h(x, \Delta)$. Given two functions f and g, we sampled the time between the minimum time $\underline{\tau}_f + \underline{\tau}_g$ and the maximum time $\overline{\tau}_f + \overline{\tau}_g$ using decisecond precision yielding a vector T with length n of time samples. For every sample $T[i]$ we computed all values $h(T[i], T[i:n])$ and determined the value Δ that yielded the smallest result, where two results yielded the same result we used the smallest one.

Additionally we verified our C++ implementation of the algorithms by using a naive baseline algorithm that only computes the operations $\oplus$ and $\odot$ for single functions, not the full piecewise functions. By applying these operators pairwise to all sub-functions of two piecewise functions, we get a set of overlapping solutions, the lower bound of these solution represented our baseline. Again we used a set of samples T to compute the minimal value over all solutions in the set and compare it to the result our algorithm produced.

Empirical Verification

We empirically verified our algorithm by using a baseline algorithm to compute (tight) upper bounds of the minimal travel time for a set of test routes and compared these values to the minimal travel time our algorithm computed. For this purpose we used an extension of the BSP algorithm proposed by Baum et al. in [BDHS+14]. This algorithm uses a multi-graph where each edge represents a time-consumption trade-off $(\tau, f(\tau))$. Using the same graph model with hyperbolic consumption functions that we also use for the FPC algorithm, we sample the consumption function f_e of an edge e using a recursive algorithm. Given an interval $[\underline{\tau}, \overline{\tau})$ we split it in half if $f(\underline{\tau}) - f(\overline{\tau}) > y_{min}$, where y_{min} is our minimal sample resolution. For our tests we used a default sample resolution of 10 Wh on Luxembourg and 100 Wh on Switzerland. Splitting the interval recursively on consumption value is important because otherwise we would over-sample the parts of the consumption function with a shallow slope. Finding a constrained shortest path in this graph can be achieved using a modified Multi-Criteria Dijkstra with battery constraints, that terminates as soon as the first feasible fastest path to the target is found. However, this algorithm does not include charging stations yet and hence would not yield tight bounds for routes that make use of them. To incorporate this into the BSP algorithm, we check at every node if it is a charging station. If we arrive with the label $(\tau, f(\tau))$, and the node has a charging function $g(x, y)$ we insert all values $(\tau + \tau', g(\tau', f(\tau)))$ where $\tau' < \overline{\tau}_g$. Since we can only represent finite many trade-offs in the Pareto set, we need to sample $g(x, f(\tau))$ using the same approach described above for consumption functions. We use the same sample rate y_{min} for sampling the charging functions, that we use for sampling the consumption functions. Before inserting each sampled trade-off into the set of unsettled solutions, we also apply the time penalty for charging described in Section 3.2. Since charging functions typically span (almost) the whole capacity range of a battery this approach will generate many trade-off samples. To reduce the memory consumption of blowing up the set of unsettled labels after visiting a charging station, we compute the lower-envelop of all unsettled solutions after sampling the charging function. We will refer to this algorithm as MCC (Multi-Criteria with Charging) in this thesis. For our PTV Germany instance we increased the number of edges by 30% using a sample rate of 10 Wh,

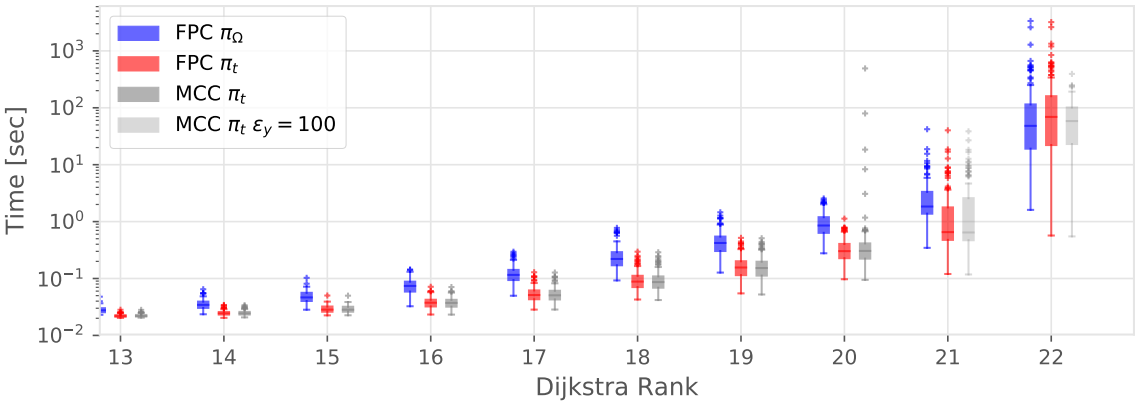

Figure 7.1: Dijkstra rank plot of algorithms presented here. We executed 100 queries per rank on PTV Germany using Tesla data and battery capacity of 85kWh. We only include data where no query exceeded two hours of processing time.

			Germany 85 kWh			Germany 64 kWh		
CS	Alg.	ϵ_y	#Labels	#Dom.	Avg. Time (Max) [s]	#Labels	#Dom.	Avg. Time (Max) [s]
Tesla	FPC π_t		5 698 119	13 806 981	56.21 (1331.58)	10 599 732	26 007 471	117.34 (2774.32)
	FPC π_Ω		4 231 923	10 421 942	**45.23 (1312.58)**	4 062 427	10 003 179	**43 (1782.48)**
	FPC π_t	100	1 830 543	3 759 531	8.22 (65.64)	2 866 834	5 861 370	12.43 (106.10)
	FPC π_Ω	100	971 061	1 981 139	5.28 (39.58)	815 778	1 652 646	4.46 (22.69)
	MCC π_t	100	196 946 559	19 176 673 363	202.98 (1452.40)	149 124 984	11 438 173 995	140 (727.31)
CM	FPC π_t		50 445 318	130 872 964	705.13 (4127.80)	*	*	*
	FPC π_Ω		*	*	*	*	*	*

Table 7.1: Comparison of average number of label pushes, dominance checks and query time between different algorithms on PTV Germany using 100 random queries. For entries marked with * we were not able to complete all measurements in time.

for OSM Switzerland we inserted and additional 10% more edges using the sample rate 100 Wh.

We evaluate 1000 random queries on OSM Luxembourg and 1000 random queries on OSM Switzerland using charging stations from ChargeMap, all travel times computed by FPC are below the values computed by MCC.

7.3 Performance

In this section we evaluate the performance of our algorithm, the proposed speedup techniques, and the proposed heuristics. To compare the performance of the different algorithms we used the *Dijkstra rank*. For this we selected 100 source nodes randomly and computed the distance to all other nodes in the graph. We then sorted all nodes by distance to the source, which corresponds to the order in which these nodes will be processed in Dijkstra's algorithm. Using this ordering we pick every 2^i-th node as target node, where we call i the rank of the node. For every target node we obtained this way, we verify if it is within battery range to a charging station or the start node. We do this by computing the minimum consumption needed to reach the proposed target node from all charging stations and the start node. Furthermore, we ensure that the target node is in the same strictly connected component as the start node, to avoid unreachable queries on OpenStreetMap data with many connected components.

See Figure 7.1 for an overview of query performance on PTV Germany using Tesla chargers. For each algorithm we only included ranks, where no query exceeded two hours of processing time, which was only the case for the MCC algorithm in rank 21 and 22. Instead, we show the performance of heuristic queries with $\epsilon_y = 100$ (light gray).

We can also see that on the PTV Germany + Tesla dataset the potential π_Ω only performs better than the simple potential π_t at the highest rank. We attribute this to the fact that

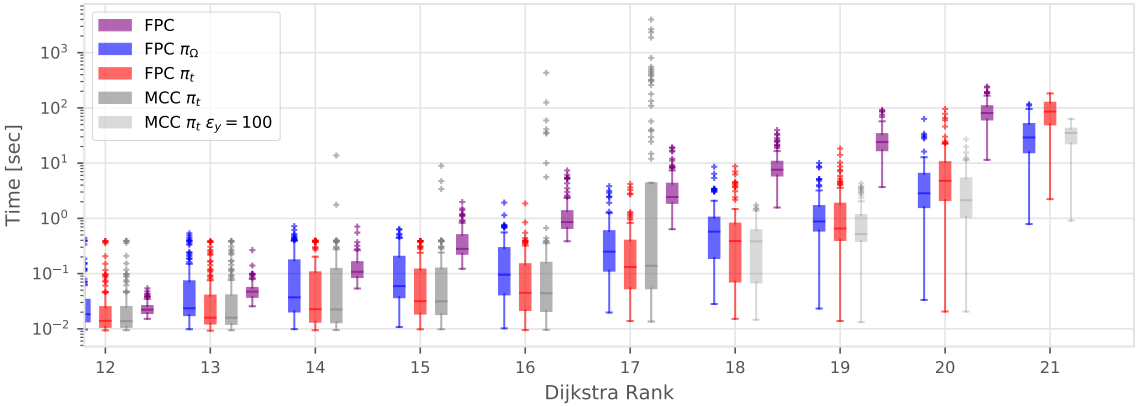

Figure 7.2: Dijkstra rank plot of algorithms presented here. We executed 100 queries per rank on OSM Switzerland using ChargeMap data using a battery capacity of 16kWh. We only include data where no query exceeded two hours of processing time.

			Switzerland 32 kWh			Switzerland 32 kWh		
CS	Alg.	ϵ_y	#Labels	#Dom.	Avg. Time (Max) [s]	#Labels	#Dom.	Avg. Time (Max) [s]
Tesla	FPC		38 692 691	92 257 370	279.30 (1463.40)	38 852 019	91 171 053	282.99 (1414.95)
	FPC π_t		2 247 819	5 307 885	8.97 (120.09)	7 966 532	19 059 594	43.54 (577.53)
	FPC π_Ω		1 848 287	4 389 391	**7.51 (92.39)**	6 079 161	14 500 810	**31.62 (421.37)**
	FPC π_t	100	832 206	1 692 487	2.55 (22.52)	1 991 756	4 067 413	5.64 (36.03)
	FPC π_Ω	100	632 482	1 300 421	2.31 (19.86)	1 611 910	3 330 062	5.12 (34.84)
	MCC π_t	100	9 096 723	195 798 056	6.34 (77.52)	16 880 069	283 744 457	11.29 (68.40)
CM	FPC π_t		2 346 115	5 620 224	9.58 (112.56)	5 327 858	12 792 839	22.59 (153.44)
	FPC π_Ω		1 745 867	4 221 445	8.20 (130.13)	2 106 863	5 064 340	8.49 (87.83)

Table 7.2: Comparison of average number of label pushes, dominance checks and query time between different algorithms on OSM Switzerland.

computing the π_Ω potential is significantly more expensive than computing the π_t potential. The actual benefit of the potential π_Ω only becomes relevant once we have to charge. For a battery capacity of 85 kW on Germany this is only the case for queries with ranks higher than 21. For all other queries we either never exceed the battery capacity, or it is sufficient to increase the range by driving slower. This effect can also bee seen in Table 7.1. For a battery capacity of 85kWh the average query time improvement of the potential π_Ω over the potential π_t is only moderate, however for the smaller capacity of 64 kWh the speedup is about 2.67.

An interesting observation is the number of outliers for each method. Even for low ranks we can see that the MCC algorithm has outliers that take almost take half an hour to compute. We see a similar behavior for our algorithm, even with a potential function. We attribute this to the fact that all algorithms presented here cannot guarantee a polynomial running time. To show the effect our speedup methods have on these hard queries, we include the maximal query time along the average in Table 7.1. While the speedup between different potentials seems about the same for hard queries, we see a dramatic difference when inspecting the maximal query time for heuristics queries. For the exact algorithm some queries can take about half an hour, even with speedup methods, this shrinks to about a minute for heuristic methods using $\epsilon_y = 100$. As we can see in Section 7.4 the maximal error for these heuristic methods is still very low, below 0.01% for the setting used in the table. This suggests that heuristics are especially helpful for interactive applications, that have tight limits on the time each computation can take before the user notices.

For Germany our ChargeMap instance was significantly harder than the Tesla instance. The average query time for the potential π_t increased by about factor 12 over the same configuration using Tesla charging stations. We attribute this to the higher number of charging stations, and the lower overall charging rate which creates a lot of solution with a

similar time-consumption trade-off. Unfortunately due to time constrains we were not able to complete all measurements for this dataset on Germany. Preliminary results show that there is a similar factor 12 overhead between the other configurations for the average query time. However the maximal query time for the potential π_Ω increased a lot, one query taking over 12 hours to complete. This suggests that a high number of charging stations, or a low maximal charging rate, is not beneficial for the potential π_Ω and can lead to an explosion of the search space.

For queries on our OSM Switzerland dataset in Figure 7.2 we see a similar behavior than for PTV Germany. As reference we additionally include the performance of unaccelerated FPC, which clearly shows that even for small ranks the speedup offered by the potential functions is significant. Note that we did not include the highest rank for the unaccelerated method, since some queries did not complete in two hours. The potential π_Ω is only faster than the potential π_t for the highest two ranks 20 and 21. For random queries with a battery capacity of 32kWh, we only see a slight speedup of the potential π_Ω over the potential π_t in Table 7.2. However, the speedup of the Ω-potential over the unconstrained fastest path potential for the smaller battery capacity 16 kWh is only 1.37 in this case. We attribute this to some features of the Switz road networks, that is constrained by tunnels and mountain passes. Unlike the German road network with a dense network of motorways that offer multiple equally fast alternatives, the Switz road network has only few viable paths. This leads to the unconstrained fastest path being very similar to the constrained fastest path, which works in favor of the π_t potential. Table 7.2 also includes measurements for random queries on Switzerland without using a potential function. We can see that for both capacities the FPC algorithm without a speedup method needs almost 5 minutes per query, even on a relatively small dataset like Switzerland. Our speedup methods yield a speedup of about 31 for the potential π_t and 37 for the potential π_Ω. For our heuristic queries using an ϵ_y value of 100 Wh, we can even achieve a speedup of two orders of magnitude for both potentials.

Please note the measurements we obtained here for the Switzerland dataset, were obtained on a somewhat harder dataset then necessary. Since we included height profiles for tunnels, some routes have an excessive consumption. Preliminary results show that on a Switzerland dataset with a more realistic modeling of tunnels, we can expect speedups of around factor three for each measurement.

Comparing Label Ordering Between Potentials

In Chapter 5 we claimed that the π_Ω potential would lead to charging stations being settled earlier in the search. To verify this claim we ran 4000 random queries on our PTV Luxembourg dataset and recorded the iteration a charging station label was settled.

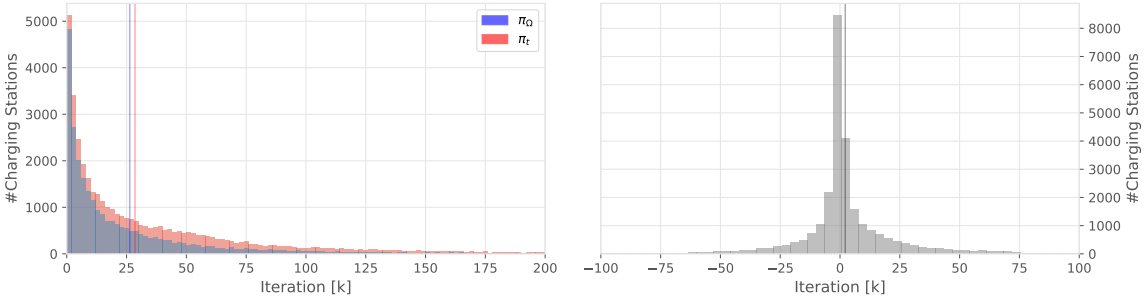

Figure 7.3: Left: Histogram of the iteration in which a label of a charging station is settled. The red vertical line shows the average for π_t, the blue vertical line shows the average for π_Ω. Right: Histogram of the iteration difference for each charging station between the potential π_t and π_Ω, a negative value mean π_t settled the label before π_Ω.

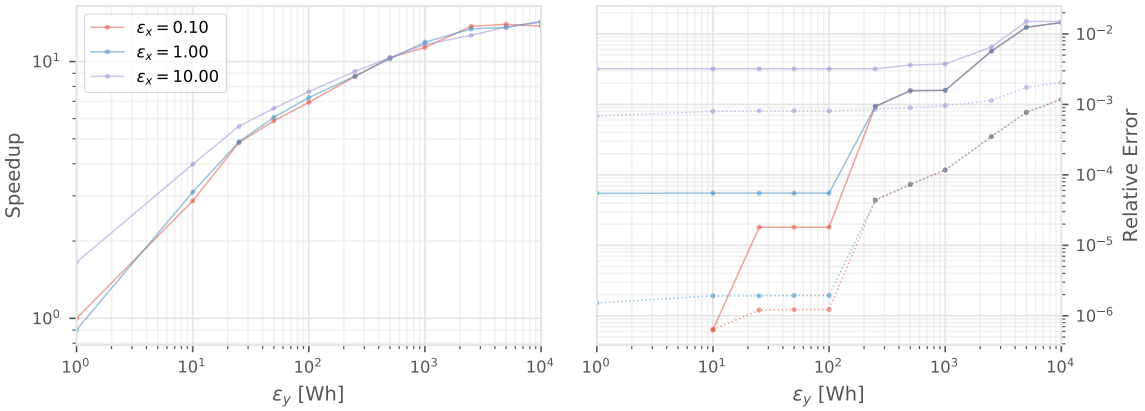

Figure 7.4: FPC algorithm using different epsilon values for x and y. On the left size we denote the speedup to the exact FPC, on the right side the solid lines indicate maximum error, the dotted lines average error. All queries were run on PTV Germany with Tesla charging stations using 85kW battery. Every data point was computed using random 100 queries.

The results are shown as two histograms in Figure 7.3. From the left histogram it is evident that π_t settles a lot more charging station labels than π_Ω, but we can not see if π_Ω generally settles charging stations earlier. Thus, we compared the iteration in which the same charging station label was settled by the two potentials. Figure 7.3 shows the histogram of differences between those iterations, a positive difference means the charging station was settled by π_Ω before it was settled by π_t. Clearly both cases occur, while the mean of π_Ω is about 26 367 iterations to settle a charging station, and about 28 525 iterations for π_t. To establish if this difference is actually statistically significant, we used Student's t-test for related measurements with the null-hypothesis that both π_Ω and π_t have the same mean value. With a p-value of below 10^{-67} ($t = 17.526$) this hypothesis was rejected, it is thus likely that π_Ω indeed settles charging stations before π_t.

7.4 Heuristics

In this section we will evaluate the performance of the heuristics we proposed in Chapter 6. To evaluate the solution quality, we used the relative error of the computed travel time. Since the exact travel τ is smaller or equal to the heuristic travel time $\hat{\tau}$, we define the relative error as $\tau/\hat{\tau} - 1$. We will usually report both the maximum relative error and the average relative error over all test queries, together with the speedup of the average query time. For some queries the heuristic might not find a solution at all, so we additionally report the fraction of queries that have not found a solution.

Epsilon-Dominance

The first heuristic we want to evaluate is the relaxed dominance criterion. For this purpose we ran a parameter sweep of different pairs of ϵ values and compared them to the exact (or in the case of MCC "exactest") values. One thing is immediately clear when looking at the results in Figure 7.4 and Figure 7.5, of both algorithms do not profit much from an increased ϵ_x value. The speedup for all values of ϵ_x is about the same for all ϵ_y values, with the notable exception of small ϵ_y values for the FPC algorithm. We attribute this to the fact that a label in the FPC algorithm consists of piecewise functions, for which each sub-function is either fully dominated and removed, or undominated. If a sub-function has a slightly smaller $\underline{\tau}$ value than another sub-function, it can not be dominated even if the time-consumption trade-off is much worse for most pats of the function. This effect is less prevalent with the MCC algorithm where each label is only a single point and does not suffer from all-or-nothing effects like these. While a higher ϵ_x only marginally increases

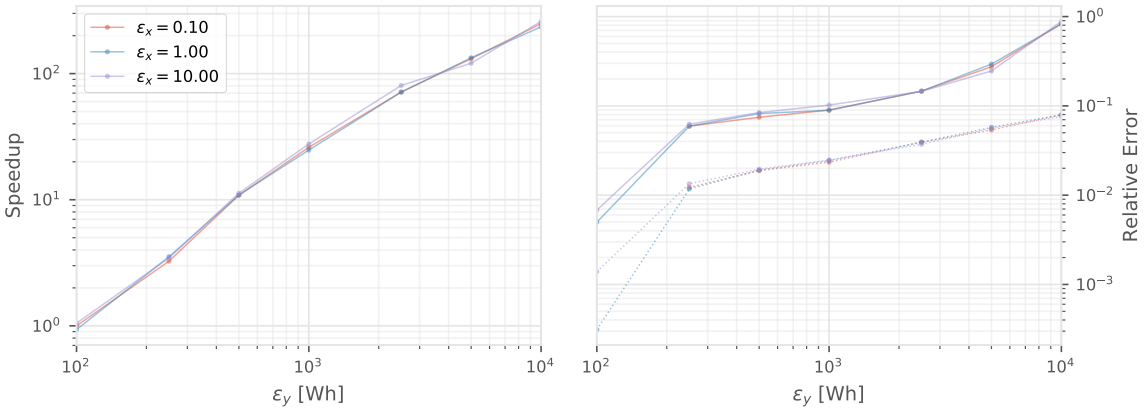

Figure 7.5: MCC algorithm using different epsilon values for x and y. On the left size we denote the speedup to MCC with $\epsilon_y = 100$, on the right side the solid lines indicate maximum error, the dotted lines average error. All queries were run on PTV Germany with Tesla charging stations using 85kW battery. Every data point was computed using random 100 queries.

the speedup, in the case of the FPC algorithm it can dramatically increase the relative error. An value ϵ_x of 10 has maximum error of 2%, which is much worse then 0.1 and 1 even for higher ϵ_y values. Hence, we conclude the minimal "exact" setting $\epsilon_x = 0.1$ (one deci-second) is optimal.

For the FPC algorithm we see some diminishing returns for values of ϵ_y above 1000. The speedup of about 10 does not increase much after that, while the relative error quickly starts to rise. The value $\epsilon_y = 100$ offers a good trade-off between a speedup of about factor 7 and an error below 10^{-5}, while the maximum error for $\epsilon_y = 1000$ is about $10^{-}3$.

For the MCC algorithm we observe much higher speedups for bigger ϵ_y values than for the FPC algorithm. This can be attributed to the fact that the sampling method implied by MCC introduces more labels than the FPC algorithm, since each label only represents a single trade-off point. By increasing the ϵ_y we create bands in the Pareto set of size ϵ_y, reducing the number of labels per node. For example, increasing the ϵ_y from 1 Wh to 10 Wh already limits the maximal label size to a tenth. This approach does not profit from an increased ϵ_x value. If we consider that the slope of all trade-off and charging functions is limited, consecutive sampled values of these functions will also have a limited slope between points. Since the values on the Pareto front of a label in the MCC algorithm typically represent consecutive values of such a function, it is easy to see that a value ϵ_y already implies a minimum distance Δx that every other label with a smaller consumption value y needs to have. Given the minimal slope $\alpha_{min} < 0$, we can calculate the implied ϵ_x as $-\epsilon_y/\alpha_{min}$. For a minimal slope of -33 (if we include super chargers in charging stations) this amounts to an implied ϵ_x of about 7.57 for $\epsilon_y = 250$, which is just below the highest ϵ_x value we tried.

Model Simplifications

In Chapter 6 we discussed several methods to simplify the realistic model we proposed in this thesis to speedup the query calculation. In this section we will present the speedups we can gain using these simplifications and at what cost in quality they come. All results can be found in detail in Table 7.3, which is the result of 4000 random queries on the PTV Luxembourg dataset. Since the dataset is quite small we scaled the battery capacity to 4kWh to make sure the queries need to use charging stations.

The speedups for each heuristic highly depend on the speedup technique they are used with. For example, the simplification of trade-off functions as a single linear functions, generally performs poorly, expect when combined with the potential π_Ω. In this case the heuristic

leads to a speedup of 1.64, despite a significantly increased number of sub-functions. We attribute the speedup to the lower constant factors for operations like combining dominance checks, and the search space being smaller. We believe the reason this heuristic works well with the π_Ω potential is because the search space is already small enough to not generate many long labels. This is apparent if look at the number of sub-function for the `linear` heuristic using no potential functions. In this case the heuristic doubles the number of sub-functions, which is much more then the moderate increase we see for the Ω potential. These speedups come at a cost of an average error of around 1.6% and a maximum error of 9.4%.

On the other hand the heuristic that only considers trade-offs on roads with a maximum speed above 100 km/h (denoted `only fast` in the table) yields good speedups, for the π_t potential and using no potential functions, but is actually slightly slower than using the π_Ω potential without a heuristic. Since the Luxembourg dataset only has few motorways, only 622 edges qualify for trade-offs in this case (below 1% of the edges), hence it is not always possible to use paths with trade-offs which limits the range of queries. This can also be seen by the fact that this heuristic generates much less sub-functions than all other settings, even for the π_Ω heuristic it cuts the length of labels in half. However the limited use of trade-offs, means more energy-efficient slower roads are explored rather then driving slower. This is leads to an increase in the search space, as is apparent by the number of labels, that is much higher for this heuristic then for all other settings. In about 5% of the cases the decreased range does not only lead to a higher travel time of up to 50%, but also will result in no path being found. This result underlines the importance of allowing time-consumption trade-offs for high quality results. Since this heuristic heavily depends on the structure of the road network, it would be interesting to evaluate it on a bigger dataset with more motorways.

The heuristic that limits the maximal slope (denoted `min rate`) suffers from similar problems due the limited reachability. However, unlike the heuristic based on speed it consistently leads to speedups between 8% and 20% and does not increase the runtime for the π_Ω potential. We attribute this to the fact that the total number of sub-functions for this heuristic is significantly smaller than the baseline, clipping off the bad trade-offs as intended. However, since this heuristic conserves good trade-offs it does not suffer from a significant increase in the search space, like the `only fast` heuristic.

Potential	Heuristic	#Labels	#Fn.	#Dom.	Stops	Speedup	Error (Max.)	N. F.
none	none	178 247	841 365	408 733	0.72	-	-	-
	linear	181 198	1 997 400	410 950	0.74	0.90	0.009 (0.076)	0.000
	only fast	313 846	529 001	702 559	0.78	1.24	0.021 (0.457)	0.050
	min rate	195 381	679 160	450 852	0.74	1.22	0.005 (0.455)	0.000
	no slow cs.	172 439	810 954	394 174	0.72	1.05	0.000 (0.000)	0.000
π_t	none	55 664	263 861	128 934	0.72	-	-	-
	linear	55 203	664 142	125 761	0.74	0.79	0.009 (0.076)	0.000
	only fast	93 270	147 327	210 686	0.78	1.10	0.021 (0.457)	0.050
	min rate	65 471	225 862	153 809	0.74	1.08	0.005 (0.455)	0.000
	no slow cs.	54 876	260 606	126 889	0.72	1.00	0.000 (0.000)	0.000
π_Ω	none	31 859	132 512	73 662	0.72	-	-	-
	linear	16 623	169 416	36 729	0.74	1.64	0.016 (0.094)	0.000
	only fast	55 048	76 697	125 026	0.78	0.97	0.021 (0.457)	0.050
	min rate	41 412	121 549	98 247	0.74	1.00	0.005 (0.455)	0.000
	no slow cs.	32 521	137 372	74 469	0.67	0.97	0.000 (0.000)	0.000

Table 7.3: Comparison of average number of label pushes, dominance checks and query time between different algorithms on PTV Luxembourg using a battery capacity of 4kWh.

	Avg. #Speed Changes (Max.)			
	Switzerland 32kWh	Switzerland 16kWh	Germany 85kWh	Germany 64kWh
FPC	71 (176)	84 (207)	107 (261)	117 (275)
Unconstrained	72 (144)	72 (144)	91 (193)	91 (193)

Table 7.4: Comparison of the average number of speed changes for the fastest path and the constrained fastest path, based on 100 random queries each.

The heuristic of removing all slow charging stations (denoted `no slow`) only yields a small speedup of 5%, which is only the case when no potential function is used. However this heuristic does not seem to impact the quality at all, most likely the FPC algorithm select alternative charging stations or compensates by driving slower.

All presented model simplifications only yielded small speedups, much below the speedups we see using ϵ-dominance, while having a higher error. Thus, we feel confident in our choice of using a realistic model over simpler models.

7.5 Route Quality

In this section we want to investigate the route quality of routes computed using our algorithm. There are multiple factors that determine if a route is "good", some of them highly subjective and based on personal preference. That is why we want to focus here on two criteria that can limit the practical usefulness of the routes. The first criteria is the number of speed changes per query, which we define as the times a driver either needs to slow down or accelerate. These can be caused by legal limits, for example when slowing down every time we pass through a small town on rural roads, but in our model we also allow the algorithm to lower the driving speed to save energy. Hence, we need to evaluate the impact this has on the number of speed changes. Another criteria we want to investigate is the number of charging stations each query uses. Frequent use of charging stations can quickly become impractical, since every charging stop adds a certain time overhead and disrupts the drive. Since we allow to charge the battery only partially, the number of charging stops is not necessarily implied by the minimal consumption to reach the target. It might simply be more time effective to charge the battery three times only for a short amount of time and otherwise save energy by driving slower. One method to limit the number of charging stops is adding a charging penalty, we evaluate different values and their impact on the charging patterns.

Speed changes

Looking at the profile plots of our worst-case queries in Figure 7.10 and Figure 7.9, it is apparent that we frequently need to change the driving speed. This might limit the usefulness of the route for actual driving, since a driver might not actually be able to replicate the optimal driving speed profile. To evaluate the number of speed changes for each query, we count the number of times where the absolute difference between the speed of two segments is bigger or equal to 5 km/h. Especially for our OSM dataset small segments or segments with zero length throw off the measurements, so we only measure speed changes over segments of at least 5 meters of length. Some of the speed changes will not be caused by our algorithm, but are just the result of different legal maximum speeds on the route. To account for this we compare the measurements to the number of speed changes on unconstrained shortest paths on the same datasets. The results for PTV Germany and OSM Switzerland can be found in Table 7.4. For all cases, except Switzerland with 32kWh battery, the constrained shortest path has more speed changes on average. For smaller battery capacities the number of speed changes increases, which fits

Turn Cost	#Labels	#Fn.	#Dom.	#Stops	#Speed Changes (Max.)	Speedup
zero	216 117	1 016 032	427 509	0.68	39 (133)	-
static	211 653	971 981	420 783	0.67	62 (197)	1.07
avg. cons.	233 321	1 222 960	449 321	1.07	40 (138)	0.69
avg. cons. + st.	212 043	1 098 925	383 194	1.07	63 (197)	0.76

Table 7.5: Comparison different heuristics for computing turn costs. Measurements are based on 4000 random queries on PTV Luxembourg.

CP	#Labels	#Dom.	Time [ms]	Stops	Charged [Wh]
0	2 032 338	4 884 708	8 073	1.99	17 158
60	2 106 863	5 064 340	8 481	1.82	16 941
120	2 130 405	5 114 475	8 515	1.74	16 861
180	2 132 330	5 108 817	8 453	1.67	16 712
300	2 153 080	5 151 353	8 563	1.60	16 447

Table 7.6: Various metric for different charging penalties measured using 100 random queries on OSM Switzerland with ChargeMap charging stations.

our intuition that speed changes are caused by using trade-offs that induce lower driving speeds. In the worst case we insert about 80 additional speed changes in a route. This suggests that using a post-processing method that tries to reduce the number of speed changes, while ensuring a similar consumption, would be beneficial for practical use of our algorithm.

Turn Costs

One approach we proposed to reduce the frequent number of speed changes was using a model that allows for specifying turn costs. By computing the additional energy we loose by slowing down and subsequently accelerating again, a route with fewer speed changes would be more energy efficient. However since we were not able to incorporate an exact model of consumption turn costs into our trade-off model, we will evaluate three heuristics here. The results can be found in Table 7.5. Every measurement is based on 4000 random queries on PTV Luxembourg. We can see that none of the heuristics we proposed had the intended effect of reducing the number of speed changes. At best, we did not add a significant amount of speed changes, which is the case for adding the average consumption caused by speeding up or slowing down. Adding a static turn penalty yields a speedup for computing queries, does however increase the number of speed changes significantly. While a static time penalty makes it more likely for the route to stay on the fastest path, it might lead to more trade-offs being used on that path to get a similar consumption than using a shorter more energy efficient route.

Charging Penalty

Frequent use of charging stations should in general be discouraged, since it is disruptive to the driver. For this purpose we introduced charging penalties in the form of a delay between arrival at a charging station and the time were the charging actually starts. To evaluate our quite conservative choice of using one minute as charging penalty, we compare the basic query statistics in Table 7.6. We can see that with no charging penalty we use on average about two charging stations per query, while using only 1.82 with a charging penalty of 60 seconds, and 1.60 for waiting 5 minutes before starting the charging. For all charging penalties at most 17% used more then 3 charging stations, one query used 8 charging stations.

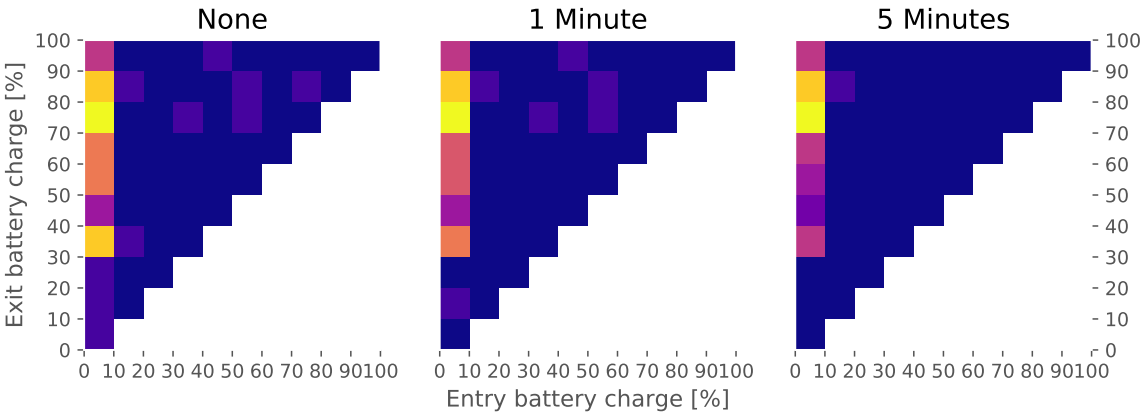

Figure 7.6: Impact of the charging penalty on when we start and stop charging based on the SoC of the battery.

In this context it is interesting to evaluate the charging pattern that is employed when using charging stations. Figure 7.6 shows multiple heat maps that denote at which battery charge we stop charging, depending on the battery charge when we started charging. Overall we can see that for a higher charging penalty, we will only start to charge the battery when the charge is low. We attribute this to the fact, that in our charging model the charging rate decreases rapidly after we reach 80% of charge. Adding charging penalties further decreases this charging rate, making it less attractive than using other trade-offs like driving slower. We can also see that in some cases it also pays off to only charge the battery slightly, probably to increase the reachability just enough to reach the target. This justifies our choice of using realistic charging functions, since simplified charging models that only charge to full capacity would not allow for this.

This distribution also suggest possible charging heuristics, for example to only allow charging if the state of charge of the battery is below a certain threshold. Another heuristic could be to only charge to 40% and 80% of the battery capacity, since most queries seem to do just that. Both heuristics could reduce the number of superfluous charging station labels.

7.6 Survey

In this section we want to present a few selected example queries that we computed using our algorithm. For this purpose we developed an interactive web frontend using HTML, Javascript, React and Mapbox GL JS with OpenStreetMap map data. The web frontend sends requests to a HTTP server and that returns the full route with height profile, maximum and actual speed for each segment, and geometry. Charging stations are denoted by colored points on the map. Slow charging stations below 22 kW charging rate are colored yellow, stations with 22 kW are color orange, and stations above 44kW are colored red.

Trade-offs vs. Charging Stations

To get a feeling for the preference of our algorithm with regard to driving slower, or charging more often we inspected a few example queries on PTV Luxembourg that can be found in Figure 7.7. Since Luxembourg is a very small dataset, we limited the battery capacity to 4 kWh to force the algorithm to use charging stations or trade-offs. In general the algorithm uses both methods to reduce consumption. The image on the left of Figure 7.7 shows a query that makes almost no use of reducing the driving speed, due to having access to a lot of charging stations along the route. The route branches off twice to reach nearby

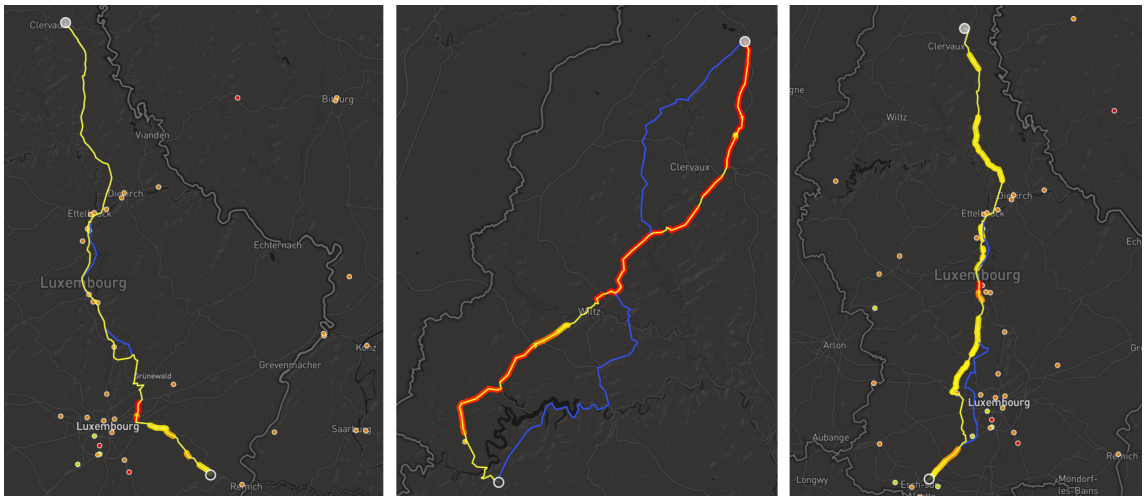

Figure 7.7: Examples of three different queries on PTV Luxembourg using a 4000 kW battery and ChargeMap charging stations. The yellow line represents the route computed using FPC, the blue line is the unconstrained fastest path. Segments with speed reduction are marked by color, where yellow is 5 km/h and red 30 km/h speed reduction.

charging station, but otherwise coincides with the fastest path. The image in the middle of Figure7.7 on the other hand, shows a query in the northern part of Luxembourg that makes excessive use of reducing the speed to achieve a lower consumption. This reliance on using time-consumption trade-offs is cause by a very low density of charging stations in this part of Luxembourg. There is only one charging station in reach at the very start of the query. By comparing the route to the unconstrained fastest path in blue, it is easy to see that in addition to driving slower the route is also optimizing for distance to save additional consumption. The image on the right of Figure7.7 is an example of a query where the speed is only reduced moderately, in combination with using charging stations along the way. In general the route overlaps with the unconstrained fastest path, except for detours to reach charging stations. We conclude that both options to reduce the consumption are used by the algorithm, which confirms our choice in allowing this degree of freedom in the model.

Search Space

In this section we investigate the search space of a few example queries and the impact the potentials π_Ω and π_t have on it. Figure 7.8 shows the search space for the example query from Figure 7.7 (right) for each potential function. Every node in the search space is visualized using its geographical location, the color indicates the number of settled labels in the Pareto set. Green nodes have one or two labels, while red nodes have above 20 labels in their label set. We can see that for both potentials we initially only investigate labels on the fastest path, until we exceed the battery capacity. After 40 000 iterations the potential π_t only increased the search space around the fastest paths, adding a lot of labels that slightly improve the consumption by taking more energy-efficient detours, but do not increase the range much. At the same time the potential π_Ω has already settled at least one charging station, it increased the search space further along the fastest paths to the target. After 60 000 iterations the potential π_t settled at least one charging station label as well, which is evident by the search space expanding furtherer along the fastest path. For the same number of iterations, the potential π_Ω has extended its search space down towards Luxembourg City that has a cluster of charging stations. After 95 000 iterations the potential π_t is making its way to Luxembourg City as well, while the π_Ω potential has settled a second charging station and has almost reached the target in Esch-sur-Alzette.

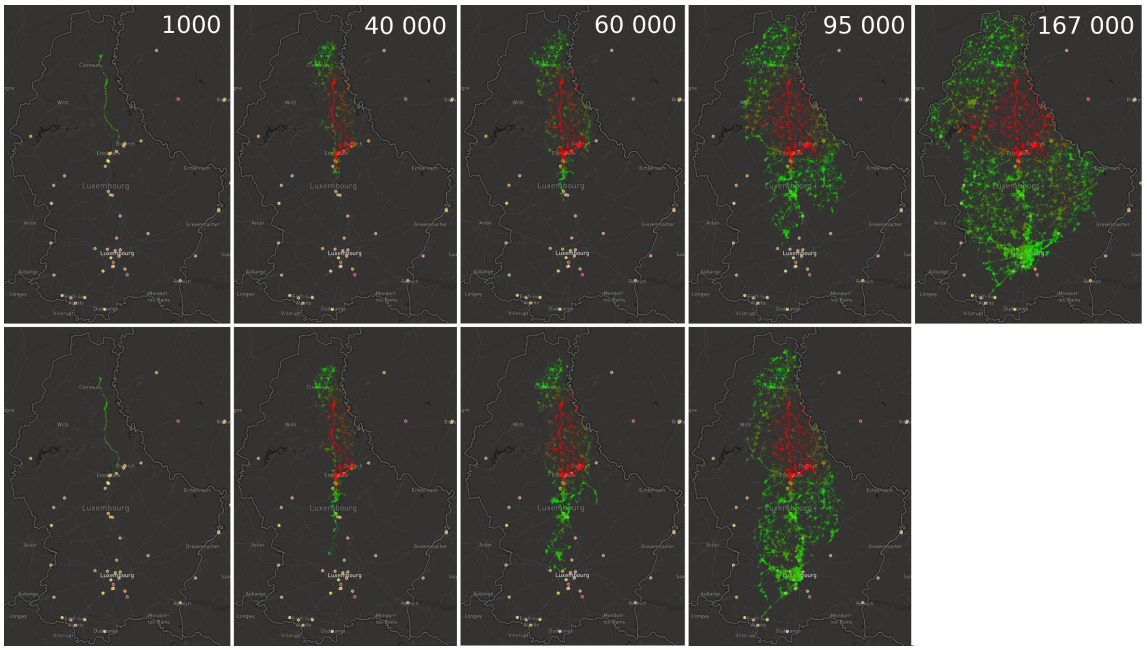

Figure 7.8: Examples of the search space for a query using a 4000 kW battery on PTV Luxembourg. Top row shows the search space for the potential π_t, bottom row shows the search space for the potential π_Ω. Each column is marked with the number of iterations that have been executed so far. Since the π_Ω potential finished after 95 000 iterations, we don't show a search space for the last column.

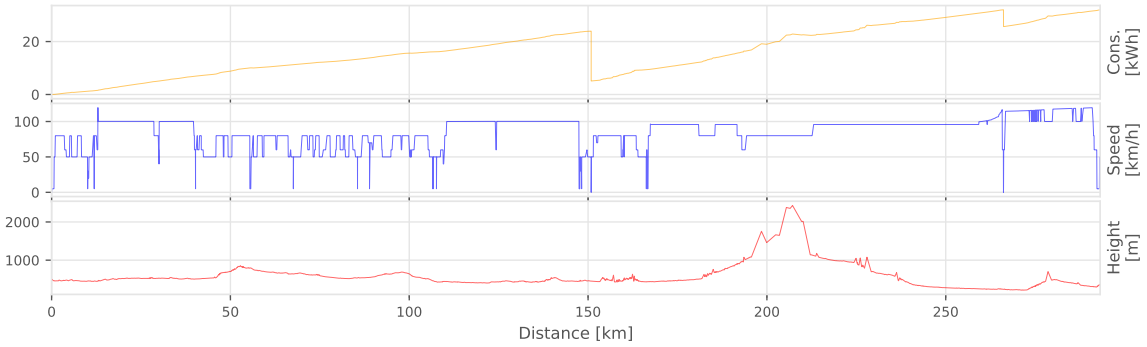

Figure 7.9: Consumption, speed and height profiles of the worst case query on OSM Switzerland, that goes from Bern to Lugano. Note the peek of 2000 meters on the height profile, which is caused by the Gotthard tunnel, which was assigned the height profile of the mountain it passes through.

The query using the potential π_Ω terminates after 97 000 iterations, while the query using potential π_t needs another 70 000 iterations to reach the target and terminate.

Worst-Case Queries

In this section we want to inspect two worst-case queries on the datasets that we used in this evaluation. Interestingly the worst-case queries for both potential functions π_t and π_Ω are the same, which suggests something intrinsically hard about these queries. The first worst-case query we want to investigate was run on the OSM Switzerland dataset from Bern to Lugano and took about two minutes to complete. When looking at the consumption, speed and height profile of this query in Figure 7.9, the height profile features a high peak of 2000 meters. This is in fact an artifact caused by using SRTM height data interpolation, since the query in question passes through several tunnels, which each got the height profile of the mountain range assigned to them. The peak in question is caused

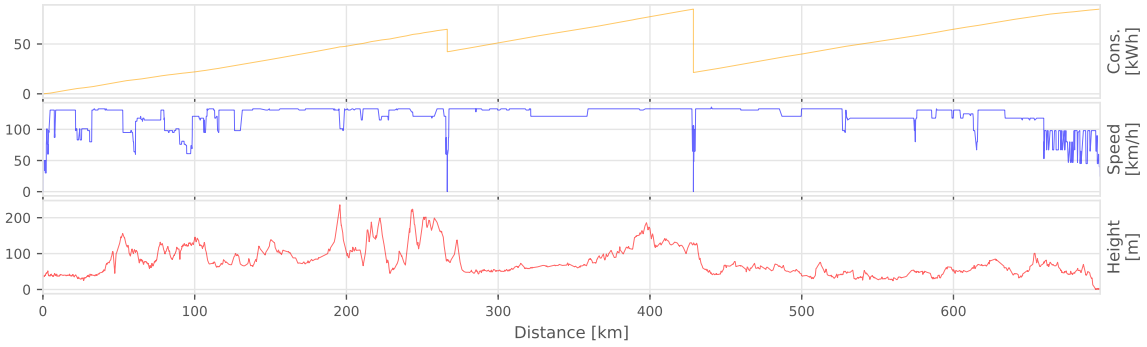

Figure 7.10: Consumption, speed and height profiles of the worst case query on Germany that goes from Mönchengladbach to Schwedt on the border to Poland. This route passes a number of very large metropolitan areas with lots of charging stations, that introduce new labels.

by passing through the Gotthard tunnel, which explains why this query was particularly hard, since there is no easy way to circumvent this tunnel when going to Lugano. Since the incline caused an abnormally high consumption for this path, most solutions that pass through the tunnel have to use their time-consumption trade-off, thereby increasing their minimal travel time. This causes the search to first explore more nodes until no sensible alternative is found, before finally passing through the mountain. This can also be seen by the 12 955 charging station labels that this query created, which is actually the highest number of charging station labels for all queries. To address this issue special care needs to be taken to post-process the elevation data, for example removing it from tunnels and interpolating the elevation between the entry and exit of the tunnel. Preliminary results show that the same query on a Switzerland dataset with contracted tunnels is three times faster.

The second worst-case query we want to investigate, is the query from Mönchengladbach to Schwedt on our PTV Germany dataset. This route passes by multiple major metropolitan areas such as Dortmund, Köln, Hanover, and Berlin. In addition to that it uses a route through middle Germany that is a charging station “desert”, especially on the Tesla dataset. That leads the search so settle a lot of charging station labels near the beginning of the route to bridge this “desert”. Since most charging stations offer similar trade-offs there is no obviously superior path until the super charger is settled. That means while the search space is huge, the final route mostly stays on the fastest path, using two super-chargers, one near Hanover one one near Magdeburg. For this query reducing the number of charging stations would help, for example by per-computing at which state of charge it makes sense to start charging at a given charging station at all.

8. Conclusion

In this thesis we presented our own novel algorithm to compute travel time minimal paths using a model that considers charging stations and time-consumption trade-offs. We presented several speedup techniques for computing travel time minimal paths, both exact and heuristic, that allowed for queries in a reasonable time on realistic datasets such as Germany. We investigated possible model simplifications that allow for even bigger speedups, while sacrificing only a small amount of accuracy. We think our algorithm is an excellent starting point for further developing more optimized models and algorithm, as it allows for the first time to measure simplifications against a comprehensive exact baseline on realistic datasets.

8.1 Future Work

We see great potential for further improving the algorithm proposed here, using the already established technique of Contraction Hierarchies. We believe that combing the approach presented by [BDWZ17], the approach of [BDG$^+$15], our algorithm and our potential function π_Ω could yield an algorithm that could scale even on continental sized networks. For this, charging stations need to be kept in the core of the contracted graph, as in [BDG$^+$15], and shortcuts are only allowed to represent increasing or decreasing paths as in [BDWZ17]. By carefully tuning the model, we believe that query times similar to non-consumption based models can be achieved. One particular adaption would be to only allow trade-offs on roads with high maximum speed such as motorways, and assume constant edge costs for all other edges. While we only saw a limited success of this method on graphs with no preprocessing, this could simplify the implementation of the Contraction Hierarchies based algorithm dramatically since only constant edge weights need to be considered, albeit with battery constraints.

One big problem, especially for short-term queries, is that in order to compute the π_Ω potential we need to run three one-to-all backward searches from the target node. This is incredibly wasteful for large datasets, as only a small fraction of nodes will actually be considered as part of the search space. We already introduced a method to mitigate this problem, using lazy generation of the potentials, though the potential generation still has a significant overhead. Another option would be to adapt the ALT algorithm [GH05]. For the computation of the π_Ω potential it is sufficient to have lower bounds for $dist_t(v,t)$, $dist_c(v,t)$ and $dist_\Omega(v,t)$. By pre-computing all these metrics for a set of landmarks the ALT approach could be used to extract a lower bound for each metric. This would eliminate

the need for running a backward search from the target t but make the potential more optimistic.

The distribution of charging stations is very clustered around bigger cities. However, when passing these cities only a few of these charging stations are actually used for long distance routes. This suggests a hierarchy of charging stations. Few fast charging stations dominate all results from charging stations with a lower charging rate in a certain radius. Using this idea to pre-compute at which SoC it would make sense to start charging at a given charging station could improve the query time especially for outlier queries that settle lots of charging stations.

Due to traffic the departure time can have a large effect on the arrival time, both for fossil fuel based vehicles and electric vehicles. Hence, it would make sense to include this time-dependence in the model. If we restrict ourselves to an approach without preprocessing, only using the π_Ω potential, this is straightforward to implement. Only the bounds $[\underline{\tau}, \overline{\tau})$ of the trade-off functions would depend on the arrival time, which can easily be adjusted for all outgoing edges when relaxing a node.

In this thesis we introduced an exact model for computing turn cost, both for consumption and time. However, these turn cost depend on the arrival, departure and intermediate speed at an intersection. Since we work with trade-off functions these values are generally only determined when computing the optimal trade-off between functions. Integrating the turn cost into the objective function of the minimization problem to calculate optimal trade-offs, makes the problem hard to solve analytically. To circumvent these difficulties we used heuristic turn cost, that yielded unsatisfactory results. However, it might be feasible to adjust the speeds on the fastest constrained shortest path after the fact, to minimize the acceleration/deceleration and get an exacter value for the consumption and time with fewer speed changes.

To model the consumption we relied on empirically computed parameters for the time-consumption trade-off functions. These parameters might depend on the specific make of electric vehicle that was used to capture the empirical data. In our case the data was based on the compact car Peugeot Ion, which could have a very different consumption profile than a sedan like the Tesla Model S. For practical use in a navigation application it would make sense to allow specifying model parameters that depend on the electric vehicle at query time.

Bibliography

[BDG+15] Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 44. ACM, 2015.

[BDHS+14] Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner. Speed-consumption tradeoff for electric vehicle route planning. In *OASIcs-OpenAccess Series in Informatics*, volume 42. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[BDSV09] G Veit Batz, Daniel Delling, Peter Sanders, and Christian Vetter. Time-dependent contraction hierarchies. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 97–105. Society for Industrial and Applied Mathematics, 2009.

[BDWZ17] Moritz Baum, Julian Dibbelt, Dorothea Wagner, and Tobias Zündorf. Modeling and engineering constrained shortest path algorithms for battery electric vehicles. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 87. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[Cor01] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2001.

[Dij59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[EFS11] Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal route planning for electric vehicles in large networks. In *AAAI*, pages 1108–1113, 2011.

[FHS14] Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014.

[GH05] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[GP14] Michael T Goodrich and Paweł Pszona. Two-phase bicriterion search for finding fast and efficient electric vehicle routes. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 193–202. ACM, 2014.

[GSSV12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[HF14] Frederik Hartmann and Stefan Funke. Energy-efficient routing: Taking speed into account. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 86–97. Springer, 2014.

[HNR68] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[HRZL09] Stefan Hausberger, Martin Rexeis, Michael Zallinger, and Raphael Luz. Emission factors from the model phem for the hbefa version 3. *Report Nr. I-20/2009 Haus-Em*, 33(08):679, 2009.

[MSS15] Sören Merting, Christian Schwan, and Martin Strehler. Routing of electric vehicles: Constrained shortest path problems with resource recovering nodes. In *OASIcs-OpenAccess Series in Informatics*, volume 48. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[OSR17] Team OSRM. Turn function definition osrm 5.13, 2017. https://github.com/Project-OSRM/osrm-backend/blob/02a2d25a3febf04c6c8e57641a4e927abe958fd6/profiles/car.lua#L423-L433.

[PwC16] PwC. Bis 2030 ist jeder dritte neuwagen in der eu ein elektroauto, 2016. https://www.pwc.de/de/pressemitteilungen/2016/bis-2030-ist-jeder-dritte-neuwagen-in-der-eu-ein-elektroauto.html.

[Sin09] S. Singh. *An Introduction to Engineering Physics*. Discovery Publishing House, 2009.

[SLAH11] Martin Sachenbacher, Martin Leucker, Andreas Artmeier, and Julian Haselmayr. Efficient energy-optimal routing for electric vehicles. In *AAAI*, pages 1402–1407, 2011.

[Sto12] Sabine Storandt. Quick and energy-efficient routes: computing constrained shortest paths for electric vehicles. In *Proceedings of the 5th ACM SIGSPATIAL international workshop on computational transportation science*, pages 20–25. ACM, 2012.

[WAB16] WABCO. Bremsverzögerung und hochrechnung, parameter einer druckluftbreme, 2016. ftp://internic.at/Druckluftbremse/8150200573-23.pdf.