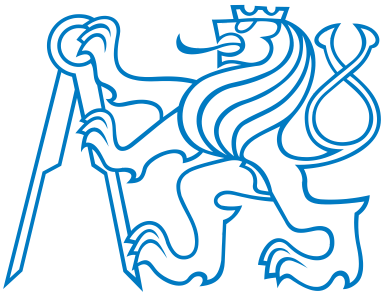
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor Thesis

Integrated Route and Charging Planning for Electric Vehicles

Tomáš Fišer

Supervisor: doc. Ing. Michal Jakob, Ph.D.

Study Programme: Open Informatics

Specialisation: Computer and Information Science

January 10, 2017

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Tomáš F i š e r

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Integrated Route and Charging Planning for Electric Vehicles

Guidelines:

1. Familiarize yourself with the existing approaches to formalize and solve the problem of route planning for electric vehicles (EV) with integrated charging station selection.
2. Select and formalize an appropriate variant of the EV routing problem that considers dynamic charging prices for charging station selection.
3. Propose and implement an algorithm for the selected variant of the EV routing problem.
4. Evaluate the performance of the implemented EV routing algorithm on real-world data.
5. Integrate the algorithm into an EV routing demonstration application.

Bibliography/Sources:

[1] Emmanouil S. Rigas, Sarvapali D. Ramchurn, and Nick Bassiliades. "Managing electric vehicles in the smart grid using artificial intelligence: a survey." In IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 4, 2015.
[2] Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. "Energy-optimal routes for electric vehicles." In Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 54-63. ACM, 2013.
[3] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. "Route planning in transportation networks." arXiv preprint arXiv:1504.05140 (2015).

Bachelor Project Supervisor: doc. Ing. Michal Jakob, Ph.D.

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 11, 2016

Aknowledgements

At first, I would like to thank my supervisor doc. Ing. Michal Jakob, Ph.D. for the guidance and for providing the opportunity to work on this challenging topic. Then, I wish to thank Ing. Jan Nykl for many consultations associated with implementation and writing, Ing. Pavol Žilecký for the knowledge transfer about Open Street Map data and Tomáš Breník for comments to a language usage. Also, I am grateful to the Open Street Map contributors for providing geographical data for free.

Finally, the access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

Declaration

I declare that the presented work was developed independently and I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague on January 9, 2017 .....

Abstract

This thesis is aimed at journey planning for electric vehicles (EVs), where it is necessary to make stops at charging stations. We minimize the travel costs of the journey in a model o transport network where the price per unit of energy may vary due to the 'Dynamic pricing' strategy. To avoid inappropriate detours, we consider that travel time is also included in the travel costs. Furthermore, the significance of the time spent on the journey is determined by the EV driver himself. We have proposed a bicriteria algorithm that computes a set of optimal journeys. The set contains a journey with the minimum travel costs, a journey with the minimum travel time and alternative journeys that are the trade-off between both criteria. The algorithm is based on Bicriteria Shortest Path algorithm. We extended the Bicriteria Shortest Path algorithm to satisfy the EV battery constraints and to allow recharging at charging stations. Moreover, we proposed some techniques that speed up the algorithm at the expense of harming the optimality of solutions. We implemented the algorithm in Java language and tested on the real-world model of Germany road network with Tesla's charging stations. The evaluation of our experiments shows that the computation time of the algorithm is 622 ms on average. Finally, we developed the web application 'Charge Here' where the same algorithm is applied.

Abstrakt

V této práci jsme se zaměřili na plánování tras pro elektromobily, kde je potřeba využít nabíjecích stanic k dobití baterie. Minimalizujeme cestovní náklady v modelu silniční sítě, kde cena za jednotku energie se může během dne lišit v souvislosti se strategií "Dynamic pricing". Abychom se vyhnuli nadbytečně dlouhým trasám, započítáváme do cestovních nákladů i dobu jízdy. Řidič si sám může určit, jaký význam pro něj má čas strávený na cestách. Navrhli jsme bikriteriální algoritmus, který počítá množinu optimálních tras. Množina tras obsahuje trasu s minimální dobou jízdy, trasu s minimalními cestovními náklady a několik alternativních tras. Náš algoritmus vychází z algoritmu "Bicriteria Shortest Path Algorithm". Tento algoritmus jsme rozšířili tak, aby splňoval omezení daná baterií v elektromobilu a zároveň umožnil nabíjení u nabíjecích stanic. Dále jsme navrhli některé techniky, které zrychlují algoritmus na úkor porušení optimality řešení. Navržený algoritmus jsme implementovali v jazyce Java a otestovali na modelu silniční sítě Německa s nabíjecími stanicemi od Tesla Motors. Podle výsledků evaluace je průměrný výpočení čas algoritmu 622 ms. Nakonec jsme vyvinuli webovou aplikaci "Charge Here", ve které je aplikován uvedený algoritmus.

Contents

Chapter 1

Introduction

The global population growth and urbanization are associated with the transport. Most motor vehicles are propelled by petrol or diesel. These types of fuel pollute the air and are often blamed for contributing to climate change. Regardless of the fact that petroleum reserves are running out, there is an increasing interest in the protection of the planet, thus the alternative fuel vehicles were developed, e.g., electric vehicles (EVs). The EVs bring new problems to the journey planning such as recuperation or battery constraints. The cruising range of the EV is limited because the capacity of the battery in the vehicle is small. Therefore, it is required to count properly the energy consumed by EV during the journey, because the EV driver does not want to get stuck in the middle of the journey. The energy consumption is influenced by various parameters, e.g., speed, elevation or weather conditions.

Since longer journeys are not feasible by EV per one charge cycle of the battery, it is crucial to plan the journey via charging stations. However, recharging the EV at the charging station lasts longer than refueling a standard petrol vehicle and the charging station has a limited number of stalls. These differences extends the total travel time of the journey and should be considered while planning the journey.

We assume, that the costs of the journey are also relevant to many EV drivers. Nowadays, the costs for recharging the battery of EV are stable, more or less. Nevertheless, the number of EV owners is growing rapidly and the EV traffic is hardly predictable. This will have a big impact on the electrical grid. We believe, that the **Dynamic Pricing** strategy is a way to balance the electrical grid. In this work, we will consider that the price per unit of energy might update anytime, anywhere. In other words, the price per unit of energy at charging station will be dependent on the time of a day and on the location of the charging station. The Dynamic Pricing strategy forbids us to use several speed-up techniques that are commonly used in journey planning. We would like to provide an algorithm that considers the Dynamic Pricing strategy as well as the EV battery constraints and computes the journey with minimal travel time and the journey with minimal travel costs. We found this problem challenging and actual. The fact that nobody considered Dynamic Pricing strategy in journey planning for EV before, is even more motivating. Also, we will develop the web application which should help an EV driver to plan the journeys comfortably on demand.

Chapter 2

Related Work

In this chapter, we introduce works that are closest to the subject of this thesis. Firstly, we provide a survey of the literature relating to journey planning for electric vehicles (EVs). Secondly, we review existing web applications.

2.1 Journey Planning for Electric Vehicles

Many articles, theses and papers were written about the journey planning for EVs. Most of them are focused on minimizing energy consumption or travel time. They integrates battery constraints into the graph algorithms that are based on the **Dijsktra's shortest path algorithm** [1]. The reason why the basic Dijkstra's algorithm cannot be used is that the battery could recuperate energy by braking or going downhill. Thus, the **negative edge weights** describing energy consumption are present in graph. Mostly, the simplified model of energy consumption is used, it considers the horizontal distance and the elevation change between two vertices.

The first contribution that compute the energy-optimal paths for EV is [2]. They formulated the energy-efficient path problem for EV as an instance of the **constrained shortest path problem** [3], where the battery constraints are considered. Then, they provided a label-correcting algorithm with worst case time complexity of $\mathcal{O}(n^3)$. This work was extended in [4] where **Johnson's potential shifting** [5] handles the negative weights and **A*** search algorithm [6] is used. This improved the time complexity to $\mathcal{O}(n^2)$. In [7] was introduced the **chaining of edges** with no loss of information about energy consumed during the path. It allowed the usage of the speed-up technique called **Contraction Hierarchies**, firstly introduced in [8]. This technique computes shortcuts between vertices, it is commonly used in large graphs. With such a pre-processing approach, [7] improved the time complexity of the algorithm to $\mathcal{O}(n \log n + m)$. They achieved average computation time of less than a second on the road network with more than million road segments. In [9] is provided a nontrivial bidirectional search algorithm with **Customizable Route Planning** technique [10], that answer queries within 0.3 ms on average. The algorithm was implemented in C++ and evaluated on the road network of Europe.

Note that all previous studies do not contain charging stations in the road graph. The battery switch stations are considered in [11] and the EV problem minimizes the number of

battery switches during the path. This approach uses the pre-processed graph that contains all shortest paths between charging stations. In the query time, the shortest paths from the origin vertex to all feasible charging stations are computed (**First Mile problem**), as well as the shortest feasible paths from charging stations to destination vertex (**Last Mile problem**). The solutions of the First Mile and Last Mile problem are added to the pre-processed graph. Then, the pre-processed graph is used to compute the requested journey.

The bachelor thesis [12], written by Jonas Sauer, deals with finding an energy-optimal journey for EV, where the recharging at the regular charging stations is allowed. The energy-optimal path problem is extended such that the number of visited chargers is minimized. The similar approach is also in the Zündorf's master thesis [13]. However, it aims to minimizing the travel time considering more types of charging stations, i.e., the charging stations with different charging speeds and the battery swapping stations. Both [12, 13] provided the algorithm that is based on **Multi-objective A*** search (MOA*) [15] that returns a Pareto sets of paths. Also the Contraction Hierarchies and the graph pre-processing is used. The solution use the piece-wise linear convex functions to model the charging process. In [13], the potential shifting is not used. He optimizes the travel time which has non-negative edge weights in the graph, the energy consumption is used as the constraint only. The part of this thesis was also published in the paper [14].

Nevertheless, the energy-efficient journeys may have disproportionate detours. In [16] is added additional criterion besides the criterion of energy consumption, such as travel time or length. The optimal journeys that solves their problems, probably satisfy the requirements of the EV driver better. Unfortunately, as in [11], only the switching charging stations are taken into account. Another approach with combined criteria is introduced in [17]. They consider the travel time and the energy consumption as the criteria. Furthermore, they use model with multiple speed limits for each road segment. The recharging is not allowed in the study. Their multicriteria algorithm with several speed-ups achieve computation time of 750 ms on average on the continental road network.

The Previous literature was focused on the grap-based algorithms only. A different approach for EV routing is introduced by Boston University in [18, 19]. They formulated **Mixed-Integer Nonlinear Programming** (MINLP) problem for EVs that minimizes travel time considering recharging at the charging stations.

Every mentioned work that solve problems for EV, consider the recuperation effect while driving downhill and prevents the overcharging and undercharging the battery of EV during a journey.

For more general insight into problems connected to EVs, we also mention [20] given by IEEE, it is the survey of works written before 2015. It analyzes the application of artificial intelligence to the major challenges that arise with deployment and management of EVs. It is mainly focused on the energy-efficient EV routing, charging station selection and integration of EVs into the smart grid.

Note that the main purpose of this work is to minimize travel costs. More precisely, money spent for the energy charged at charging stations during the journey. In the case, the price per unit of energy is the same for all charging stations, the problem is equivalent to the minimization of energy consumption. However, we consider the dynamic pricing strategy, then the prices are frequently updated. Thus, the price-optimal journey may not be neither the energy-efficient nor the fastest. Furthermore, the prices are unknown before

the query time and they depends on the arrival time to the charging station. It causes that several techniques used in the studies, mentioned in this section, cannot be directly applied.

2.2 Web Journey Planners for Electric Vehicles

Journey planner is a search engine which finds the optimal journey between two points, which is typically the shortest or the fastest journey. Several journey planners that are focused on EVs are available online. Most of them are under development, in beta version or area limited.

The first published web journey planner for EVs is the project of the South West College and Action Renewables called Egomap[1]. This planner compares the public transport to the EVs on CO_2 emissions. According to our test, it works in Ireland and Portugal only. The another interesting planner is the EV Trip Planner[2] developed by Ben Hannel. It predicts the energy consumption for the route. The physics based model that computes the energy consumption considers speed, air density, elevation. The EV Route [3] application by Controtex was developed mainly for Nissan Leaf and covers the UK, Ireland and Japan. Both the EV Trip Planner and the EV Route use the Open Charge Map API to locate the nearest chargers. The other planners that plan via chargers are the planner by Go Electric Stations[4] and the evRoutes[5].

The last type of available web applications are planners that use the Google Maps Directions API to compute the journeys and shows the charging stations nearby, e.g., the journey planners by KELAG[6], PlugShare[7] or Chargemap[8].

Nevertheless, the documentation of all these projects is unavailable, incomplete or none, so we can only guess what algorithms and techniques were used. Note that, none of the projects from major competitors on the market such as Google Maps[9], Bing Maps[10] or HERE Maps[11] provide journey planning for EVs.

[1] http://egomap.eu/
[2] http://evtripplanner.com/planner/2-7/
[3] http://evroute.controtex.com/
[4] http://goelectricstations.com/map-charging-stations.html
[5] http://evroutes.com/
[6] http://ev-charging.com/at/en/directions
[7] http://plugshare.com/
[8] http://chargemap.com/points/searchRoute
[9] http://maps.google.com
[10] http://bing.com/maps
[11] http://maps.here.com

Chapter 3

Problem Specification

In this chapter, we formalize the electric vehicle routing problem with recharging. Firstly, we recall several definitions from the graph theory that are useful for this work. Secondly, we describe the transport network for EVs. Thirdly, we introduce the exact model of such transport network as an extension of directed weighted graph. Finally, we specify two problems, the Earliest Arrival Problem for EV and the Minimum General Cost Path Problem for EV.

3.1 Preliminaries

3.1.1 Directed Weighted Graph:

The directed weighted graph is a graph $G = (V, E, f)$, where

- V is a set of vertices,
- E is a set of oriented edges,
- f is a weight function $f : E \mapsto \mathbb{R}$ which assigns a numeric value to an edge $e \in E$.

Note that the difference between the oriented edge $e = (u, v)$ from a vertex $u \in V$ to a vertex $v \in V$ and the non-oriented edge $e' = \{u, v\} = \{v, u\}$ between two vertices u, v. Only the oriented edges are used in this work, even if it is not explicitly mentioned.

3.1.2 Path

Given a graph $G = (V, E, f)$, the path $P = (v_1, \ldots, v_n) \in V^n$ is a sequence of vertices from a vertex $v_1 \in V$ to a vertex $v_n \in V$ such that exists an edge $e_i = (v_i, v_{i+1}) \in E$ for $i = 1, \ldots, n-1$ where $n = |P|$ is a number of vertices included in the path P. Then, we define the weight of the path as $w(P) = \sum_{i=1}^{n-1} f(e_i) \in \mathbb{R}$.

Shortest Path: Given a graph $G = (V, E, f)$, the shortest path $P^*_{u,v} \in G$ is a path $P = (u, \ldots, v)$ from a vertex $u \in V$ to a vertex $v \in V$ with the smallest weight $w(P^*_{u,v}) = \min(w(P)) \in \mathbb{R}$ of all possible paths $P \in G$ from u to v.

Constrained Path: Given a graph $G = (V, E, f)$ and a set of constraints C, the constrained path $P_{u,v,C} \in G$ is a path $P = (u, \ldots, v)$ from a vertex $u \in V$ to a vertex $v \in V$ such that every constraint $c \in C$ is satisfied.

Constrained Shortest Path: Given a graph $G = (V, E, f)$ and a set of constraints C, the constrained shortest path $P^*_{u,v,C} \in G$ is a constrained path from a vertex $u \in V$ to a vertex $v \in V$ with the smallest length $w(P^*_{u,v,C}) = \min(w(P)) \in \mathbb{R}$ of all possible constrained paths $P_{u,v,C} \in G$ from u to v.

3.2 Transport Network

The transport network is a system of locations and ways used for transporting objects or people. Various types of transport are included in the transport network, for example aviation, ship transport or land transport. Traveling by vehicles refers to a road transport which is a part of the land transport.

3.2.1 Road network

The road network is represented by junctions and roads between them. It is divided into road categories, where every road category (e.g. motorways or primary roads) has different restrictions for the transportation. The road network could be expressed as a graph structure where junctions are vertices and roads are edges.

3.2.2 Filling stations

Filling stations is a set of locations where it is possible to refill a vehicle with a fuel, e.g., gasoline, diesel fuel or electric energy. Petrol vehicles are refueled at the fueling stations and EVs are recharged at the charging stations. Considering a time, a driver of petrol vehicle does not bother to stop for refueling petrol because the process takes a few minutes. Whereas for a driver of EV, recharging is less pleasant. Grid of charging stations is sparse and recharging a vehicle takes at least half an hour. The EV driver has a choice of several types of charging stations. A lot of charging stations provide very slow charging. One of the fast charging station is the Tesla supercharger[1], where recharging to the 80 % of battery capacity takes approximately 40 minutes. Superchargers are placed carefully such that the recharge to 80 % of battery capacity should suffice to reach another supercharger. A Special type of charging station is the swapping station, where the battery of EV is replaced with another fully charged battery and it is possible to continue the journey without recharging.

3.2.3 Dynamic Pricing

Dynamic pricing is a business strategy that sets a price to products based on the current state of the market. We consider dynamic pricing strategy to determine the price per unit of energy on the charging station. Prices may be affected by various external influences,

[1] http://tesla.com/supercharger

e.g., situation of the electrical grid. In rush hours, energy could cost more than at night. We take into account that the price is dependent on the location of the charging station and the exact time of visit.

3.2.4 Time Monetization

As Benjamin Franklin mentioned the phrase *"Time is money"* in his *"Advice to a Young Tradesman"* [21], we are convinced that travel time is crucial for drivers. Optimizing the amount of money spent for recharging could lead to finding a journey with a big detour. To give a balance between time and the amount of money spent during a journey, we add the parameter Φ that could include the value of travel time to the travel costs. The parameter is set by the EV driver. If the driver likes driving, there is also the opportunity to turn off this feature by setting the parameter to zero.

3.3 Model of Transport Network for Electric Vehicles

Our transport network model is a graph which contains junctions and chargers as vertices and roads as edges. Junctions and chargers are described by a pair of GPS coordinates and its elevation. The road is an oriented connection from one junction to another. Road description, e.g., the road length or energy consumption, is expressed by functions.

Road Graph: The Road Graph $G_r = (V_r, E_r, M_r, g, \delta, \nu, R_r, \varrho)$ is a directed weighted graph, where

- V_r is a set of vertices describing the junctions, where a vertex $v \in V_r$ is defined by its latitude $lat_v \in \mathbb{R}$, longitude $lon_v \in \mathbb{R}$ and elevation $elev_v \in \mathbb{R}$,
- E_r is a set of edges describing the roads between junctions,
- M_r is a set of supported EV models in the graph,
- a function $g : E_r \times M_r \mapsto \mathbb{R}$ assigns an energy consumption to an edge $e \in E_r$ and to an EV model $m \in M_r$,
- a function $\delta : E_r \mapsto \mathbb{R}_0^+$ assigns a distance to an edge $e \in E_r$,
- a function $\nu : E_r \mapsto \mathbb{R}_0^+$ assigns an average speed to an edge $e \in E_r$,
- R_r is a set of all road categories that occur in the graph,
- a function $\varrho : E_r \mapsto R_r$ assigns a road category to an edge $e \in E_r$.

Charger Extended Road Graph: The Charger Extended Road Graph (CERG) $G'_r = (V'_r, E_r, M_r, g, \delta, \nu, R_r, \varrho, \phi, \varsigma)$ is a directed weighted graph, where

- $V'_r = V_r \cup S$ is the union of the set of vertices V_r from the Road Graph G_r and the set of vertices S, where a vertex $s \in S$ is a charger defined by its latitude $lat_s \in \mathbb{R}$, longitude $lon_s \in \mathbb{R}$ and elevation $elev_s \in \mathbb{R}$,

- E_r is a set of edges in G_r describing the roads between junctions,
- M_r is a set of supported EV models in the graph,
- a function $g : E_r \times M_r \mapsto \mathbb{R}$ assigns an energy consumption to an edge $e \in E_r$ and to an EV model $m \in M_r$,
- a function $\delta : E_r \mapsto \mathbb{R}_0^+$ assigns a distance to an edge $e \in E_r$,
- a function $\nu : E_r \mapsto \mathbb{R}_0^+$ assigns an average speed to an edge $e \in E_r$,
- R_r is a set of all road categories that occur in the graph,
- a function $\varrho : E_r \mapsto R_r$ assigns a road category to an edge $e \in E_r$,
- a function $\phi : S \times \mathbb{R}_0^+ \mapsto \mathbb{R}_0^+$ assigns a price per unit of energy to a charger $s \in S$ at time $t \in \mathbb{R}_0^+$,
- a function $\varsigma : S \times M_r \mapsto ct_{s,m}$ assigns a charging function to a charger vertex $s \in S$ and an EV model $m \in M_r$.

3.3.1 Model of Electric Vehicle

Let us define the simple EV model $m \in M_r$ such that $m = \{b_{min}, b_{max}\}$ where $b_{max} \in \mathbb{R}_0^+$ is a maximum of battery capacity and $b_{min} \in [0, b_{max}]$ is a minimum of battery capacity. Note that, the b_{min} is not fixed to zero value. It gives an ability to each EV model to specify the minimum battery state of charge which is safe to not get stranded in the middle of the journey.

Figure 3.1: Tesla Model S.

3.3.2 Battery State of Charge

Let us have a Charger Extended Road Graph (CERG) G'_r and a EV model $m = \{b_{min}, b_{max}\}$. We have to allow the possibility of recharging the battery in a vertex $v \in V'_r$, we distinguish the arrival (incoming) state of charge (SoC) $b_{in}(v) \in [b_{min}, b_{max}]$ and the departure (outgoing) SoC $b_{out}(v) \in [b_{min}, b_{max}]$ of the vertex v. In other words, the $b_{in}(v)$ is a SoC before the recharging and the $b_{out}(v)$ is a SoC after the recharging. Then, the amount of energy $b_{ch}(v) \geq 0$ recharged in a vertex v is the difference between the departure SoC $b_{out}(v)$ and

the arrival SoC $b_{in}(v)$, formally written as $b_{ch}(v) = b_{out}(v) - b_{in}(v)$. Note that the departure SoC is greater or equal than the arrival SoC ($b_{out}(v) \geq b_{in}(v)$) if the vertex $v \in S$ is a charger, otherwise $b_{in}(v) = b_{out}(v)$. The battery recuperation causes that CERG G'_r may contain edges with a negative energy consumption. This means that SoC will increase or decrease after traversing the edge. While computing a path, the battery SoC has to stay in the battery interval $[b_{min}, b_{max}]$, because overcharging and undercharging the battery are not allowed. Since it is unknown how much energy is charged on each visited charging station, the definition of path does not suffice to describe a journey for EV in G'_r. To describe a journey in CERG G'_r properly, we also need a function b_{ch} that assigns the amount of charged energy to the visited charger. According to [12] we define a function that computes the arrival SoC and the departure SoC of each vertex that is contained in the constrained path. We extended the function from [12] that recharging in the vertex is possible.

Let us have a set of constraints $C_{ev} = \{b_{min}, b_{max}\}$, a constrained path $P_{v_1,v_n,C_{ev}} \in G'_r$, where $n = |P_{v_1,v_n,C_{ev}}|$ is the number of vertices included in $P_{v_1,v_n,C_{ev}}$ and a function $b_{ch} : S \cap P_{v_1,v_n,C_{ev}} \mapsto \mathbb{R}_0^+$. Then, we define the function $b_{in} : P_{v_1,v_n,C_{ev}} \times M_r \mapsto [-\infty, b_{max}]$ assigns an arrival SoC to the vertex $v_i \in P_{v_1,v_n,C_{ev}}$ for each $i = 1, \ldots, n$, as follows:

$$b_{in}(v_i, m) = \begin{cases} b_{init}, & \text{if } i = 1 \\ \min(b_{in}(v_{i-1}, m) + b_{ch}(v_{i-1}) - g((v_{i-1}, v_i), m), b_{max}), & \text{if } v_i \in S \\ \min(b_{in}(v_{i-1}, m) - g((v_{i-1}, v_i), m), b_{max}), & \text{otherwise} \end{cases}$$

where b_{init} is the initial SoC in the vertex v_1 where g is a function of CERG G'_r that assigns an energy consumption to a tuple of an edge and an EV model. The function b_{in} adds the amount of energy recharged at the charger $s \in S$ to the arrival SoC in previous vertex in the path. Then, subtracts the amount of energy consumed by traversing the edge. When overcharging would occur, the function b_{in} return the maximum of the battery capacity b_{max}. In contrast to the overcharging, the undercharging affects the feasibility of the path. Also, we define the function $b_{out} : P_{v_1,v_n,C_{ev}} \times M_r \mapsto [-\infty, b_{max}]$ assigns a departure SoC to $v_i \in P_{v_1,v_n,C_{ev}}$ for each $i = 1, \ldots, n$ as follows:

$$b_{out}(v_i, m) = \begin{cases} b_{in}(v_i, m) + b_{ch}(v_i), & \text{if } v_i \in S \\ b_{in}(v_i, m), & \text{otherwise} \end{cases}$$

Then, the path $P_{v_1,v_n,C_{ev}} = (v_1, \ldots, v_n)$ is feasible by EV if and only if the $b_{in}(v_i) \in [b_{min}, b_{max}] \wedge b_{out}(v_i) \in [b_{min}, b_m ax]$ holds for $i = 1, \ldots, n$.

3.3.3 Approximated Consumption Function

We provide a simplified model of the consumption function g in the CERG G'_r, similarly as in [13, 7]. This model use the elevation change and the length between two vertices to compute an energy consumption of the edge. Given the vertices $u, v \in V'_r$, their elevation change $\Delta elev_e = elev_v - elev_u$ and a function $\delta \in G'_r$, we define a function $g : E \times M_r \mapsto R$ that assigns an energy consumption $g(e, m)$ to every edge $e = (u, v) \in E$ traversed by an EV model m in the CERG G'_r such that:

$$g(e, m) = \begin{cases} \kappa \cdot \delta(e) + \lambda \cdot \Delta elev_e, & \text{if } \Delta elev_e \geq 0 \\ \kappa \cdot \delta(e) + \alpha \cdot \Delta elev_e, & \text{otherwise} \end{cases}$$

where $\kappa, \lambda, \alpha \geq 0$ are tuning constants. The function g distinguishes an energy consumed while driving uphill ($\Delta elev_e > 0$) from driving downhill ($\Delta elev_e < 0$). While driving downhill, the function g could return a negative value, it represent the recuperation of the battery. Note that we do not use any the property of EV model m to influence the energy consumption, however this function suffices for our requirements. For now, we can affect the consumption by setting the tuning constants.

3.3.4 Charging Function

Given a sequence of continuous intervals ($I_1 = [x_1, x_2), I_2 = [x_2, x3), \cdots, I_k = [x_k, \infty)$), a piecewise linear function $f : X \mapsto Y$ is a function that is linear on each interval I_i for $i = 1, \cdots, k$. Then each point $(x_i, f(x_i))$ for $i = 1, \cdots, k$ is the supporting vector of f. Given a sequence of supporting vectors $((x_1, y_1), \cdots, (x_k, y_k))$ where $x_i < x_{i+1}$ for $i = 1, \cdots, k-1$, the piecewise linear function is uniquely defined as follows:

$$f(x) = \begin{cases} \frac{(x-x_1)(y_2-y_1)}{x_2} + y_1, & \text{if } x_1 \leq x < x_2 \\ \frac{(x-x_2)(y_3-y_2)}{x_3} + y_2, & \text{if } x_2 \leq x < x_3 \\ \vdots & \\ \frac{(x-x_{k-1})(y_k-y_{k-1})}{x_k} + y_{k-1}, & \text{if } x_{k-1} \leq x < x_k \end{cases}$$

Then, the charging function $ct_{s,m} : [b_{min}, b_{max}] \mapsto \mathbb{R}_0^+$ is a non-decreasing piecewise linear function that assigns a time needed to recharge the EV model $m = \{b_{min}, b_{max}\} \in M_r$ on the charger $s \in S$ from the minimum battery SoC b_{min} to the desired SoC $b_d \in [b_{min}, b_{max}]$. We reused this model of charging functions from [13].

3.3.5 Journey for Electric Vehicle

Because the constrained path in the CERG G'_r is not sufficient to describe recharging during the journeys, we provide another definition. Given a G'_r, an EV model $m = \{b_{min}, b_{max}$ and a set of constraints $C_{ev} = \{b_{min}, b_{max}\}$, we define the journey $J(u, v, C_{ev}) = \{P_{u,v,C_{ev}}, b_{ch}\} \in G'_r$ as an union of the constrained path $P_{u,v,C_{ev}} \in G'_r$ and the function $b_{ch} : S \cap P_{u,v,C_{ev}} \mapsto \mathbb{R}_0^+$ that assigns the amount of charged energy $b_{ch}(s)$ to a charger vertex $s \in S \cap P_{u,v,C_{ev}}$ contained in the constrained path $P_{u,v,C_{ev}}$. The journey $J(u, v, C_{ev})$ is feasible by EV if and only if the arrival SoC $b_{in}(v_i)$ and the departure SoC $b_{out}(v_i)$ is in the interval $[b_{min}, b_{max}]$ for $i = 1, \cdots, n$ where n is the number of vertices in the journey. The journey J has two criteria, a travel time $tt(J, m) \in \mathbb{R}^+$ and travel costs $tc(J, m) \in \mathbb{R}_0^+$.

Driving Time Function: Let us define the driving time function $dt : E \mapsto \mathbb{R}$ which assigns a time needed to traverse an edge e without recharging, as follows:

$$dt(e) = \frac{\delta(e)}{\nu(e)}$$

where $\nu : E \mapsto \mathbb{R}$ is a function assigning an average speed of EV to an edge $e \in E$ and $\delta(e)$ is a function that assigns a length to the edge e.

Travel Time Function: Given the journey J, we define the travel time criteria $tt(J, m) \in \mathbb{R}^+$ for the journey J surpassed by an EV model $m \in M_r$ as

$$tt(J, m) = \sum_{i=1}^{n-1} tt(e_i, m)$$

where $e_i = (v_i, v_{i+1}) \in E_r$ is i-th edge in the journey J, $n = |J|$ is the number of vertices contained in the J and $tt : E_r \times M_r \mapsto \mathbb{R}^+$ is the travel time function that assigns a travel time to e_i. The travel time function contains the driving time as well as the charging time, it is defined, as follows:

$$tt(e_i, m) = \begin{cases} dt(e_i) + ct_{v_i,m}(b_{out}(v_i, m)) - ct_{v_i,m}(b_{in}(v_i, m)), & \text{if } v_i \in S \\ dt(e_i), & \text{otherwise.} \end{cases}$$

Travel Cost Function: Given the journey J, we define the travel cost criteria $tc(J, m) \in \mathbb{R}_0^+$ for the journey J surpassed by an EV model $m \in M_r$ as follows:

$$tc(J, m) = \sum_{i=1}^{n-1} tc(e_i, m)$$

where $e_i = (v_i, v_{i+1}) \in E_r$ is i-th edge in the journey J, $n = |J|$ is the number of vertices in J and $tc : E_r \times M_r \mapsto \mathbb{R}_0^+$ is the travel cost function that assigns a travel costs to e_i. The travel costs function contains the monetized travel time as well as the costs for charged energy. It is defined as follows:

$$tc(e_i, m) = \begin{cases} \Phi \cdot tt(e_i, m) + \phi(v_i, t_{dt}(v_i, m)) \cdot (b_{out}(v_i, m) - b_{in}(v_i, m)), & \text{if } v_i \in S \\ \Phi \cdot tt(e_i, m), & \text{otherwise} \end{cases}$$

where $t_{dt}(v_i)$ is the departure time in the vertex $v_i \in J$ defined as

$$t_{dt}(v_i, m) = \begin{cases} t_{init}, & \text{if } i = 1 \\ t_{init} + tt(J(v_1, v_i, C_{ev}), m), & \text{otherwise} \end{cases}$$

3.4 Electric Vehicle Routing Problem with Recharging

EV routing covers the problematics connected with computing journeys for EVs. EVs has several restrictions that was not taken into account in routing for ordinary cars. Cruising range of EV is small and recharging takes more time on the charging station. Also, the battery may be recuperated during the ride. The recuperation occurs while decelerating or driving downhill, it causes that several edges with negative weight are present in the CERG. Furthermore, we consider **dynamic pricing** to determine the price per unit of energy. In this section, we define a Electric Vehicle Routing Problem with Recharging where the CERG G'_r is given.

Electric Vehicle Journey Request: We define an EV request $q_{ev} = (o, d, t_{init}, m, b_{init}, \Phi)$, where

- $o \in V'_r$ is an origin vertex in G'_r,
- $d \in V'_r$ is a destination vertex in G'_r,
- $t_{init} \in \mathbb{R}_0^+$ is a departure time from the origin vertex o,
- $m = \{b_{min}, b_{max}\} \in M_r$ is an EV model that will be driven during the journey,
- $b_{init} \in [b_{min}, b_{max}]$ is an initial SoC at the origin vertex o,
- $\Phi \in \mathbb{R}_0^+$ is a time monetization constant that describes the price per unit of time.

3.4.1 Earliest Arrival Problem for Electric Vehicles

Given a Charger Extended Road Graph G'_r and an EV request q_{ev}. The Earliest Arrival Problem for Electric Vehicles with Recharging (EAP-EV) asks for a journey $J^* = \{P_{u,v,C_{ev}=\{b_{min},b_{max}\}}, b_{ch}\} \in G'_r$ that is feasible by EV. The journey $J*$ starts at the origin vertex o with the initial SoC b_{init} no earlier than t_{init} and ends in the destination vertex d with the minimum possible travel time $tt(J^*, m)$. The function b_{ch} assigns the amount of charged energy to every charging station contained in the journey J^*. The journey $J*$ has to satisfy the set of constraints C_{ev} from the journey definition 3.3.5.

3.4.2 Minimum General Cost Path Problem for Electric Vehicles

Given a Charger Extended Road Graph G'_r and an EV request q_{ev}. The Minimum General Cost Path Problem for Electric Vehicles (MGCPP-EV) asks for a journey $K^* = \{P_{u,v,C_{ev}=\{b_{min},b_{max}\}}, b_{ch}\} \in G'_r$ that is feasible by EV. The journey K^* starts at the origin vertex o with the initial SoC b_{init} no earlier than t_{init} and ends in the destination vertex d with the minimum possible travel costs $tc(K^*, m)$. The function b_{ch} assigns the amount of charged energy to every charging station contained in the journey p. The journey $K*$ has to satisfy the set of constraints C_{ev} from the journey definition 3.3.5.

3.4.3 Electric Vehicle Journey Response

We define the EV journey response r_{ev} as a set of journeys that contains at least a journey J^* and a journey K^*, where

- $J^* = \{P_{u,v,C_{ev}}, b_{ch}\}$ is the optimal journey that is asked by the EAP-EV,
- $K^* = \{P_{u,v,C_{ev}}, b_{ch}\}$ is the optimal journey that is asked by the MGCPP-EV.

The EV response wraps the solutions of both problems to a set, it may also contain an additional set of alternative journeys.

Chapter 4

Solution Approach

We solve the both problems from Section 3.4 by multicriteria graph-based algorithm. The whole process of solving the problem is shown in Figure 4.1. Firstly, we describe the exact structure of the Search Graph. Secondly, we provide the multicriteria algorithm that computes a set of n feasible journeys $(J_1 = J^*, J_2, \ldots, J_n = K^*)$, where J^* is the solution of the EAP-EV, K^* is the solution of MGCPP-EV and each J_i for $i = 2, \ldots, n-1$ is the alternative feasible journey that is a trade-off between J^* and K^*. We call such a set the EV response r_{ev}. Thus, we compute both journeys by running the multicriterial algorithm once. Finally, we propose a few techniques that speeds up the algorithm.

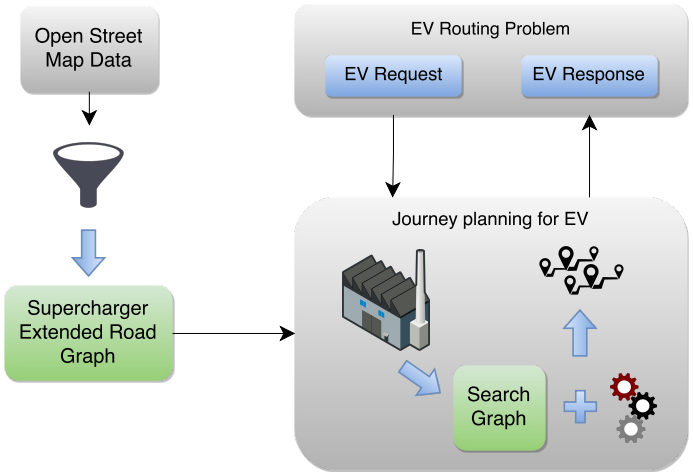


Figure 4.1: Journey planner schema. The factory represents the process of building the Search Graph, the funnel illustrates filtering the input (OpenStreetMap) data. The gears interpret the algorithm used to compute the journeys. The paths with map markers symbolize the computed journeys J^* and K^*.

4.1 Search Graph

Since the CERG graph G'_r is the general model of transport network for EVs, the weight functions are not adapted to solve problems with graph-based algorithms. For our purpose, we build another graph in query time that is more appropriate to solve problems introduced in Chapter 3. We call such a graph, the **Search Graph**. The Search Graph is built in query time, so the EV request q_{ev} is already known. The CERG graph G'_r is also given. Then, the Search Graph is a weighted graph $G_{search} = (V_{search}, E_{search}, tt', tc', dt, cp, \phi, \varsigma)$, where

- V_{search} is a set of vertices,
- E_{search} is a set of edges,
- $tt' : E_{search} \times S \times [b_{min}, b_{max}] \times [b_{min}, b_{max}] \mapsto \mathbb{R}^+$ is a weight function that assigns a travel time to an edge traversed by the EV model m. Travel time is contains driving time as well as time spent while recharging at charger $s \in S$ from a SoC $b_{in_s} \in [b_{min}, b_{max}]$ to a chargeable SoC $b_{out_s} \in [b_{min}, b_{max}]$,
- $tc' : E_{search} \times S \times [b_{min}, b_{max}] \times [b_{min}, b_{max}] \times \mathbb{R}_0^+ \mapsto \mathbb{R}_0^+$ is a weight function that assigns a travel costs to an edge traversed by the EV model m. Travel costs are derived from costs while driving and from costs for recharging at charger $s \in S$ from a SoC $b_{in_s} \in [b_{min}, b_{max}]$ to a chargeable SoC $b_{out_s} \in [b_{min}, b_{max}]$ with a price per unit of energy $\pi_s \in \mathbb{R}_0^+$,
- $dt : E_{search} \times [b_{min}, b_{max}] \mapsto \mathbb{R}^+$ is a weight function that assigns a driving time to an edge traversed by the EV model m according to the departure SoC b_{out_u},
- $cp : [b_{min}, b_{max}] \times E_{search} \mapsto \mathbb{R}$ is an energy consumption function that assigns an energy consumption to an edge traversed by the EV model m according to departure SoC in u.
- a function $\phi : S \times \mathbb{R}_0^+ \mapsto \mathbb{R}_0^+$ assigns a price per one unit of energy to a charger $s \in S$ at time $t \in \mathbb{R}_0^+$,
- a function $\varsigma : S \mapsto ct_{s,m}$ assigns a charging function to a charger vertex $s \in S$ visited by the EV model m.

Note that the search graph G_{search} is dependent on EV model m, so journeys computed in this graph may not be feasible by other EV models. The functions tt' and tc' are adapted such that it is possible to recharge in arbitrary charger seen earlier in the journey, they are defined as follows:

$$tt'(e, s, b_{in_s}, b_{out_s}) = \begin{cases} dt(e, b_{out_u}) + ct_{s,m}(b_{out_s}) - ct_{s,m}(b_{in_s}), & \text{if } u \in S \wedge s \neq \perp \\ dt(e, b_{out_u}), & \text{otherwise.} \end{cases}$$

$$tc'(e, s, b_{in_s}, b_{out_s}, \pi_s) = \begin{cases} \Phi \cdot tt'(e, s, b_{in_s}, b_{out_s}) + \pi_s \cdot (b_{out_s} - b_{in_s}), & \text{if } u \in S \wedge s \neq \perp \\ \Phi \cdot tt'(e, s, b_{in_s}, b_{out_s}), & \text{otherwise} \end{cases}$$

4.2 Shortest Path Algorithm

The shortest path problem is a problem of computing a path $P^*_{u,v}$ from $u \in V$ to $v \in V$ in a Graph G such that the path $P^*_{u,v}$ has the smallest weight $w(P^*_{u,v}) = min(w(P))$ of all paths $P_{u,v} \in G$ from u to v. If there is no path, we denote the shortest path weight $w(P^*) = \infty$ as infinitely long. If graph G is a strongly connected component then a shortest path $P^*_{u,v}$ always exists.

The problem defined above is single-pair variation of shortest path problem. There are more variation as single-source, single destination and all-pairs shortest path problem. Single-source shortest path problem computes shortest path from source vertex u to all other vertices in the graph G. Single-destination shortest path problem computes shortest path from all vertices in the graph G to a destination vertex v. All-pairs shortest path problem computes shortest path for every pair of vertices (u, v) in a graph G.

4.2.1 Dijkstra's Algorithm

The Dijkstra's shortest path algorithm [1] solves the shortest path problem. The original Dijkstra's algorithm published in 1959 finds the shortest path between start and goal vertices in a weighted graph with **non-negative** edge weights. In other words it solves single-pair shortest path problem (SP-SPP). Also, by simple modification we can solve the single-source (SS-SPP), single-destination (SD-SPP) and all-pairs (AP-SPP) shortest path problem. But solving AP-SPP on large graphs with Dijkstra's algorithm is slower than Floyd-Warshall algorithm [22, 23]. Solving SD-SPP by Dijkstra's algorithm may be done in two variations. First variation is finding shortest path from all starting vertices to one destination. But better approach is to convert the problem from SD-SPP to SS-SPP such that we run the algorithm from destination and investigate incoming edges except outgoing, we call it the backward search.

Given an directed weighted graph $G = (V, E, f)$ and an origin vertex $o \in V$, we describe the process of Dijkstra's algorithm to solve the single-source shortest path problem. In initialization phase, the algorithm sets distance label $l_o = 0$ to origin vertex o and $l_v = \infty$ to all other vertices $v \in V \backslash \{o\}$. The origin is added to priority queue Q, which sorts the vertices by its labels in ascending order. In each iteration, the vertex $u \in V$ with the minimum label l_u is polled from the priority queue Q. Then all edges $e = (u, v) \in E$ outgoing from the vertex u are inspected. For every successor v is created new label $l'_v = l_u + f(e)$, if $l'_v < l_v$ then label l_v is replaced by l'_v and v is added to the priority queue Q. Algorithm ends when priority Q is empty. In the case of single-pair shortest path problem, the algorithm ends when the label of destination vertex is polled from the queue Q.

The Dijkstra's algorithm has the property that once the vertex $u \in V$ is polled from the Q, then there does not exist a shorter path $P_{o,u}$ from the vertex o to the vertex u, it is also known as **label-setting property**. In other words, every vertex in the graph is polled no more than once. This algorithm returns labels with the smallest weights of the path as the solution, not the path itself. For the path retrieval, the pointer of parent label is added to each label during the computation.

The shortest path computed by this algorithm may not be feasible by EV (see Figure 4.2). Therefore basic version of Dijkstra's algorithm can not be used.

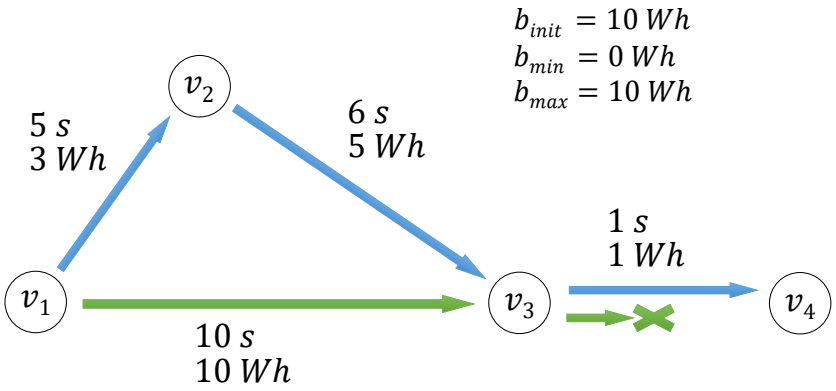


Figure 4.2: Let us consider the optimization on the travel time criteria and none of the vertices is charging station. The green path from v_1 to v_3 is optimal. By expanding the v_3, we get the path $P_g = (v_1, v_3, v_4)$ with the travel time of 11 s. Nevertheless, the continuation to the vertex v_4 is not possible, because the battery SoC is zero. If we consider discarding unfeasible labels. Then, the solution is that the v_4 is unreachable from v_1, which is not true. Thus, we cannot discard the blue slower path in v_3 because is has better battery SoC and it is the only way to reach the v_4. This picture also shows that one vertex may have more labels that cannot be discarded.

4.3 Constrained Shortest Path Algorithm

The constrained shortest path problem (CSPP) is an extension of the shortest path problem such that the computed path has to satisfy a set o constraints. Given an directed weighted graph $G = (V, E, f)$, a set of constraints C, an origin $o \in V$ and a destination $d \in V$. The CSPP asks for a constrained path $P^*(u, v, C)$ that has the smallest weight $w(P^*) = min(P)$ of all constrained paths P that satisfies a set of constraints C.

It is possible to say that the formulation of our problems is an extension of CSPP, because the journey is nothing else than the constrained shortest path, accompanied by the function to retrieve the amount of charged energy by each visited charging station. The CSPP problem is usually solved by a modification of Dijkstra's algorithm that is still single-criterion. However, the time complexity is much worse. Since we have two problems with battery constraints, we would have to run the algorithm twice, with the travel time criteria and with the travel cost criteria. However, the travel costs are often affected by the travel time, thus the search space may become really similar for both problems. We decided to merge these two criteria and provide the multicriterion algorithm with constraints. The advantage of this decision is that we will solve both problems at once. Furthermore, the outcome of the algorithm contain alternative solutions that are the trade-off between the minimum travel time and the minimum travel cost criteria.

4.3.1 Bicriteria Shortest Path Algorithm

As a basis of our algorithm, we use the Bicriteria Shortest Path (BSP) algorithm [24, 25] that does not satisfy any constraints. The BSP is an extension of Dijkstra's algorithm, that allow us to compute a set of journeys that contains an optimal journey for each criteria. The outcome also contains alternative paths representing a trade-off between two criteria.

Since we have two criteria, the label from Dijkstra's algorithm has to be extended. Given the graph G'_r and the EV request q_{ev}. Let us have a label $l_u = (\tau, \pi)$ of a vertex $u \in V'_r$, where $\tau = tt(J, m)$ is the travel time and $\pi = tc(J, m)$ are the travel costs of the path $J_{o,u} \in G'_r$ from the origin vertex $o \in V'_r$ to the vertex $u \in V'_r$. In the Dijkstra's algorithm, we replace the label when the newly visited label of the same vertex had lower value. In this case, it is not that straightforward because we optimize two criteria at the same time. Then, we need to determine whether the label would be discarded or not. The label l_u is dominated by another label $l'_u = (\tau', \pi')$, signed $(l_u \prec l'_u)$, if following conditions holds:

$$\tau \geq \tau' \tag{4.1a}$$

$$\pi \geq \pi' \tag{4.1b}$$

$$\tau \neq \tau' \vee \pi \neq \pi' \tag{4.1c}$$

In other words, we can discard only the label that is worse in both criteria. The **Pareto set** L_u is a set of labels assigned to vertex $u \in V'_r$ where each label $l_u \in L_u$ is not dominated by any other label $l'_u \in L_u$. In case of Dijkstra's basic approach, every vertex has one label only. In this extended algorithm, every vertex $u \in V'_r$ has a pareto set L_u which may contain an arbitrary number of labels.

In the initialization phase, the algorithm adds the origin label $l_o = (0, 0)$ to the pareto set L_o, pareto sets of all other vertices are empty. Also, the origin label l_o is added to the priority queue Q. In the priority queue, labels are sorted firstly by travel time, secondly by travel costs. Note that this time, we add the labels to the queue, not the vertices. In each iteration, the label l_u is polled from the top of the priority queue Q. Then all edges $e = (u, v) \in E_r$ outgoing from u are inspected. For every successor v is created a new label $l'_v = l_u + (tt(e, m), tc(e, m))$. If the label l'_v is not dominated by any label $l_v \in L_v$, then the label l'_v is added to L_v and Q. At the same time, each label $l_v \in L_v$ that is dominated by newly created label l'_v is removed from L_v and Q. Algorithm terminates when Q is empty. The solution of the algorithm is the Pareto set L_d of the destination vertex d. This algorithm is **label setting** because the functions tt and tc are non-negative. In this case it means, that once the vertex u is polled from Q then the actual label l_u cannot be dominated anymore, we call such a label as **settled**. If we stop the algorithm after the first label l_d of the destination vertex is polled from Q, we get only the path with the minimum travel time.

The basic BSP algorithm does not support the battery constraints nor the charging at charging stations. We have to do several modifications to make this algorithm applicable for our problems. Firstly, we need to track the battery SoC of EV model in each vertex contained in the journey. The battery SoC is changed, according to the energy consumed by the EV model, during the traversing an edge. We consider battery recuperation when driving downhill, thus some of the edges have negative energy consumption. In papers that optimizing energy consumption, the potential shifting technique [5] is used to get rid of

negative weights. For our purpose, it is not needed because we use energy consumption to compute a SoC which is the constraint only. Note that negative cycles are not occurred in a graph G'_r considering the g as a weight function, because it is not possible to arrive to a visited position with greater SoC than before, without recharging. However, in G'_r are charging stations that violates this assumption. Moreover, the situation of returning to the vertex later with recharged energy is appreciated, when the EV driver is forced to do a detour to a charging station. Fortunately, the SoC is limited by b_{max}, thus the negative cycle cannot cause the infinite loop of the algorithm.

4.3.2 Modified Consumption Function

The consumption function g in G'_r does not consider battery constraints of the EV model m. During the computation we must consider situation when traversing an edge will exceeds the maximum SoC b_{max} or surpass the minimum SoC b_{min}. Similarly as in [12], let us define an energy consumption function $c : [b_{min}, b_{max}] \times E_{search} \times M_r \mapsto \mathbb{R}$ which returns energy consumed while traversing an edge $e = (u, v) \in E_{search}$ with an EV model m from the vertex u and a departure SoC $b_{out}(u) \in [b_{min}, b_{max}]$ as follows:

$$c(b_{out}(u), e, m) = \begin{cases} \infty, & \text{if } b_{out}(u) - g(e) < b_{min} \\ g(e), & \text{if } b_{out}(u) - g(e) \in [b_{min}, b_{max}] \\ b_{out}(u) - b_{max}, & \text{if } b_{out}(u) - g(e) > b_{max} \end{cases}$$

This function regulates the energy consumption assigned by function g depending on the departure SoC $b_{out}(u)$ in the vertex u. If the edge is not feasible by EV model m, the function returns the infinity value. If the maximum of battery capacity b_{max} would be exceeded by traversing an edge, the function c returns a value that recuperate the battery to the maximum of battery capacity b_{max}.

4.3.3 Charging on the Charger

While the charger $s \in S$ is visited during the computation, we are facing the problem that we do not know much energy is optimal to be charged. Rather than trying all options from the interval $[b_{in}(s), b_{max}]$, we use the concept of recharging from [13]. The purpose is that we compute the amount of charged energy on the charger retrospectively. If we visit the first charger, we mark the arrival SoC $b_{in}(s)$ and pointer on the charger s. When the following charger is visited, we derive from the information about the path between chargers, how much energy is charged on the last seen charger s. Then the charging time and travel costs are added to the following label. When we reach the destination node, it is needed to add another last correcting label that contains charging on the last seen charger. On the other hand, it complicates the domination of labels during the search, because labels with the different s may not be dominated.

Consumption Profile Function: Since we do the recharging retrospectively, tracking the SoC with the consumption function c is not possible. As in [7, 13, 12], we profile the energy consumption from the actual charger to the next charger by keeping three values only, the minimum SoC $in_e \in [-b_{max}, b_{max}]$ to traverse an edge, energy consumed $cost_e \in$

$[-b_{max}, b_{max}]$ by traversing an edge (if the battery is not overcharged) and the maximum possible SoC $out_e \in [-b_{max}, b_{max}]$ after traversing an edge. Then the consumption profile function $cp_e(b, e, m) : [b_{min}, b_{max}] \times E_{search} \times M_r \mapsto \mathbb{R}$ is defined as follows:

$$cp_e(b_{out}(u), e, m) = \begin{cases} \infty, & \text{if } b_{out}(u) < in_e \\ b_{out}(u) - out_e, & \text{if } b_{out}(u) - cost_e > out_e \\ cost_e, & \text{else.} \end{cases}$$

For a single edge $e \in E$ is consumption profile function created by setting:

$$\begin{aligned} in_e &= \max\{b_{min}, g(e)\}, \\ out_e &= \min\{b_{max}, b_{max} - g(e)\}, \\ cost_e &= \max\{g(e), -b_{max}\}. \end{aligned}$$

If $g(e) > b_{max}$, then the edge e is not feasible and the function $cp(e, b_{out}(u), m)$ is undefined. Note that the functions c and cp are equivalent for a single edge. However, we need to profile an energy consumption for more edges chained in the path. The advantage of using the function cp is that after the edges are chained, we are still able to describe the energy consumption by three values, while traversing both edges consecutively. Thus, we define how consumption profiles are chained. Given two edges $e_1 = (u, v), e_2 = (v, w) \in E$ and their consumption profiles cp_{e_1}, cp_{e_2}, we get chained consumption profile $cp_{e_1 \circ e_2}$ by setting:

$$\begin{aligned} in_{e_1 \circ e_2} &= \max\{in_{e_1}, cost_{e_1} + in_{e_2}\}, \\ out_{e_1 \circ e_2} &= \min\{out_{e_2}, out_{e_1} - cost_{e_2}\}, \\ cost_{e_1 \circ e_2} &= \max\{cost_{e_1} + cost_{e_2}, in_{e_1} - out_{e_2}\}. \end{aligned}$$

If $\max\{in_{e_1}, cost_{e_1} + in_{e_2}\} > b_{max}$, then the shortcut is not feasible and $cp_{e_1 \circ e_2}$ is undefined. Note that $cost_p$ is not always the sum of particular edge costs, it is the case when any subpath recuperate more than maximum energy. Also, the profile of energy consumed on chained edges differs for each EV model.

Chargeable Limits: As suggested earlier, algorithm do the charging procedure at the charger or the destination vertex only if any charger were seen before. Now, we need to determine the amount of charged energy at the last seen charger vertex. When we arrive to such a vertex $u \in S \cup d$, where the charging by last seen charger $s \in S$ is allowed. The SoC $b_{in}(s)$ is a SoC at last seen charger vertex and $cp_{s,u}$ is consumption profile from s to u. We determine the maximum SoC β_{max} and minimum SoC β_{min} that pays of to be charged as follows:

$$\begin{aligned} \beta_{max} &= cost_{s,u} + out_{s,u} \\ \beta_{min} &= \max(in_{s,u}, \min(b_{in}(s), \beta_{max})) \end{aligned}$$

If the charging time function and price per unit of energy is the same for all chargers, then would be optimal to charge to SoC β_{min}. Nevertheless, we consider dynamic pricing on the chargers, thus we have to try all chargeable SoCs in the interval $[\beta_{min}, \beta_{max}]$. We discretize the interval to a finite list of SoCs. Let us define a list of chargeable SoCs $CL =$

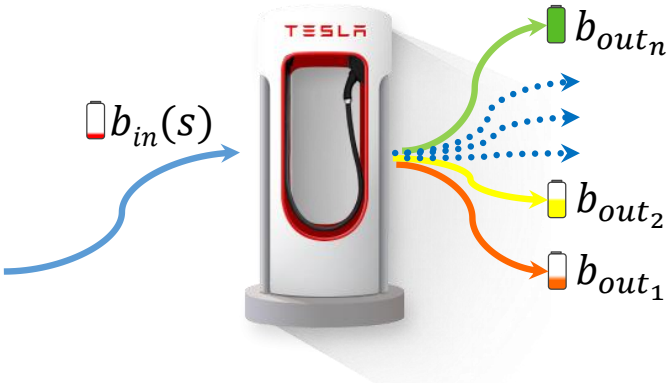


Figure 4.3: Chargeable state of charges. The $b_{in}(s) \in [b_{min}, b_{max}]$ is a SoC while arriving to the charger and $CL = (b_{out_1}, b_{out_2}, ..., b_{out_n})$ is finite list of n SoCs that are possible to charge.

$(b_{out_1} = \beta_{min}, b_{out_2}, \ldots, b_{out_n} = \beta_{max})$, where $n = |CL|$ is number of SoCs in the list and $b_{out_i} < b_{out_{i+1}}$ for $i = 1, \ldots, n-1$, see Figure 4.3.

Thus, for each successor of the actual vertex u we generate exactly n labels. By discretizing the interval, we may violate the optimality of solution, it is important to choose SoCs between β_{min} and β_{max} wisely. On the other hand, we can adapt to the EV driver by choosing the list of SoCs that he desire to charge. In real use, driver will not want to go to a charging station because of the recharging a small amount of energy.

4.3.4 Extended Label

Given the search graph G_{search}, let us have an **extended label** $l_u = (\tau, \pi, b_{ch}, cp_{s,u}, s, b_{in_s}, \pi_s)$, where

- $\tau \in \mathbb{R}_0^+$ is travel time in vertex u without charging time at the last seen charger s,
- $\pi \in \mathbb{R}_0^+$ is travel costs in vertex u without travel costs for charging at the last seen charger s,
- $b_{ch} : V_{search} \mapsto \in \mathbb{R}_0^+$ is a function that assigns the amount of charged energy to each vertex in the path,
- $s \in S$ is last seen charger,
- $cp_{s,u}$ is a consumption profile function of the path from the last seen charger s to the vertex u,
- $b_{in_s} \in \mathbb{R}$ is the arrival SoC at the last seen charger s.
- $\pi_s \in \mathbb{R}_0^+$ is the price per unit of energy at the last seen charger s

Similarly as in basic BSP algorithm, we define the **domination** rule ($\prec$) for labels. The label l_u is dominated by another label $l'_u = (\tau', \pi', cp'_{s',u}, s', b'_{in'_s}, \pi'_{s'})$ if all conditions in 4.2 or in 4.4 are fulfilled:

$$\tau \geq \tau' \wedge \pi \geq \pi' \tag{4.2a}$$

$$s = s' \wedge \pi_s = \pi'_s \tag{4.2b}$$

$$b_{in_s} \leq b'_{in'_s} \wedge cp_{s,u} \prec cp'_{s',u} \tag{4.2c}$$

To be this rule complete, we also have to define a domination for the consumption profile function. Given two consumption profile functions $cp_{u,v}, cp'_{u,v}$ where $u, v \in V_{search}$. The consumption profile function $cp_{u,v}$ is dominated by $cp'_{u,v}$ if $cp_{u,v}(b) < cp'_{u,v}(b); \forall b \in [b_m in, b_{max}]$. It is really inconvenient to iterate all battery SoCs. The dominance could be simplified using three values we defined before, which will hold the correctness. The $cp_{u,v}$ is dominated by $cp'_{u,v}$, signed as $cp_{u,v} \prec cp'_{u,v}$ when all following conditions hold:

$$in_{u,v} \leq in'_{u,v} \tag{4.3a}$$

$$cost_{u,v} \geq cost'_{u,v} \tag{4.3b}$$

$$out_{u,v} \leq out'_{u,v} \tag{4.3c}$$

The conditions in 4.2 are truly limiting because we can dominate label only if the same last seen charger was visited with the same price. We strengthen the domination such that we compare the best case of the recharged label l_u to the worst case of recharged label l'_u. If the worst case of l'_u is better than the best case of l_u, the l_u is also dominated. We recharge the l'_u to the maximum chargeable SoC at last seen charger s and the label l'_u to the minimum rechargeable SoC at last seen charger s'. When we recharge the labels l_u and l'_u in last seen charger, then it is not needed to include information about last seen charger to the label dominance rule. Let us have two triplets, $(\tau_{ac}, \pi_{ac}, b_{ac})$ and $(\tau'_{ac}, \pi'_{ac}, b'_{ac})$ where

- $\tau_{ac} \in \mathbb{R}_0^+$ is the travel time in u after recharging the label l_u to the minimum chargeable SoC in s
- $\pi_{ac} \in \mathbb{R}_0^+$ is the travel costs in u after recharging the label l_u to the minimum chargeable SoC in s
- $b_{ac} \in \mathbb{R}$ is the arrival SoC in u after recharging the label l_u to the minimum chargeable SoC in s
- $\tau'_{ac} \in \mathbb{R}_0^+$ is the travel time in u after recharging the label l'_u to the maximum chargeable SoC in s
- $\pi'_{ac} \in \mathbb{R}_0^+$ is the travel costs in u after recharging the label l'_u to the maximum chargeable SoC in s
- $b'_{ac} \in \mathbb{R}$ is the arrival SoC in u after recharging the label l'_u to the maximum chargeable SoC in s

Then, the l_u is also dominated by l'_u if all following conditions hold:

$$\tau_{ac} \geq \tau'_{ac} \tag{4.4a}$$
$$\pi_{ac} + \Phi \cdot (\tau'_{ac} - \tau_{ac}) \geq \pi'_{ac} \tag{4.4b}$$
$$b_{ac} \leq b'_{ac} \tag{4.4c}$$

In 4.4b we monetized the delayed time of label l_u and included it to the travel costs.

4.3.5 Constrained Bicriteria Shortest Path Algorithm

Finally, we have all set to provide the Constrained Bicriteria Shortest Path Algorithm (CBSP) that solve the EAP-EV and MGCPP-EV.

The search graph G_{search} and the EV request q_{ev} is given. In the initialization phase, the algorithm adds the origin label $l_o = (0, 0, \perp, \perp, \perp, \perp, \perp)$ to the pareto set L_o, pareto sets of all other vertices are empty. Also, the origin label l_o is added to the priority queue Q. In the priority queue, labels are sorted firstly by travel time, secondly by travel costs. In each iteration, the label $l_u = (\tau, \pi, b_{ch}, cp_{s,u}, s, b_{in_s}, \pi_s)$ is polled from the top of the priority queue Q. Then all edges $e = (u, v) \in E_{search}$ outgoing from u are inspected. For each successor v we generate a new label, sometimes more than one. It depends on the current label.

Case 1: If the vertex u of the current label is **not the charger** ($u \notin S$), then only one new label l'_v is generated. The criteria τ, π are extended by using the weight functions $tt', tc' \in G_{search}$. The consumption profile function $cp_{(s,u)}$ is linked with the cp_e. The function b'_{ch} will is the same as b_{ch} in l_v. Information in l_u about last seen charger s is reused in new label l'_v. The generated label $l'_v = (\tau', \pi', b'_{ch}, cp'_{s',v}, s', b'_{in_{s'}}, \pi'_{s'})$ is as follows:

$$\begin{aligned}
\tau' &= \tau + tt'(e, s, b_{in_s}, b_{in_s}), \\
\pi' &= \pi + tc'(e, s, b_{in_s}, b_{in_s}, \pi_s), \\
b'_{ch} &= b_{ch}, \\
cp'_{s',v} &= cp_{(s,u) \circ e}, \\
s' &= s, \\
b'_{in_{s'}} &= b_{in_s}, \\
\pi'_{s'} &= \pi_s.
\end{aligned}$$

Case 2: If the vertex u of the current label is **the first visited charger** on the journey ($u \in S \wedge s = \perp$), then the recharging is skipped. Thus, the only one new label l'_v is generated as in the first case. We mark current charger u as the last seen charger to the l'_v. Also, we compute the actual price per unit of energy and SoC in u. The criteria τ, π are extended by using the weight functions $tt', tc' \in G_{search}$. We start new profiling of energy consumption from the current (charger) vertex u, then the consumption profile function of l'_v is cp_e. The function b_{ch} stays the same in l'_v, because we do not charged any amount of energy yet. The

generated label $l'_v = (\tau', \pi', b'_{ch}, cp'_{s',v}, s', b'_{in_{s'}}, \pi'_{s'})$ is as follows:

$$\begin{aligned}
\tau' &= \tau + tt'(e, s, b_{in_s}, b_{in_s}),\\
\pi' &= \pi + tc'(e, s, b_{in_s}, b_{in_s}, \pi_s),\\
b'_{ch} &= b_{ch},\\
cp'_{s',v} &= cp_e,\\
s' &= u,\\
b'_{in_{s'}} &= b_{init} - cp_{s,u}(b_{init}),\\
\pi'_{s'} &= \phi(u, t_{dt}(u)),
\end{aligned}$$

where t_{dt} is the function that computes the date time when a vertex is reached, as follows:

$$t_{dt} = \begin{cases} t_{init} + \tau + ct_{s,m}(b_{out_s}) - ct_{s,m}(b_{in_s})), & \text{if } u \in S \wedge s \neq \perp \\ t_{init} + \tau, & \text{otherwise.} \end{cases}$$

Case 3: If the vertex u of the current label **is charger and another charger was already visited** on the journey ($u \in S \wedge s \neq \perp$), we retrospectively recharge to all chargeable SoCs in last seen charger. Thus, for each chargeable SoC $b_{out_s,i} \in CL$ (see Section 4.3.3) is generated a new label l'_{v_i}. The new function $b'_{ch,i}$ of l'_{v_i} is created by extending the b_{ch} by $b_{ch}(s) = b_{out_s,i} - b_{in_s}$. The criteria τ, π are extended by using the weight functions $tt', tc' \in G_{search}$. Also, we compute the actual price per unit of energy and SoC in u. We start new profiling of energy consumption from the current (charger) vertex u, then the consumption profile function of l'_{v_i} is cp_e. The generated label $l'_{v_i} = (\tau', \pi', b'_{ch}, cp'_{s',v}, s', b'_{in_{s'}}, \pi'_{s'})$ is as follows:

$$\begin{aligned}
\tau' &= \tau + tt'(e, s, b_{in_s}, b_{out_s,i}),\\
\pi' &= \pi + tc'(e, s, b_{in_s}, b_{out_s,i}, \pi_s),\\
b'_{ch} &= b_{ch} \circ b_{ch}(s) := b_{out_s,i} - b_{in_s},\\
cp'_{s',v} &= cp_e,\\
s' &= u,\\
b'_{in_{s'}} &= b_{out_s} - cp_{s,u}(b_{out_s}),\\
\pi'_{s'} &= \phi(u, t_{dt}(u)),
\end{aligned}$$

where the function t_{dt} is the same as in the case 2.

After generating new labels, we discard all labels that are not feasible by the EV model m. Then, we inspect all labels that left. If the label l'_v is not dominated by any label $l_v \in L_v$, then the label l'_v is added to L_v and Q. At the same time, each label $l_v \in L_v$ that is dominated by newly created label l'_v is removed from L_v and Q. Algorithm terminates when Q is empty. The solution of the algorithm is the Pareto set L_d of the destination vertex d.

Note that when the destination vertex d is reached, it is also needed to decide whether to recharge. If last seen charger is defined $s \neq \perp$, then we need to generate labels modified by recharging.

This algorithm is **label setting** because the functions tt' and tc' are non-negative. In this case it means, that once the vertex u is polled from Q then the actual label l_u cannot be dominated anymore, we call such a label as **settled**. If we stop the algorithm after the first label l_d of the destination vertex is polled from Q, we get only the path with the minimum travel time.

The similar algorithms were introduced in [13, 17, 16]. However, these works differs in the problem specification. We were inspired by their approaches and adapted them to our problems. The difference is mainly in the structure of label and in the label dominance.

4.4 Speed-up Techniques

Our extension of BSP solve the problem directly in the search graph with non-polynomial time in the worst case. It is not sufficient for real-time applications, where it is assumed that the result is returned in few seconds at worst. Therefore, we propose some techniques that improves the computation time with little or no loss in solution optimality.

4.4.1 Pre-processed Search Graph

The edge pre-processing is a speed-up technique that creates shortcuts between nodes, it is crucial for computing paths in large graphs. For our purpose, it is useful to precompute optimal paths between chargers. This can be done only once before running the program. In query time, it is needed to compute non-dominated paths from origin vertex to each charger (First Mile problem) and non-dominated paths from each charger to the destination vertex (Last Mile problem). Then, we may use another search graph that contains only the shortcuts to solve our problems. This technique was also used in [16].

For the precomputation of edges, we use the CBSP algorithm with several modifications. We set the charging stations as the goal vertices and disable the recharging at charger vertices. Furthermore, the travel costs criteria is set to zero in each label. During precoputation, we do not know the exact SoCs in vertices contained into the shortcut. To not loose the information about energy consumption while computing the shortcuts, we use the chaining of consumption profile function cp. Nevertheless, the energy consumption profile has to be precomputed for each EV model $m \in M_r$ contained in CERG. Moreover, between each pair of vertices is pareto of non-dominated journeys, it leads to parallel edges in the precomputed graph. It is caused by battery constraint, the slower paths with higher SoC at the destination cannot be removed to not harm the optimality. The number of solutions in pareto sets may be really big. However, some solutions in the pareto sets have approximately the same criteria. Thus, we decided to define a tuning constant $k \in \mathbb{N}$ that will limit the number of created edges from each pareto sets.

The **Pre-processed Search Graph** G'_{search} is a Search Graph (see Section 4.1) where the set of edges E'_{search} is the composition of precomputed edges between chargers E_{ch}, first mile edges E_{fm} and last mile edges E_{lm}. The set of vertices V'_{search} is the set of charger vertices S merged with an origin vertex o and a destination vertex d. The example of pre-processed search graph G_{search} is in Figure 4.4. Note that pre-processed graph does not have to be a strongly connected component, it depends on cruising range of EV.

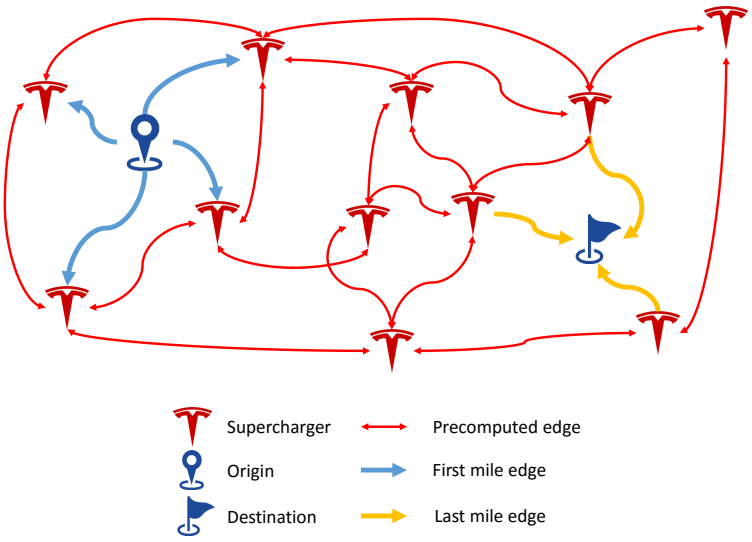


Figure 4.4: Example of the Pre-processed Search Graph. Each connection in the picture describes a list of parallel edges.

Edges Between Chargers We run the CBSP from every charger $s \in S$ in the basic Search Graph G_{search}. Since the charging on the chargers is disallowed, the EV's cruising range is limited by the initial SoC. At the time of pre-processing, it is unknown what will be the arrival SoC $b_{in}(s)$ and how much energy will be charged at the charger s. Thus, the initial SoC $b_{init} \in B$ is set to maximum SoC b_{max}. For every termination of the algorithm, we extract all Pareto sets L_v of charger vertices $v \in S - s$. Then for each Pareto set L_v we create parallel edges $e = (s, v, i); i = 1, \cdots, n$ from start charger s to destination charger v where $n \in \mathbb{R}^0$ is the number of labels in the Pareto set L_v, index i indicates the order of parallel edge. Parallel edges are ordered by the travel time and the energy consumption respectively. This leads to the generation of too many edges. Even the slowest path between chargers with the highest SoC could be the part of optimal journey. Because some edges may be really similar, the reduction of the number of generated edges is reasonable. We select no more than k edges of each pair of chargers. The outcome is set of edges between chargers E_{ch}

First and Last Mile Edges The First Mile problem is solved by running the CBSP algorithm in the basic Search Graph G_{search} from origin vertex $o \in V_r$. Charging is disallowed again, but the initial SoC b_{init} is defined by user. Set of edges E_{fm} is created from Pareto sets of charger vertices similarly as explained before. We also create edges from the Pareto set L_d of the destination vertex if it is not empty. The last mile problem is solved by running the CBSP algorithm (without charging property) on G_{search} from destination vertex $d \in V_r$, but with the difference in inspection of edges. Instead of outgoing edges, the incoming edges are inspected. Also, the consumption profile function has to be chained in reversed order. Initial SoC b_{init} is set to b_{max}. During the reversed search, the actual SoC in

label need not to hold in battery interval $[b_{min}, b_{max}]$. It does not matter because we have profile of energy consumption in a label, not the exact SoC. Thus, the algorithm can discard labels if consumption profile function is undefined. If the initial SoC is too small, the First mile does not find any path to Charger nor to the destination vertex. Thus, the feasible path does not exist. When the Last Mile computation returns the empty set of edges and there are no edge from origin vertex to destination vertex in the First Mile set of edges, the algorithm terminates and the third phase is not triggered.

4.4.2 Dominance Relaxation

Such a definition of label dominance, as in 4.3.5, may cause that pareto sets contain a labels with the almost same criteria values. Also, the pareto sets may become too large. With reference to [26, 17], we applied the dominance relaxation technique that reduce the number of labels in Pareto sets during the search. The purpose of this technique is to favor the criteria values of the labels that were inspected earlier. Let us have two labels l_u, l'_u from CBSP and a value $\epsilon \in [0, 1]$. Then we determine a ϵ-dominance ($\precsim$). A label l_u is ϵ-dominated by label l'_u ($l_u \precsim l'_u$) if the all condition in 4.5 or 4.6 are fulfilled:

$$\tau \geq \epsilon \cdot \tau' \wedge \pi \geq \epsilon \cdot \pi' \tag{4.5a}$$

$$s = s' \wedge \pi_s = \pi'_s \tag{4.5b}$$

$$b_{in,s} \leq b'_{in,s'} \wedge cp_{s,u} \prec cp'_{s',u} \tag{4.5c}$$

Similarly, we relax the rule 4.4. The l_u is also ϵ-dominated by l'_u if all following conditions hold:

$$\tau_{ac} \geq \epsilon \cdot \tau'_{ac} \tag{4.6a}$$

$$\pi_{ac} + \Phi \cdot (\tau'_{ac} - \tau_{ac}) \geq \epsilon \cdot \pi'_{ac} \tag{4.6b}$$

$$b_{ac} \leq b'_{ac} \tag{4.6c}$$

Where the triplets $(\tau_{ac}, \pi_{ac}, b_{ac})$ and $(\tau'_{ac}, \pi'_{ac}, b'_{ac})$ are the same as in Section 4.3.5.

4.4.3 Search Space Reduction

Since we do not allow large detours, we decided to reduce the search space by an ellipse as it is described in [27]. In query time, we set the focal points of the ellipse to the origin o and destination d. Then during search, only the vertices that intersects the fixed ellipse are inspected. This pruning heuristic may reduce the size of the optimal pareto set of journeys. In worst case, it is possible that solution might not be found at all. This usually occurs when the distance between the origin and the destination is short or when it is needed to drive around the water area. To improve the this technique we define the minimum length of the minor axis.

Chapter 5

Implementation

This chapter we provide the implementation details of this work. Firstly, we describe the tool that build the graphs from OpenStreetMap data. Then, we provide the pseudocode of the Constrained Bicriteria Shortest Path algorithm proposed in Chapter 4. Finally, we describe the backend and the frontend of the web application Charge Here.

5.1 Tool for Building Graphs

We have defined several types of graphs in this work, the Charger Extended Road Graph (CERG) that describes transport network for EV, the basic Search Graph that has same set of vertices and set of edges as CERG, the Pre-processed Search Graph that contains precomputed edges between chargers, First Mile edges and Last Mile edges. In this section we introduce a new type of graph that is used in our implementation.

5.1.1 Building Charger Extended Road Graph

OpenStreetMap (OSM) project provides free geographical data that contains information about the transport network. Thus, we decided to use OSM data as the input. At first, the data are downloaded in OSM format (.osm). We extract the road network using the Osmfilter[1]. In the file, the location is defined as **node** and road is defined as **way** which is the sequence of nodes. Nodes are defined by id, latitude and longitude, for the energy consumption function we need information about elevation also. The Osmosis plugin[2] uses the Shutter Radar Topography Mission (SRTM) data to assign elevation tag to each node. The OSM format (.osm) is the extension of XML format and it is possible to parse the OSM file to Java Objects using the `SAXParser` class from the `javax.xml.parsers` package. Now, we have the road network represented as the graph structure with vertices and edges in Java Object. We implemented algorithm that extracts the biggest strongly connected component from a graph. In other words, for every pair of vertices $u, v \in V$ exists a path from u to v in graph $G = (V, E, f)$. We extend such a component by charging stations data imported from CSV file. For each charging station we add the vertex with coordinates, the incoming

[1] http://wiki.openstreetmap.org/wiki/Osmfilter
[2] http://github.com/locked-fg/osmosis-srtm-plugin

edge and the outgoing edge. The result is the CERG graph. Note that information about charging stations are not sufficient in OSM data. However, we expect that information about charging stations may be imported from CSV files.

5.1.2 Building Basic Search Graph

The basic Search Graph has the same number of edges and vertices as the CERG, but contains different functions. To build the basic Search Graph we need to compute the consumption profile values for each edge in CERG. In implementation, we distinguish the search graph for solving First Mile problem and for solving Last Mile problem. Both graphs implements interface with the function `getSuccessors()` and `buildPath()`. The function `getSuccessors()` takes the current label in the argument and generates new labels. The basic Search Graph for First Mile inspects outgoing edges and the basic Search Graph for First Mile inspects incoming edges. Both graphs has disallowed recharging and returns only the labels that satisfies battery constraints. The function `buildPath()` builds the path by going through label ancestors. During the process, the vertex ID is assigned to every vertex.

5.1.3 Building Pre-processed Search Graph

We use the CBSP algorithm with disabled recharging to create First Mile edges, Last Mile edges and edges between chargers, as described in 4.4.1. From each pareto set of goal vertex with the size n we extract k journeys, the fastest journey, the journey with the minimum energy consumption and $k-2$ evenly distributed journeys in the set. If $n \leq k$ then all journeys are used. In the case, the journey with the minimum needed SoC to surpass the journey is not contained in the selection, we add it as well. We use the selected journeys to create new parallel edges between chargers. We set the maximum number k of parallel edges to 5. We build the Pre-processed Search Graph by the set of charger vertices and the set of precomputed edges. This graph implements the same interface as the basic Search Graph. Thus, the functions `getSuccessors()` and `buildPath()` are also present. In this case, the function `getSuccessors` has allowed recharging. For each parallel edge and for each possible charging limit a new label is generated. Also, only the labels that satisfies battery constraints are returned by this function.

5.1.4 Graph Serialization

It is unnecessary to build the Pre-processed Search Graph from the OSM data every time we receive the request. We prevent the precomputation from scratch by an object serialization. We can serialize the graph that extends CERG by precomputed edges between chargers. Also, all edges has precomputed consumption profile values. We call such a graph the **Double Graph** and its example is shown in Figure 5.1. The graph is visualized in CardoDB[3] and can be also accessed via public link[4]. Then in query time, the basic Search Graph is created by extracting edges from the Double Graph that are not precomputed. The First Mile problem and Last Mile problem are solved by CBSP algorithm in the basic Search

[3] http://carto.com

[4] http://fiserto2.carto.com/viz/c072f736-1c81-11e6-9f80-0e31c9be1b51/public_map

Graph. After all, the Pre-processed Search Graph is created from charger vertices, First Mile edges, Last Mile edges and precomputed edges between chargers.

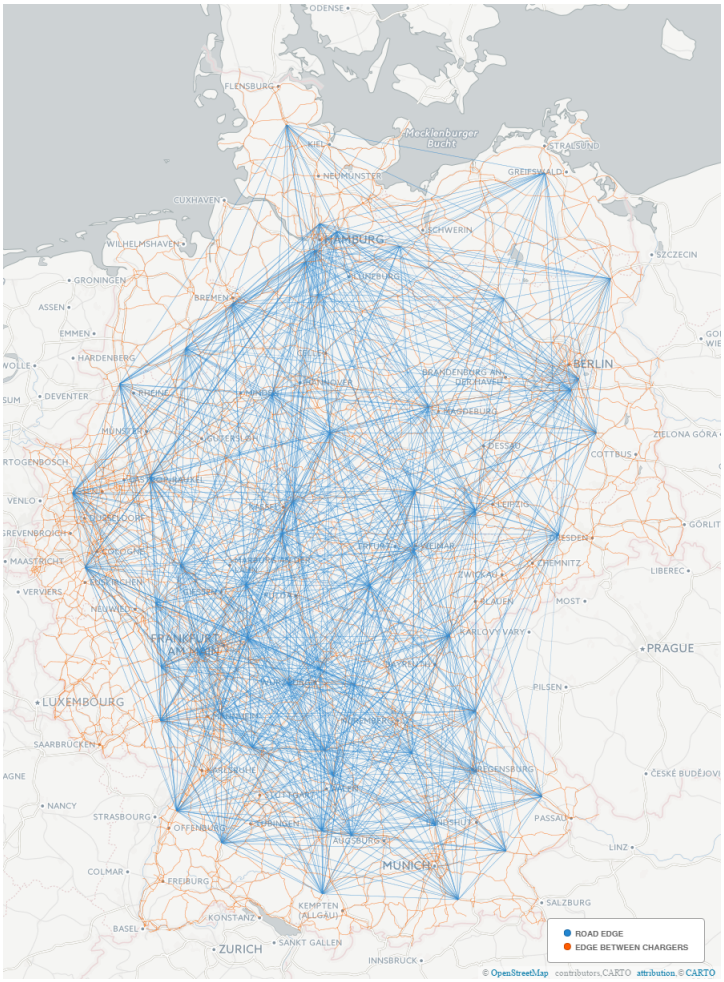


Figure 5.1: Serialized Double Graph. It is a visualization of the graph that contains two layers. First layer is the Charger Extended Road Graph, the second layer is a set of precomputed edges between chargers E_{ch}. Motorway, trunk and primary road tags were used for constructing the graph.

5.2 Routing

This part of implementation contains mainly the Constrained Shortest Path Algorithm and structure of the label proposed in Section 4.3.5. The purpose of our implementation is to have the one algorithm applicable on more search graphs with the same interface. The search graph generates new feasible labels by the function `getSuccessors` and the algorithm manages the pareto sets. When the graph returns new labels, every label is checked against the labels in existing pareto set of the same vertex. All pareto sets are stored in

bags which is the instance of `HashMap` class. The key of the hashmap is vertex ID and the value is the `HashSet` of labels. In the bag the settled labels, as well as unsettled. Settled label is such a label that was polled from queue and cannot be dominated anymore. We store the settled labels of goal vertices in another hashmap. In this case, the lists of paths are mapped on the goal vertex IDs. The list of paths is an instance of the `ArrayList`. For better understanding the implementation, we introduced simplified pseudo-code.

Function: cbspAlg()

Input: search graph $G = (V, E, tt', tc', dt, cp, \phi, \varsigma)$, start vertex s, goal vertex g, initial SoC $initB$, departure time $initT$, costs per hour phi
Output: List of journeys journeyList retrieved from goal settled labels
Data: priority queue Q, hash map bags containing label set mapped on each vertex in V

```
begin
    foreach v ∈ G.NODES do
        bags.put(v, ∅);
    end
    l ← createStartLabel ();
    bags.get(s).add(l);
    Q.add(l);
    while not Q.isEmpty() do
        l ← Q.poll();
        l.settle ();
        if l.VERTEX = g then
            goalLabelSet.add(l);
        end
        succLabels ← G.getSuccessorLabels (l);
        foreach l' ∈ succLabels do
            labelSet ← bags.get(l'.VERTEX);
            if not (l' ≺ labelSet) then
                foreach l'' ∈ labelSet do
                    if l'' ≺ l' then
                        labelSet.remove(l'');
                        Q.remove(l'');
                    end
                end
                labelSet.add(l');
                Q.add(l');
            end
        end
    end
    journeyList ← retrieveJourneys (goalLabelSet, G);
    return journeyList;
end
```

The priority queue is instance of the class **PriorityQueue**. G.getSuccessors() returns for each parallel edge and for each possible charging limit new label. only labels that satisfies battery constraints. This function is different for each search phase (FM, LM, SP).

5.3 Web Application

The web application Charge Here is divided to the server-side (Backend) and the client-side (Frontend). Both parts are deployed on the virtual server (see Figure 5.1) with Apache Tomcat container. The application is freely accessible online[5].

CPU	Intel Xeon E5-26xx 2.0 GHz (Sandy Bridge)
CPU cores	4
RAM	16 GB
Operating System	Linux 3.16.0-4-amd64 (Debian)
Tomcat Version	Apache Tomcat 8.0.32
JVM Version	1.8.0_72-b15

Table 5.1: Description of the virtual server.

5.3.1 Backend

On the server side is the Java servlet that includes the CBSP algorithm to compute journeys. The application is designed in Representational State Transfer (REST) software architectural style. The journey planner for EV supports only the territory of **Germany**. The charging station network contains only Tesla's Superchargers.

Project Structure: The Apache Maven project consists of the following parts:

- **ev-structures** - contains basic structures that are needed for building the graph of transport network, as well as for the routing for EVs, e.g. `EVNode`, `EVEdge`, `Charger`
- **zone-builder** - converts the input data to a Java Objects, builds the serializable graph from Section 5.1.4
- **routing** - contains the implementation of the CBSP algorithm described in Section 5.2
- **api** - contains the classes that define the REST service by using the JAX-RS annotations and classes that describes JSON objects of the request and the response.

REST API: The application accepts request via HTTP POST method in JSON date type. The receiving JSON model contains the origin coordinates and the destination coordinates, initial SoC of the EV model in percents and the time monetization parameter in cents per hour (see Listing 5.1).

[5] http://charge-here.eu

```
{
 "client": <string>,
 "initSoCInPerc": <int>,
 "centsPerHour": <int>,
 "origin": {
  "latE6": <int>,
  "lonE6": <int>
 },
 "destination": {
  "latE6": <int>,
  "lonE6": <int>
 }
}
```

Listing 5.1: The JSON model of the request

An elevation of the origin and the destination is not present in the request, because it is assigned by the journey planner, as well as the EV model. As the response, the servlet sends back another JSON with all found journeys. In this case, we call the journey, the plan because the structure is quite different. Every plan is described as a list of plan segments divided by chargers. Each segment contains an arbitrary number of consecutive steps. The step is described by the pair of coordinates with additional information (see Listing 5.2).

```
{
 "coordinates": {
  "latE6": <int>,
  "lonE6": <int>
 },
 "distanceToNextStep": <int>,
 "leavingStateOfCharge": <int>,
 "travelTimeToNextStep": <int>,
}
```

Listing 5.2: The JSON model of one step

The whole process of the backend part of the application is described in pseudo-code below. The Double Graph from Section 5.1 is deserialized once in the initialization phase of the servlet.

Program: Journey Planner for EV

Input: JSON object *request* containing origin o, destination d, initial SoC $initB$, costs per hour *phi*

Output: JSON object *response* containing the list of plans

Data: deserialized graph $dGraph$, EV model $evModel$,

begin

- $initT \leftarrow$ `getCurrentDateTime` ();
- start $\leftarrow$ `findNearestNode` (o, $dGraph$);
- goal $\leftarrow$ `findNearestNode` (d, $dGraph$);
- edgesFM $\leftarrow$ `computeFirstMileEdges` (start, $dGraph$, $evModel$, $initB$);
- edgesLM $\leftarrow$ `computeLastMileEdges` (goal, $dGraph$, $evModel$);
- searchGraph $\leftarrow$ `buildSearchGraph` (edgesFM, edgesLM, $dGraph$, $evModel$);
- journeyList $\leftarrow$ `cbspAlg` (searchGraph, start, goal, $initB$, $initT$, phi);
- $response \leftarrow$ `buildResponse` (journeyList, $dGraph$);
- **return** *response*;

end

5.3.2 Frontend

The client-side is coded in HTML/CSS and JavaScript (jQuery[6], in particular) languages. Some features from Bootstrap[7] framework are used, e.g., Bootstrap Grid to simplify positioning of elements. Drawing the markers and journeys is handled by Mapbox extension[8] of Leaflet library[9]. Map tiles, also provided by Mapbox, are based on OpenStreetMap data.

Design: The design of the application is straight forward. The most of the page is covered with the map. The map has an additional layer with superchargers, the availability of supercharger is shown on click. In the top left corner is the panel for the addresses (or GPS coordinates) of the origin and the destination of the requesting journey. Both input boxes have enabled the autocomplete feature that is connected with the Mapbox Geocoding API. In the bottom of the search panel are two sliders. The first slider indicates the battery state of charge of EV at the start in percents. The second slider sets the money that would the driver spent per an hour on a journey in euros, we call it costs per hour (CpH). Greater value of CpH prioritizes the fast journeys. Another panel will appear after any journey is planned. This panel contains the list of plans ordered by travel time. Each plan is described by four values, the travel time, the length, the energy consumption and the travel costs. The travel time criterion contains time spent while driving as well as the time spent by recharging the vehicle. The travel costs contain costs for the recharged energy and travel time monetized by the value (CpH) which was chosen by the driver before. The selected journey has shown the battery profile graph also.

Usage: By using the sliders, the user selects the initial battery state of charge of his EV model and his opportunity costs of an hour spent on the journey. Then, he determines the position of the origin marker and the destination marker by clicking in the map or filling the coordinates in the search input fields. Another opportunity is to use the autocomplete feature by typing a part of the address. Note that there is a need to press Enter for placing the markers into the map correctly. The frontend sends immediately the JSON request (see Listing 5.1) via the HTTP POST method. If the JSON response is returned successfully, the application shows new panel as the list of the recommended plans with the description. At the same time, all journeys are drawn into the map as a polyline. Along the selected journey is also shown the markers that describes the superchargers where the EV will be recharged. The additional info about recharging is shown after clicking on a marker. The result is time-dependent because the prices per unit of energy are influenced by departure time. The plans would be disappeared by clicking the 'X' button in the bottom right corner of the main panel.

[6] http://jquery.com
[7] http://getbootstrap.com/
[8] http://www.mapbox.com/mapbox.js/api/v3.0.1/
[9] http://leafletjs.com/reference

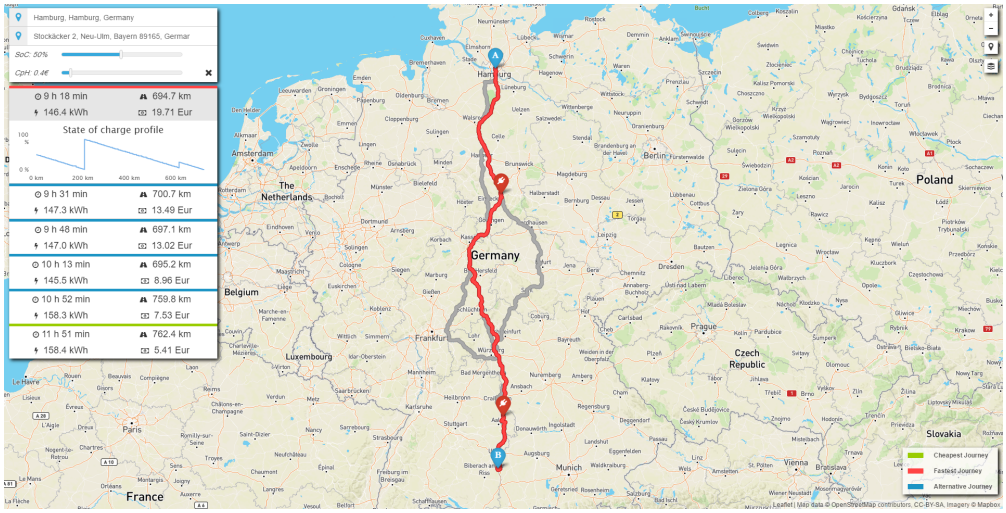


Figure 5.2: Web application Charge Here

Chapter 6

Evaluation

In this chapter we evaluate our implementation of the CBSP algorithm introduced in Chapter 4. We run several experiments with different input parameters, that affect the computation time of the algorithm.

The whole program, that solves our problems from Chapter 3, is divided into three phases. The first phase is the First Mile (FM) computation, where the CBSP algorithm without recharging is used to compute pareto sets of paths from origin vertex to all feasible chargers or to the destination vertex, if it is also feasible. The second phase is the Last Mile (LM) computation, where the CBSP algorithm without recharging is used to compute pareto sets of paths from the destination vertex to all feasible chargers by expanding reversed edges. Both phases run in the basic Search Graph G_{search}. In the third phase (TP), the CBSP (with recharging) computes the optimal journeys from the origin vertex to the destination vertex in the Pre-processed Search Graph G'_{search}.

6.1 Fundamental Settings and Input Data

In this section, we introduce the input graph and determine the origin-destination pairs for our experiments. We also set the general parameters for the CBPS algorithm.

6.1.1 Input graph

We decided to run our experiments on the model of Germany transport network. The OSM data of Germany were freely downloaded from Geofabrik website[1]. We extracted the road data of motorways, trunks, primary roads and secondary roads. Elevation data are taken from NASA's Shuttle Radar Topography Mission (SRTM). The model is extended by data[2] of Tesla Superchargers, price per unit of energy ϕ is generated by random for each supercharger and each hour from range [0,50] cents per kWh. According to the official Tesla Motors website[3], the charging function $ct_{s,m}$ for a EV model $m \in M_r$ at superchargers $s \in S$ is defined by the set of supporting vectors $((b_{min}, 0), (0.8b_{max}, 2400), (b_{max}, 4500))$. The unit

[1] http://download.geofabrik.de/europe/germany.html
[2] http://supercharge.info/
[3] http://www.tesla.com/supercharger

of energy is the watt-hour (Wh) and unit of time is the second (s). The only supported EV model is Tesla Model S with 85kWh battery pack. Thus, the battery maximum state of charge SoC b_{max} is **85,000 Wh**. We do not take the risk of getting stranded during the path, thus the minimum SoC b_{min} is set to **500 Wh**. The constants κ, λ, α in the consumption function g are set as follows:

$$\kappa = 0.2, \lambda = 2, \alpha = 1.5.$$

The set of supported road categories R_r and their average speeds are shown in Table 6.1.

Road category	**Average speed [km/h]**
Motorway	100
Motorway link	40
Trunk	70
Trunk link	40
Primary	60
Primary link	40
Secondary	60
Secondary link	40

Table 6.1: Average speeds for road categories.

The outcome graph structure corresponds to the Double Graph from Section 5.1.4. The Double Graph has **245,211** vertices and **497,928** edges. The set of edges includes 488,491 road edges, which are used to create the basic Search Graph G_{search} and 9,437 pre-processed parallel edges between chargers is 9,437. The sum of single edges between chargers is 1,943 and the number of superchargers in the graph is **56**.

6.1.2 Requests

For the experiments, we have chosen 100 origin-destination pairs (OD-pairs) described in Table 6.2. All requests consider that Tesla Model S (85 kWk) is the EV model m that will be driven on the requested journeys. We also fixed the departure time t_{init} at **10 a.m.** to hold the result consistent. Chosen values of the initial SoC b_{init} and monetization constant Φ are determined in the beginning of each experiment.

6.1.3 Algorithm settings

All speed-up techniques are enabled, in general. The dominance relaxation constant ϵ is set to **0.95**. The search space is reduced by the ellipse where a length of minor axis is the direct distance between the origin and the destination vertices doubled. The maximum number of parallel edges between two points k is set to **5**. The charging policy is such that the set of chargeable limits CL contains the minimum chargeable SoC β_{min} and the maximum chargeable Soc β_{max} as well as every **twentieth** percentage of battery capacity that is in the interval $[\beta_{min}, \beta_{max}]$.

Request	Origin	Destination
1	Berlin	Munich
2	Frankfurt am Main	Munich
3	Köln	Berlin
4	Köln	Hamburg
5	Frankfurt am Main	Stuttgart
6-100	Random OD-pairs from the bounding box 6.3	

Table 6.2: Origin-Destination pairs (OD-pairs), 5 OD-pairs represents real requests to find journeys between selected German cities and 95 OD-pairs are generated randomly from the bounding box 6.3 under one contidion that direct distance between origin and destination is greater than 20 km. Seed value for the random generator is 53684.

Position	Latitude	Longitude
North East	53.058141	14.542051
South West	47.270211	6.866241

Table 6.3: The bounding box of Germany.

6.2 Experiments

All experiments were run by a remote computing machine provided by Metacentrum[4]. Technical specification of the virtual machine is in Table 6.4. In Table 6.5 we introduce a few tags that are used in this section.

Cluster	alfrid-cluster.meta.zcu.cz
CPU	Intel Xeon E5-2650v2 2.60GHz
CPU cores	1
RAM	32GB
Operating System	Linux (Debian)
JVM Version	1.8.0

Table 6.4: Technical specification of the virtual machine where the experiments were run.

[4]http://metacentrum.cz

Tag	Description
b_{init}	Initial battery state of charge
FM	Computation time of the CBSP algorithm used to solve First Mile problem.
LM	Computation time of the CBSP algorithm used to solve Last Mile problem.
TP	Computation time of the CBSP algorithm used in third phase of the program.
BP	Computation time of non-search parts of the program, which includes building of the response, building of the graphs.
T	Total computation time of the program including FM, LM, TP, BP
N	The number of journeys found by the algorithm
C	Criteria of the journey, a tuple that includes travel time and travel costs of the journey respectively

Table 6.5: Tags used throuhout this chapter.

6.2.1 Average Computation Time

At first, we measured the average computation time of our program. The initial SoC and costs per hour (CpH) are randomly set for each request. Interval for initial SoC is $[34000, 85000]$ (Wh) and interval for CpH parameter Φ is $[0, 100]$ (cents). Other settings are used as defined in Section 6.1. We queried 100 requests with OD-pairs from Table 6.2, each request five times.

The result is that the average total computation time (T) of the program is **622 ms**. The proportion of individual algorithm phases on the computation time is in Figure 6.1.

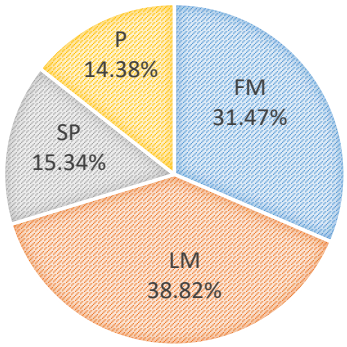


Figure 6.1: The proportion of individual algorithm phases on the average computation time.

The CBSP algorithm in third phase run for 96 ms on average, it is more than we expected. The number of edges in G_{search} which is used for FM and LM computation is much greater than the number of edges in the Pre-processed Search Graph G'_{search}. However, the G'_{search} produces too many successors in each iteration. It is caused by presence of parallel edges and because of recharging. Unfortunately, not enough successors can be dominated, it leads

to large pareto sets and it substantially slow down the algorithm. Thus, the form of the label dominance has crucial effect on the computation time of the third phase.

First Mile computation is faster on average than Last Mile due to the initial SoC b_{init} limitation. On the other hand, Last Mile computation is limited by the maximum battery capacity b_{max} which is given by the EV model. The average computation times of FM and LM are nearly similar in the case, the initial SoC is set to battery maximum capacity b_{max}.

Solving the First Mile and Last Mile problems together takes 70.29 % of total computation time. Both phases could be removed at the expense of memory complexity. Similarly, as the edges between chargers were pre-processed, the pareto set of feasible paths for each vertex in the CERG could be pre-processed. Exactly because of high demands on the memory, we did not choose this approach.

6.2.2 Initial State of Charge

In this experiment, we observe the dependency of computation time on the initial SoC of the EV model. For each initial SoC b_{init}, 100 OD-pairs from Table 6.2 were tested. The monetization constant Φ is set to 0 cents per hour. It means that the total travel costs represents only the money spent for the energy recharged at the chargers. We run the evaluation procedure five times, results are shown in Table 6.6.

b_{init}[%]	**FM [ms]**	**LM [ms]**	**TP [ms]**	**T [ms]**	**N**	$\mathbf{C_{J^*}}$ **[min, €]**	$\mathbf{C_{K^*}}$ **[min, €]**
10	6	349	325	803	5	(390, 20.70)	(520, 5.78)
20	21	301	314	748	4	(382, 18.04)	(511, 4.96)
30	48	307	281	743	4	(359, 15.88)	(486, 3.08)
40	86	296	230	735	4	(351, 13.17)	(482, 2.00)
50	134	304	201	749	4	(341, 10.95)	(468, 1.52)
60	176	286	324	897	4	(335, 9.36)	(448, 1.10)
70	209	270	134	714	3	(330, 8.35)	(433, 0.89)
80	282	301	100	794	3	(326, 6.06)	(409, 0.68)
90	324	307	73	807	2	(319, 5.20)	(390, 0.42)
100	336	300	51	802	2	(316, 3.71)	(373, 0.27)

Table 6.6: Average computation time while increasing the initial SoC. The first pair of criteria C_{J^*} describes optimal journey of EAP-EV and the second pair of criteria C_{K^*} describes the optimal journey of MGCPP-EV.

According to results in Table 6.6, the average computation time of First Mile problem increases with higher initial SoC. On the other hand, the average computation time of Last Mile is not influenced. For example, if a driver starts with 80% charged battery, the cruising range of EV is smaller than with fully charged battery. Also, if the initial SoC is too small, the destination may not be feasible.

For the Last Mile computation, starting at destination vertex, the initial SoC is inapplicable. In this phase, we do not know SoC at last seen charger or which charger will be used.

All we know is that the SoC at last seen charger has to be in battery range. During the search edges are chained in reverse order by inspecting incoming edges. We discard paths that are not feasible with fully charged battery. It causes the cruising range to be always identical for requests with the same destination. In other words, the Last Mile computation time is not dependent on initial SoC at all.

Even the third phase is influenced by initial SoC. When the initial SoC is small, then probably at least one recharging is required. Every inspected vertex, where recharging is allowed, generates more labels. Also, more frequent recharging leads to greater travel time and travel costs.

6.2.3 Dominance Relaxation

In Section 4.4, we introduced the constant ϵ which speeds up the CBSP algorithm by relaxing dominance of labels. Earlier visited labels are prioritized against newly visited labels by correcting criteria values in dominance conditions. It is used in each phase of the algorithm. In this experiment, we test how setting of the ϵ parameter influences the computation time and the solution optimality. We set the initial SoC b_{init} to 50 % of the maximum battery capacity and the monetization constant Φ to 0 cents per hour. It means that the total travel costs represents only the money spent for the energy recharged at the chargers. For each ϵ, we run 100 OD-pairs from Table 6.2. We run the evaluation procedure five times, results are shown in Table 6.7.

ϵ	**T [ms]**	**N**	$\mathbf{C_{J^*}}$ **[min, €]**	$\mathbf{C_{K^*}}$ **[min, €]**
0.80	345	1	(364, 5.18)	(455, 1.93)
0.82	331	2	(362, 5.67)	(458, 1.89)
0.84	342	2	(359, 6.61)	(458, 1.88)
0.86	359	2	(353, 7.32)	(463, 1.80)
0.88	478	2	(351, 8.49)	(463, 1.75)
0.90	413	3	(345, 9.97)	(466, 1.73)
0.92	471	3	(343, 10.49)	(466, 1.59)
0.94	587	4	(342, 10.08)	(467, 1.57)
0.96	794	4	(340, 11.61)	(472, 1.52)
0.98	1409	6	(338, 11.26)	(478, 1.42)
1.00	28174	21	(336, 11.68)	(496, 1.31)

Table 6.7: Average computation time while increasing the ϵ parameter. The first pair of criteria C_{J^*} describes optimal journey of EAP-EV and the second pair of criteria C_{K^*} describes the optimal journey of MGCPP-EV.

This measurement shows that the optimal search without dominance relaxation ($\epsilon = 1$) solves the problem in 28,174 ms on average. Although, the optimal journey $K*$ is about 21 cents more expensive than the journey if the algorithm with $\epsilon = 0.96$ is used, but the computation time is approximately 35 times smaller. In other words, the algorithm expands

a lot of labels that does not lead to optimal solution. The influence of ϵ on the minimal travel time is also shown in Figure 6.2.

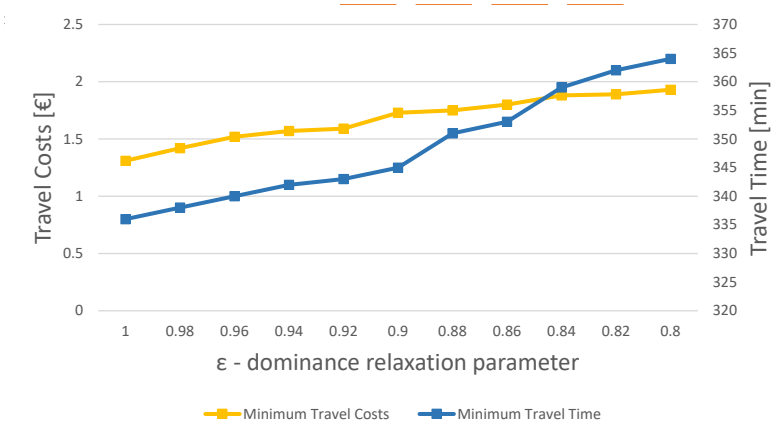


Figure 6.2: Comparison of the travel time and travel price while the dominance relaxation constant ϵ is being increased.

Note that if this technique is applied in First Mile and Last Mile phase, we risk pruning the path to a charger. It means that if the ϵ is too small, the First Mile computation may not find any path. Then the program terminates with no optimal journeys, which is incorrect. The Last Mile is affected in a similar way. In the third phase, charging is available on each vertex except origin and destination. Thus the finding of at least one journey is guaranteed if G'_{search} is strongly connected component.

6.2.4 Speed-up Comparison

In this experiment, we set ϵ to the fixed value (0.95) and compare it with the speed-up technique from Section 4.4.3. We selected a few combinations of algorithm by turning on or off the techniques. We observe how the computation time and solution optimality is affected. At the origin vertex, the battery SoC b_{min} is set to 50 % of battery capacity. The monetization constant Φ is set to 0 cents per hour, to obtain as much as possible solutions. For each setting ϵ, we used the first OD-pair from Table 6.2. We run the evaluation procedure ten times, results are shown in Table 6.7.

According to the results, our program without any speed-up technique computes the optimal journeys from Berlin to Munich in 799,640 ms on average. When the dominance relaxation ($\epsilon = 0.95$) is applied, the computation time is reduced to 2,107 ms. Nevertheless, 43 journeys from the pareto set were pruned. On the other hand, if only the technique with ellipse is applied, the computation time is 386,118 ms, but we obtained full pareto set of journeys. By applying both speed-up techniques, we get computation time under one second.

D_ϵ	**E**	**FM [ms]**	**LM [ms]**	**TP [ms]**	**T [ms]**	**N**	$\mathbf{C_{J^*}}$ **[min, €]**	$\mathbf{C_{K^*}}$ **[min, €]**
		26697	697638	75149	799640	51	(419, 11.60)	(707, 3.34)
✓		78	545	1378	2107	8	(430, 12.98)	(753, 4.18)
	✓	24141	307231	54590	386118	51	(419, 11.60)	(707, 3.34)
✓	✓	78	397	361	945	8	(430, 12.98)	(753, 4.18)

Table 6.8: Average computation time while different settings of speed-up techniques is used. Compared techniques are dominance relaxation D_ϵ and search space reduction technique E which allows visiting only nodes in the ellipse. The first pair of criteria C_{J^*} describes optimal journey of EAP-EV and the second pair of criteria C_{K^*} describes the optimal journey of MGCPP-EV.

Chapter 7

Conclusion

We have formalized a journey planning problems for electric vehicles (EVs) that consider recharging and dynamic pricing. The first problem is about finding the journey that minimizes a travel time including the time spend by recharging. The second problem asks for the journey with the minimum travel costs. We avoid the inappropriate detours, the second journey minimizes money spent for the energy recharged at the charging stations and takes driver's opportunity cost of travel time into account. We proposed the graph based algorithm, which extends the Bicriteria shortest path algorithm and solves both problems at once. The outcome of the algorithm contains also alternative paths that are the trade-off between both optimal solutions. We applied several speed up techniques as edge precomputation or dominance relaxation to achieve satisfactory computation time. We tested our Constrained Bicriteria Shortest Path algorithm on Germany transport network represented as a graph. We have chosen the network of Tesla Superchargers for recharging the battery of EV. Average computation time of the algorithm for random requests is about a second. The algorithm is applicable on larger areas with the same sparseness of charging stations with minimal change of computation time. We also developed the web application Charge Here[1] which handles on-demand journey requests. The application includes the same algorithm and shows the solution onto a map.

This work contributed to the understanding of the problem, which includes the Dynamic Pricing strategy into route planning for EV. It will be further elaborated by the Artificial Intelligence Center in the ELECTRIFIC project.

7.1 Future Work

First of all, we want to investigate more sophisticated speed up techniques that could enhance the computation time of the search algorithm. Afterwards, we want to do several experiments on the quality of returned journeys.

Another goal is to provide a model of Dynamic pricing based on the actual traffic to demonstrate our idea of balancing the electrical grid. Also, we want to improve the structure of transport network model by using more information about each road segment. We will

[1] http://charge-here.eu

add more factors to the consumption function, to determine the energy consumption of the EV more accurately. In our model, the speed nor the attributes of EV does not have an impact on energy consumption. Furthermore, breaking causes the battery recuperation. We would like to take acceleration and deceleration of EV into account as well.

Proposed transport network model assigns average speed to the edge by road type. We would like to adapt the speed on each road segment to regular traffic, as well as to the driving uphill, downhill or driving into turns.

Finally, we want to enrich the data of transport network in web application Charge Here. The graph covers Germany road network of motorways, trunks, primary roads and secondary roads. Our goal is to cover whole Europe with all road categories where the EVs have allowed access. We will also consider whether to add more types of charging stations.

Bibliography

[1] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, pages 269–271, 1959.

[2] A. Artmeier, J. Haselmayr, M. Leucker, M. Sachenbacher, et al. The optimal routing problem in the context of battery-powered electric vehicles. In *Workshop: CROCS at CPAIOR-10, Second International Workshop on Constraint Reasoning and Optimization for Computational Sustainability, Bologna, Italy*, 2010.

[3] H. C. Joksch. The shortest route problem with constraints. *Journal of Mathematical Analysis and Applications*, 14(2):191–197, 1966.

[4] M. Sachenbacher, M. Leucker, A. Artmeier, and J. Haselmayr. Efficient energy-optimal routing for electric vehicles. *AAAI Conference on Artificial Intelligence*, 2011.

[5] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, pages 1–13, 1977.

[6] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, pages 100–107, 1968.

[7] J. Eisner, S. Funke, and S. Storandt. Optimal route planning for electric vehicles in large networks. In *AAAI*, 2011.

[8] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. *In Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08), volume 5038 of Lecture Notes in Computer Science*, pages 319–333, 2008.

[9] M. Baum, J. Dibbelt, T. Pajor, and D. Wagner. Energy-optimal routes for electric vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, pages 54–63, New York, NY, USA, 2013.

[10] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. *Customizable Route Planning*, pages 376–387. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[11] S. Storandt and S. Funke. Cruising with a battery-powered vehicle and not getting stranded. In *AAAI*, volume 3, page 46, 2012.

[12] J. Sauer. Energy-optimal routes for electric vehicles with charging stops. Bachelor thesis, Karlsruhe Institute of Technology, 2015.

[13] T. Zündorf. Electric vehicle routing with realistic charging models. Master's thesis, Karlsruhe Institute of Technology, 2014.

[14] M. Baum, J. Dibbelt, A. Gemsa, D. Wagner, and T. 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*, GIS '15, pages 44:1–44:10, New York, NY, USA, 2015.

[15] L. Mandow and J. L. P. De La Cruz. Multiobjective a* search with consistent heuristics. *J. ACM*, 57(5):27:1–27:25, 2008.

[16] S. 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*, IWCTS '12, pages 20–25, New York, NY, USA, 2012.

[17] M. Baum, J. Dibbelt, L. Hübschle-Schneider, T. Pajor, and D. Wagner. Speed-Consumption Tradeoff for Electric Vehicle Route Planning. In Stefan Funke and Matúš Mihalák, editors, *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 42 of *OpenAccess Series in Informatics (OASIcs)*, pages 138–151, Dagstuhl, Germany, 2014.

[18] T. Wang, Ch. G. Cassandras, and S. Pourazarm. Energy-aware vehicle routing in networks with charging nodes. *{IFAC} Proceedings Volumes*, 47(3):9611 – 9616, 2014.

[19] S. Pourazarm, Ch. G. Cassandras, and A. Malikopoulos. Energy-aware vehicle routing in networks with charging nodes: A dynamic programming approach. *{IFAC} Proceedings Volumes*, 47(3):9611 – 9616, 2014.

[20] E. S. Rigas, S. D. Ramchurn, and N. Bassiliades. Managing electric vehicles in the smart grid using artificial intelligence: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1619–1635, 2015.

[21] B. Franklin. *Advice to a Young Tradesman, Written by an Old One.* 1748.

[22] R. W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, 1962.

[23] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962.

[24] P. Hansen. *Bicriterion Path Problems*, pages 109–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 1980.

[25] E. Q. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, pages 236–245, 1984.

[26] L. S. Batista, F. Campelo, F. G. Guimarães, and J. A. Ramírez. A comparison of dominance criteria in many-objective optimization problems. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2359–2366, 2011.

[27] D. Wagner, T. Willhalm, and Ch. Zaroliagis. Geometric containers for efficient shortest-path computation. *J. Exp. Algorithmics*, 10, 2005.

Appendix A

CD content

The CD contains three directories:

- **Thesis** - thesis in PDF file
- **Charge-here-web** - source code of the Charge-Here Frontend
- **Charge-here-backend** - source code of the Charge-Here Backend