UNIVERSITÄT ZU LÜBECK
INSTITUTE OF SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Stochastic Models and Acceleration Techniques for Green Routing in Car Navigation Systems

Master Thesis

submitted by	René Schönfelder
course of study	Computer Science
matriculation number	[. . .]
address	[. . .] [. . .]
e-mail	schoenfr@isp.uni-luebeck.de
submitted on	December 13, 2011
advisor	Prof. Dr. Martin Leucker
second advisor	Prof. Dr. Karsten Keller

Abstract

Drivers of electric vehicles are interested in energy-optimal routes not only to reduce anthropogenic carbon dioxide but also to extend the limited reach. This thesis describes different shortest path models for energy-optimal routing in order to extend these models with stochastic aspects as well as extending existing algorithms to handle these models.

A new perspective on the simple shortest path problem will be presented. Using an arbitrary set of states, for example the battery charge of an electric vehicle, sufficient properties of edge weights, namely monotonicity and extensivity, are discussed and applied, such that efficient algorithms can be developed. The main utility to extend existing algorithms is an abstract datatype describing partial preorder queues. The leading example of such algorithms are contraction hierarchies. Basic implementations yield correct results for the described models, but various algorithms and approximation techniques still need to be tested. A thorough implementation for the use in the GreenNav system is planned.

Declaration

I hereby declare that I produced this thesis without external assistance, and that no other than the listed references have been used as sources of information.

Lübeck, December 13, 2011

Acknowledgements

I want to thank Professor Dr. Martin Leucker, who gave me the opportunity to write this thesis and to explore the theory behind path finding. Besides giving me professional advices, our discussions always gave birth to new ideas and different perspectives.
Many thanks also go to Benjamin Meyer, Normann Decker, Christian Pohlmann, Susan Mielke, Annette Stümpel and Daniel Thoma who gave me advice concerning various problems both in discussions about the theory as well as in programming details. I also want to thank Chika Ogbonna, Frauke Boßelmann and Danilo Tellinghusen.
Furthermore, most thanks go to my girlfriend Mareike Boßelmann as well as my family for generously supporting and encouraging me not only to write this thesis but to successfully finish my degree.

Contents

Notation

- Natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ will be denoted by $\mathbb{Z}_{\geq 0}$, positive natural numbers by $\mathbb{Z}_{\geq 1} = \{1, 2, 3, \ldots\}$. $\mathbb{R}$ are the real numbers, $\mathbb{R}_{\geq 0}$ are non-negative reals and $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ is the extended real line.

- Intervals are denoted by $[a, b] = \left\{x \in \overline{\mathbb{R}} \mid a \leq x \leq b\right\}$. If $a > b$, then $[a, b] = \emptyset$.

- If A is a set, then its cardinality is $|A|$, its powerset is $\mathcal{P}(A)$ and its cartesian power is denoted by A^k.

- The functional composition of $f : X \to Y$ and $g : Y \to Z$ is $f \circ g : X \to Z$ with $(f \circ g)(x) = g(f(x))$ for all $x \in X$. Notice that this definition may not be the commonly used one, but it is the appropriate and natural one to use throughout this thesis.

- $G = (V, E)$ is a graph of vertices $V = V(G)$ and edges $E = E(G)$. Usually, $n = |V|$ and $m = |E|$. If G is directed, then $E \subseteq \{(x, y) \in V^2 \mid x \neq y\}$, otherwise G is undirected and $E \subseteq \{e \subseteq V \mid |e| = 2\}$. We say, vertex $v \in G$ if and only if $v \in V(G)$ and edge $e \in G$ if and only if $e \in E(G)$ (V and E must be disjoint).

- A vertex x is said to be connected to y in G, if $(x, y) \in E(G)$ in the directed case or otherwise $\{x, y\} \in E(G)$ in the undirected case.

- Let $f : E \to X$ be a function defined on edges, then $f(v_1, v_2)$ denotes either $f(\{v_1, v_2\})$ for undirected graphs or $f((v_1, v_2))$ for directed graphs.

- Graph $G' = (V', E')$ is a subgraph of G, denoted by $G' \subseteq G$, if and only if $V' \subseteq V$ and $E' \subseteq E$.

- A walk γ in G of length $k > 0$ from v_0 to v_k is a sequence of vertices $(v_0, \dots, v_k)$, such that there is an edge connecting v_{i-1} and v_i in G for all $i = 1, \dots, k$. γ^i is the prefix of γ having length i, i.e. $(v_0, \dots, v_i)$ $(0 \leq i \leq k)$. Usually Γ^G_{xy} denotes all walks from x to y in G and Γ^G denotes all walks in graph G. We omit G, if it is clear by context. Notice, that edges are walks of length 1.

- A path π in G is a walk on distinct vertices. Since π always induces a subgraph of G we say $\pi \subseteq G$ and $e \in \pi$, if and only if the induced subgraph contains edge e. Furthermore, $\Pi^G_{xy} \subseteq \Gamma^G_{x,y}$ denotes all paths from x to y in G and Π^G denotes all paths in graph G. We omit G, if it is clear by context. Sometimes paths are called simple paths in contrast to walks.Notice, that Π_{xy} is always finite in finite graphs.

- Given a probability space $(\Omega, \mathcal{F}, P)$ we say $P(\text{property})$ to denote $P(A)$, where $A \in \mathcal{F}$ contains all the elements $\omega \in \Omega$ fulfilling the given property.

- The expected value of a random variable X is denoted by $\mathbb{E}(X)$, not to be confused with the set of edges E.

- The argument of the maximum (minimum defined analogously) of a function $f : X \to \mathbb{R}$ is defined to be the set of elements $x \in X$ with $f(x) \geq f(y)$ for all $y \in X$, i.e.

  $$\arg\max_{x \in X} f(x) = \{x \in X \mid \forall y \in X : \ f(x) \geq f(y)\}.$$

  Since $\arg\max_x$ is a set, we say $x^* \in \arg\max_{x \in X} f(x)$. If $f : X \to \overline{\mathbb{R}}$ then we further require $f(x) < \infty$.

1 Introduction

There are just a few conclusions concerning the climate change that are recognized as facts. One of those facts is the ocean acidification, the decrease in the oceans pH:

> "Most carbon dioxide released into the atmosphere as a result of the burning of fossil fuels will eventually be absorbed by the ocean, with potentially adverse consequences for marine biota." [1]

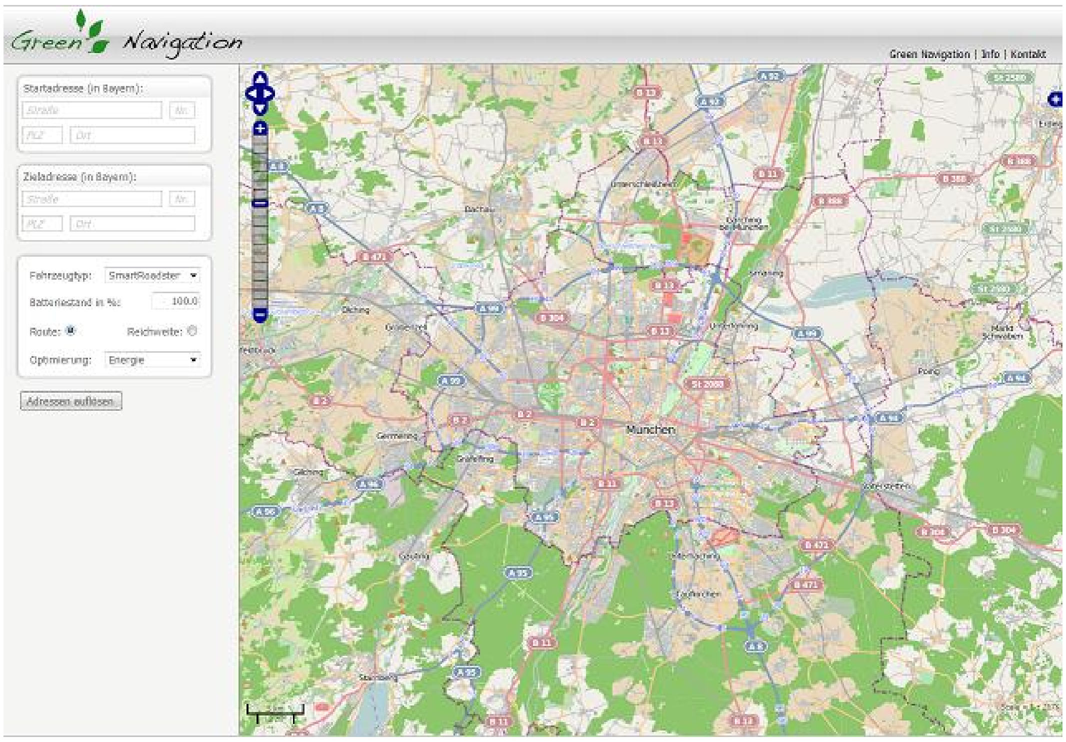
In an open letter published in the journal Science in 2010 against political assaults on climate change scientists Peter H. Gleick and more than 250 other members of the U.S. National Academy of Sciences (not speaking on its behalf) list five scientific conclusions, that are commonly referred to as facts. These are indicating why "scientists are concerned about what future generations will face from business-as-usual practices". [2]

The burning of fossil fuels is part of the problem, as stated in the above quotation. Carbon dioxide, listed as the most important factor in climate change by Moore [Moo09], is released among others by car traffic. Energy and thus carbon dioxide can be saved by improving technologies, such as the performance of vehicles. But besides developing fuel-saving technologies car navigation systems should compute and present energy efficient routes.

This thesis aims to improve routing algorithms to find energy efficient routes as part of the prototypic GreenNav system which was developed at the Technische

[1]Caldeira et al. in [CW03]

[2]Gleick et al. in [Gle10]



Startadresse (in Bayern):
Straße Nr.
PLZ Ort

Zieladresse (in Bayern):
Straße Nr.
PLZ Ort

Fahrzeugtyp: SmartRoadster
Batteriestand in %: 100.0
Route: [✓] Reichweite:
Optimierung: Energie

Adressen auflösen

Figure 1.1: The web-frontend of the GreenNav system lets you search for a route (or the range) of a specific vehicle optimized by either time, distance or energy.

Universität München and the University of Lübeck. It was designed as a client-server system providing a public web-service, which is used by different client software such as a web-frontend (see Figure 1.1) and an Android application for mobile devices. The term Green Routing is used to describe energy-efficient routing in navigation systems, but an explicit definition is still missing. The following interpretation would describe this term in the sense that it is used throughout this thesis:

Definition 1.1. Green Routing *is providing energy-efficient driving directions based on vehicle characteristics and additional network information in order to promote ecological sustainability in the context of car navigation systems.*

After reviewing the basics of routing problems in general, different aspects of energy-optimality will be discussed. The related work will be presented in the next chapter, describing the current state of energy-optimal routing and the work done so far in this field of research.

Different routing models are reviewed in Chapter 3 in order to combine different aspect into a high-fidelity model. One of these models introduced in this thesis is the state-based routing problem together with a state-based profile search, which are capable of comprising most of the other models.

The utility of partial preorder queues presented in Section 4.1 plays an important role in solving state-based routing problems. In some cases discussed in the respective section, the state-based routing problem can be efficiently solved by Dijkstra's Algorithm or by the A^*-search. Implementations of the state-based models and an adaptation of Dijkstra's Algorithm are presented in Section 4.2.

In Section 4.3 the elementary operations of the energy-optimal profile routing problem are discussed. It will be shown, that energy-optimal routing is a 'simple' problem according to the definition introduced for state-based routing. This property may be used to bound the complexity of the algorithm.

For the common shortest path problem there are various acceleration techniques. One of these techniques are contraction hierarchies. An adaption for solving state-based profile routing will be discussed in Section 4.4.

Chapter 5 lists some advantages and disadvantages of the presented models and points out the conclusions of my results, the open problems and the future work planned to improve GreenNav.

1.1 The Basic Problem

Finding paths on graphs is a large field of research, but most of the work done in this field is probably based on Dijkstra's shortest path algorithm. The shortest path problem is commonly defined as follows:

Definition 1.2. *Given a graph $G = (V, E)$, additive edge costs $C : \; E \to \mathbb{R}_{\geq 0}$ (or $C : \; E \to \mathbb{Z}_{+}$) and vertices $x, y \in V$, the (simple)* shortest path problem *(SPP) is to find a path $\pi^*_{x,y}$ from x to y in G with minimum total costs, if it exists.*

In 1959 Dijkstra published in [Dij59] a simple algorithm for (strongly-)connected graphs, i.e. there always exists at least one path between any two nodes. His original algorithm was to create a shortest path tree by using three vertex sets - describing the visited, candidate and open vertices - and three edge sets - describing the edges on optimal paths, candidates and either rejected or open edges. A simplified but complete algorithm using pseudocode is given in Algorithm 1. It was adopted from the description provided by Sniedovich in his paper [Sni06] about the dynamic programming perspective of Dijkstra's Algorithm.
In most cases Dijkstra's Algorithm is the basement for any further path finding algorithms. Even the Bellman-Ford Algorithm may be seen as a modified Dijkstra's Algorithm, that is relaxing *all* edges $|V| - 1$ times instead of just one edge at a time. The A^*-search uses a heuristic to determine the next edge relaxed. Contraction Hierarchies are a more complex algorithm using Dijkstra's Algorithm as an important subalgorithm for precomputations and its bidirectional version for querying. More is discussed in Section 4.4.

Proposition 1.3. *Dijkstra's shortest path algorithm computes a path from x to y with least costs, if it exists.*

Proof. To prove the correctness of Dijkstra's Algorithm, one needs to see that the algorithm constructs a shortest path tree with root x. A tree on V has exactly

Algorithm 1 Dijkstra's Shortest Path Algorithm

```
Require: $G = (V, E), C : E \to \mathbb{R}_{\geq 0}$ and $x, y \in V$.
 1: Initialize a field pred : $V \to V$.
 2: Initialize a field dist : $V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ with $\infty$ as default.
 3: Initialize the set $U \leftarrow V$.
 4: Set dist$(x) \leftarrow 0$ and the current vertex $v \leftarrow x$.
 5: while $v \neq y$ and dist$(v) < \infty$ do
 6:     Remove vertex $v$ from $U$.
 7:     for each successor $w$ of $v$, $w \in U$ do
 8:         Compute tentative distance $d \leftarrow$ dist$(v) + C(v, w)$.
 9:         if $d <$ dist$(w)$ then
10:             Set dist$(w) \leftarrow d$.
11:             Set pred$(w) \leftarrow v$.
12:         end if
13:     end for
14:     Choose any $v \in \arg\min_{v' \in U}$ dist$(v')$ or break, if there is no such $v$.
15: end while
16: if dist$(y) = \infty$ then
17:     return "no path of finite costs from $x$ to $y$"
18: else
19:     Initialize sequence $P$ with $y$.
20:     Set $v \leftarrow y$.
21:     while pred$(v)$ is not equal to $x$ do
22:         Set $v \leftarrow$ pred$(v)$.
23:         Add $v$ to the front of $P$.
24:     end while
25:     return Path $P$ from $x$ to $y$.
26: end if
```

one path between any two vertices, a shortest path tree with root x describes all shortest paths from x to any other vertex. Such a tree always exists (but it may not be unique) because of the principle of optimality, i.e. any subpath of an optimal path is also optimal.

By induction over the number k of vertices visited so far (line 6), we will prove that the algorithm constructs a shortest path tree. For vertex x the optimal distance is 0, which is set by the algorithm in line 4. Now assume that $S = V \setminus U$ is the set of k vertices visited so far such that their distance values are optimal. Let v be the $(k + 1)$-th vertex visited. By contradiction: If its distance was not optimal, then it was incorrectly set in line 15 with a predecessor $w \in S$ within a previous loop run. Since w by induction hypothesis has optimal distance value for evaluating v, there must be another vertex $w' \in S$ with $\text{dist}(w') + c(w', v) < \text{dist}(w) + c(w, v)$. However, since by the time v is selected in line 6, the vertex w' has been visited before, because the current vertex $v \in U$ is always chosen to be minimal. Therefore, the distance value would be set using w' instead of w, which is a contradiction to the assumption, that w is the correct predecessor of v. Thus, the algorithm constructs a shortest path tree.

Since any path from x to y in the constructed tree is an optimal path in G, the path from x to y can be determined by the predecessor field pred. Therefore, the algorithm returns the shortest path.

If the algorithm does not return a path, then the determined distance is infinity, which happens only, if y is not reachable from x. This is checked by the algorithm, because the tree constructed visits all reachable vertices. Thus if no path from x to y exists, the algorithm will correctly terminate with no path.

$\square$

1.2 Energy Optimality

The shortest path is not always the most energy-efficient path. [3]

One may think, that for energy-optimal paths we just need to label the edges with energy costs. However, by using the simple shortest path problem different aspects of Green Routing are ignored.
First it is not a trivial problem to predict the energy costs traveling along one edge in the graph. Of course we may use empirical data, but since the road network may change, we always need to collect new data for each new road being built. Furthermore, having different types of vehicles, we need to collect data for each type of car. All in all we would prefer reliable predictions of energy costs along edges, computed from the static data provided.
Furthermore, by using the shortest path problem, we assume that each type of vehicle consumes the exact same amount of energy under any condition, e.g. the time of day, the weather, the battery charge, vehicle load and so on. Obviously, this assumption is not correct, even though one would probably gain a sensible approximation by using the mean value of the empirically collected energy costs.
The overall idea for a precise energy-cost prediction model is to have a static description and dynamic information of edges as well as a description and the current state of a vehicle. This thesis aims to provide a theoretical background for implementing state based routing. The vehicle models will not be discussed in detail here, as this was done for example by Neubauer in her diploma thesis [Neu10]. Instead, some aspects are enlisted, that may be important algorithmically to Green Routing.
A first example for the limitations of the simple shortest path problem is given by Sachenbacher et al. [SLAH11] using elevation data and battery constraints. The energy cost of an edge is not only computed as a function of the distance between

[3]Miscellaneous authors, for example [KFKN08].

two vertices, but as a function of the distance and the potential energy gained (or lost). Since driving downwards may cause recuperation, i.e. regaining energy in electric vehicles from transforming potential energy, the edge costs may be negative. However, by introducing a potential function - as described in Mehlhorn et al. in [MS10] - to the vertices, one can transform this graph to a non-negatively weighted graph with the same shortest paths. Determining such values for each vertex can be done by using the Bellman-Ford algorithm, but it is naturally given for road networks by their respective heights.

This thesis copes with the problem of uncertainty. Roads with many traffic lights may have a higher variance in energy use than a road with no traffic lights. Also traffic jams are important to Green Routing, because vehicles usually consume more energy in stop-and-go situations.

Another important issue may be to route vehicles not always in an egoistic manner for personal energy saving, but instead for ecological sustainability in sum for many vehicles. Nature reserves for example should be avoided when routing each vehicle egoistically: Even though the vehicles had a little higher energy consumption, it is probably more sensible than routing the traffic through nature reserves causing serious damage to the nature. This fact can be modelled by energy penalties, but a thorough analyzation still remains open.

An obvious but also serious problem is the higher fuel consumption rate at higher velocities. In urban road network energy-efficient routing would chose slow-paces areas, because it is more efficient. On highways energy-efficient routing using SPP would probably not take into account, that cars do not necessarily need to drive at the maximum allowed speed level. Both cases yield a multivariate target function, which complicates the routing.

The SPP is widely used because of its simplicity. There are many algorithms and acceleration techniques for finding such shortest paths. However, there are still features that are not covered by SPP. Therefore, an optimal path in SPP may not be an optimal path in reality.

Some few interesting aspects that are going to be addressed are:

- Optimizing more than one edge cost function, e.g. the time needed to reach a target and the distance actually driven. This is reflected in the shortest weight-constrained path problem
- Instead of using time or distance the energy use of a path may be of interest. This includes a precise vehicle model comprising battery charge, potential and kinetic energy. The energy-optimal path problem discusses these aspects.
- The weighting function could be stochastic in order to model for traffic jams, different driving behaviours or traffic lights.
  - Stochastic properties may be independent or dependent.

  Section 3.7 reviews a small collection of stochastic routing models.
- The edge costs may change by time, for example because of high traffic volume during rush hours. Furthermore, multimodal routing refers to using different types of vehicles. Thereby arriving at time to take a particular train, flight or bus must be considered. Time-dependent routing is discussed in Section 3.4.

Remarks.

- Even though the term "shortest" implies optimizing the length of a path, it is often used as a generic term that may also refer to travelling time, arbitrary costs or energy use. Therefore, the shortest path is meant to be the optimal path depending on the current context.

2 Related Work

This thesis approaches Green Routing from two perspectives: On the one hand there are different models presented for time-dependent routing, energy-optimal routing and stochastic routing. On the other hand there are acceleration techniques for the shortest path problem. This thesis aims to generalize the routing problem in order to apply these techniques.

Stochastic routing was analysed for example by Uludag et al. in [UUN$^+$09], who used the Laplace transform in order to compute the weight of a path from independent edge weights represented by cumulative distribution functions. Moreover, Polychronopoulos et al. in [PT96] analyzed the effects of stochastic dependence of edge weights. In the context of computer networks stochastic routing was also handled by Guérin et al. in [GO99], who presented different stochastic models for energy efficient routing.

When it comes to the deterministic problem of finding simple shortest paths, there are already some interesting acceleration techniques. One of those techniques are highway hierarchies and transit node routing as presented by Schultes in his dissertation [Sch08] or contraction hierarchies as presented by Geisberger in his diploma thesis [Gei08].

The idea of using battery constraints for different routing problems originates from Sachenbacher et al. in [SLAH11], who presented a modified A^*-search to find a prefix-bounded shortest path. This will be taken up in order to combine it with stochastic routing from Uludag et al. and with contraction hierarchies from Geisberger.

The physical perspective, for example the computation of precise fuel or energy consumptions as edge weights, is not part of this thesis. Interested readers may find more information for example in the diploma thesis of Neubauer [Neu10]. She presents and evaluates different fuel consumption models. Furthermore, Kluge [Klu10] describes in his dissertation mathematically and physically sophisticated models of fuel consumption and deals with time-dependent routing.

2.1 Contribution

Stochastic routing models have already been studied by differend authors, but adding battery constraints from energy-optimal routing opens up some new perspectives. A generalized path finding model will be presented, namely the state-based shortest path problem. It will be shown, that time-dependent routing, energy-optimal routing as well as stochastic routing may all be described by the state-based model.

The complexity of the presented model will be discussed shortly. The main utility for applying already known algorithms are partial preorder queues, that are introduced in Section 4.1. As an example, Dijkstra's Algorithm will be generalized to solve a class of state-based routing problems.

Contraction hierarchies are difficult to apply to the battery constrained energy-optimal routing problem, but throughout this thesis an approach will be presented not only for energy-optimal routing, but for state-based routing in general. Nevertheless, the paper published by Eisner et al. [EFS11] during the last phase of writing approaches this problem in a similar manner. Differences and similarities will be commented whenever necessary throughout this thesis.

3 Routing Models

In order to develop Green Routing algorithms, sensible models are needed. On the one hand the reality shall be presented in an abstract way for computational reasons and on the other hand feasible information shall be drawn out of the model for making decisions in reality.

The basic model, the Shortest Path Problem (SPP) was discussed in the introduction. In the next section some terms and concepts will be clarified in the way they are commonly used in literature. To gain a broader perspective to the problem, a generalization - simply called Generalized Shortest Path Problem (GSPP) - will be presented. The NP-completeness of its decision version can be easily shown. Afterwards, the State-Based Shortest Path Problem (SBSPP) will be introduced. This problem is still a generalization of the SPP and a specialization of the GSPP, but avoiding NP-hardness. Furthermore, some well-known models, for example time-dependent routing (Delling et al. in [DW09]) and energy-optimal routing (Sachenbacher et al. in [SLAH11]), will be revised in order to show, that both may be described as state-based routing problems.

3.1 Terms and Concepts

There are many different aspects that must be taken into account when developing and refining routing models. Since there already are a lot of useful models, a short description of the nomenclature will prepare for the next sections by defining the commonly used terms (e.g. [GJ90], [UUN+09], [PT96], [SLAH11]).

Shortest Path Usually we request a shortest path in the sense of some additive length. Thereby, *length* may also refer to other target functions such as travel time, costs, energy use or a linear combination of those. The term *routing* refers to algorithmic solutions of finding shortest paths, but it is used synonymously.

Weight-Constrained Besides having one target function to optimize, there is a weight defined for each edge, that is used for additional constraints. Therefore, this term refers to optimizing more than one target function.

Dynamic The edge costs depend on a configuration $x \in X$ resulting in a mapping $X \to (E \to \mathbb{R})$ of edge costs. Usually the configuration does not change during routing, otherwise it is called *time-dependent*. For a fixed configuration $x \in X$ the routing is simple, but the intention here is that precomputations must be done either for all configurations explicitely - resulting in large overhead - or in a more general sense for groups of combinations, which is usually more difficult.

Time-Dependent The edge costs are dynamic in the sense of time. So the edge cost function becomes $E \to (T \to \mathbb{R})$ for some time set T, such that traversing the graph takes up time and changes the current time $t \in T$.

Stochastic The edge costs are random variables. We choose a probability space $(\Omega, \mathcal{F}, P)$ and measurable edge costs $E \to (\Omega \to \mathbb{R})$.

Most Likely In stochastic routing the objective here is to find a path fulfilling a certain constraint (e.g. a maximum amount of costs) with maximum probability.

Expected In stochastic routing we search for a path with least expected costs. One may search using the expected edge costs as fixed edge costs, thereby finding the least expected costs because of the linearity of the expectation operator.

Recourse When traversing the graph, information can be obtained resulting in the change of edge costs. As opposed to dynamic we do not have complete information about $x \in X$ here.

Prefix-Bounded The cost of a path is defined recursively in order to realize a lower and upper bound on all prefixes of a path. This is used to realize battery constraints of electric vehicles.

Energy-Optimal This term refers to prefix-bounded shortest paths, where edge costs represent energy use or recuperation and battery constraints are realized by lower and upper bounds on the battery. Negative edge weights are explicitely allowed.

A model discussed recourse will not be discussed within this thesis. Details are given in [PT96]. Because the model searches an optimal behaviour on what information to gather in the network, this model is beyond usual path finding problems.

3.2 Generalized Shortest Path Problem

When writing programs implementing shortest path algorithms, probably the first decision made is about what datatype to use as edge weights. The SPP is defined using either positive reals or natural numbers. Sometimes also negative values are allowed. Now the property all of these sets have in common is a total order and a binary operation for chaining edge costs. However, it is also possible to route on partially preordered edge costs. First, some necessary algebraic definitions are repeated, then a definition of a generalized routing problem will be presented. After a trivial example the complexity will be discussed shortly.

3.2.1 Preliminaries

Some algebraic definitions will be revised here, because they are important throughout this chapter. First, we want to define preorders and equivalence relations.

Definition 3.1. *Given a set S, a relation $R \subseteq S^2$ is called*

- reflexive, *if and only if* $\forall x \in S: \; x \, R \, x$*,*
- irreflexive, *if and only if* $\neg \exists x \in S: \; x \, R \, x$*,*
- symmetric, *if and only if* $\forall x, y \in S: \; x \, R \, y \to y \, R \, x$*,*
- antisymmetric, *if and only if* $\forall x, y \in S: \; x \, R \, y \wedge y \, R \, x \to x = y$*,*
- transitive, *if and only if* $\forall x, y, z \in S: \; x \, R \, y \wedge y \, R \, z \to x \, R \, z$*,*
- total, *if and only if* $\forall x, y \in S: \; x \, R \, y \vee y \, R \, x$*,*
- a partial preorder, *if and only if it is reflexive and transitive,*
- a partial order, *if and only if it is an antisymmetric preorder,*
- a total preorder, *if and only if it is a preorder and total,*
- a total order, *if and only if it is a partial order and total, and*
- an equivalence relation, *if and only if it is reflexive, symmetric and transitive.*

An important lemma for routing in general is the following. It shows, that paths in a graph do not need to be ordered directly, but they may also be preordered by a naturally given order on some image set.

Lemma 3.2. *Given a function $f : X \to S$, then a (total) preorder $\leq_S$ on S* induces *a (total) preorder $\leq_X$ on X by their images:*

$$x_1 \leq_X x_2, \; \textit{if and only if } f(x_1) \leq_S f(x_2), \quad \textit{for all } x_1, x_2 \in X.$$

When using preorders, especially partial preorders, the definition of minimal elements is important for routing. In SPP we chose a path of minimal length, where the length was totally ordered. Because of the total order the minimal element (of a finite and non-empty set) is a unique value. For partial preorders there are sets of such minimal elements:

Definition 3.3. *Given a partial preorder $\leq_S$ on S the* minimal elements *of $S' \subseteq S$ with respect to $\leq_S$ are*

$$\min_{\leq_S}(S') = \{x \in S' \mid \forall y \in S' : \ y \leq_S x \to x \leq_S y\} .$$

Each partial preorder $\leq$ on a set X induces an equivalence relation $\cong$ on X in the sense, that $a \cong b$ if and only if $a \leq b$ and $b \leq a$. Notice, that $a \cong b$ does not imply $a = b$ because preorders are not necessarily antisymmetric. The relation $\cong$ is indeed an equivalence relation, because it is reflexive (due to reflexivity of $\leq$), transitive (due to the transitivity of $\leq$) and symmetric (by definition).
An equivalence class with representative $a \in X$ denoted by $[a]_{\cong}$ for an equivalence relation $\cong$ is given by

$$[a]_{\cong} = \{x \in X \mid a \cong x\} .$$

The set of all equivalence classes of X is called the quotient set and is denoted by $X_{/\cong}$.

$$X_{/\cong} = \{[a]_{\cong} \mid a \in X\} .$$

The induced strict partial order[1] $<$ of a partial preorder $\leq$ on X is given by $a < b$ if and only if $a \leq b$ and not $b \leq a$, $a, b \in X$. Throughout this thesis a is also said to *improve* b. Notice, that $a \leq b$ does not imply $a = b \ \vee \ a < b$, because $a \leq b$ is also true for $a \cong b$ where $a \neq b$.
Using the induced strict partial order $<_S$, the $\min$ operator on $\leq_S$ can be defined

[1] Because of the irreflexivity we do not need to distinguish preorders from orders here.

equivalently by

$$\min_{\leq_S}(S') = \{x \in S \mid \neg \exists y \in S : \ y <_S x\} .$$

3.2.2 Definition

An important application of orders and preorders in routing algorithms is for sorting vertices by their distance to a start or end vertex or for sorting paths by their length.

In the following definition uses not a totally ordered set of weights (such as $\mathbb{R}$ or $\mathbb{Z}$) but an arbitrary preordered set of weights $\mathbb{W}$. Concatenating edges forming paths yields weights, that are composed by an associative operation of the edge weights. Associativity is important here, because it allows to concatenate adjacent edges in any given order. Commutativity is not necessarily provided, because switching the order of the edge in a path does usually not yield a path of adjacent edges.

The quotient set will be used to define the following routing models. A set of paths $m \subseteq \Pi_{x,y}$ from x to y are weighted by elements of $\mathbb{W}$. These weights are partially preordered by $\leq_W$. One may be interested in finding at least one path for each minimal element, but we also may take only one path for each equivalence class.

For example, if a vehicle contains two batteries then it has two states representing both battery charges. We want to find an energy-efficient path, so we optimize the sum of both battery charges. If, for whatever reason, two paths yield different battery charges but the the same sum, then we are interested only in one of those paths.

Given the set of minimal weights of all paths $W = \min_{\leq_W}(\mathcal{W}(\Pi_{x,y}))$ from x to y, we are interested in at least one path corresponding to each equivalence class in the induced quotient set $W_{/\cong_W}$ of the equivalence relation $\cong_W$ induced by the partial preorder $\leq_W$. In short we will say, that we query for at least one path for each minimal element *except for equivalence*.

Definition 3.4. *Given* $(G, \mathbb{W}, \leq_W, \mathcal{W}', \oplus)$*, where*

- $G = (V, E)$ *is a (directed) graph,*
- $\mathbb{W}$ *is a set of weights,*
- $\leq_W$ *is a (partial) preorder on* $\mathbb{W}$*,*
- $\mathcal{W}' : \ E \to \mathbb{W}$ *is a weighting,*
- $\oplus : \ \mathbb{W} \times \mathbb{W} \to \mathbb{W}$ *is associative,*

such that

- *the* extension[2] *of* $\mathcal{W}'$ *is* $\mathcal{W} : \ \Gamma \to \mathbb{W}$ *is given by*

$$\mathcal{W}(\gamma) = \bigoplus_{i=1}^{k} \mathcal{W}'(v_{i-1}, v_i)$$

  for all walks $\gamma = (v_0, \dots, v_k) \in \Gamma$, $k \geq 1$,

then the generalized shortest path problem *(GSPP) for given vertices* $x, y \in V$ *is to find at least one corresponding path for each minimal element in* $\min(\mathcal{W}(\Pi_{x,y}))$ *except for equivalence. If* $x = y$*, then* $GSPP$ *yields an empty path containing just* x *and no edges.*

As it has already been mentioned, this definition generalizes the simple SPP, which may be reduced to GSPP in the following sense:

Example. Let $G = (V, E)$ be a graph and $C : \ E \to \mathbb{R}_{\geq 0}$ be additive edge costs. With $\mathbb{W} = \mathbb{R}_{\geq 0}$ we choose $\mathcal{W} = C$, $\oplus = +$ and the (total) order $\leq_W$ to be $\leq$ on $\mathbb{R}$. Now given $x, y \in V$ the set $\mathcal{W}(\Pi_{x,y})$ is either empty (if no path from x to y exists) or a finite set of positive reals containing a unique minimum element representing the length of a shortest path. The GSPP now asks for at least one corresponding path to the shortest path length. Therefore, the GSPP is a generalization of the SPP.

[2] A problem instance is often given by an edge weighting $\mathcal{W}'$, inducing a weighting on arbitrary walks. So $\mathcal{W}'$ is only needed for the definition of $\mathcal{W}$.

3.2.3 Intractability

The decision version of GSPP may be described by asking, if there is a $w^* \in \mathcal{W}(\Pi_{x,y})$ with $w^* \leq w$ for $w \in \mathbb{W}$. Obviously this problem is NP-complete because we may choose $\geq$ for the ordering of weights in the above example yielding the longest path problem. The decision version of the longest path problem is known to be NP-complete.
Nevertheless, a specialization of the GSPP in the sense that its algebraic structure is specialized but the queried paths are just a subset of those from the GSPP. The model presented next is a state-based approach avoiding NP-completeness.

3.3 State-Based Routing

The following model aims to be a specialization of the GSPP in order to avoid NP-completeness. The main idea here is to have a set of states, which will be changed by traversing edges that are weighted with state transformation functions. This is similar to time-dependent routing and energy-optimal routing as can be seen later. There are different options about what exactly to query from the model, one is to find an optimal policy for all possible starting states and another one is to find a particular path for a given starting state. This differs from the GSPP because in GSPP at least one path for each minimal element is queried (see Figure 3.1).

3.3.1 Preliminaries

Definition 3.5. *A function $f : X \to Y$ with $X, Y \subseteq S$ is* monotone *with respect to a preorder $\leq$ on S, if and only if $x_1 \leq x_2 \to f(x_1) \leq f(x_2)$ for all $x_1, x_2 \in X$.*

Definition 3.6. *A function $f : X \to Y$ with $X, Y \subseteq S$ is* extensive *with respect to a preorder $\leq$ on S, if and only if $x \leq f(x)$ for all $x \in X$.*

The following lemma may be obvious, but it is essential for routing.

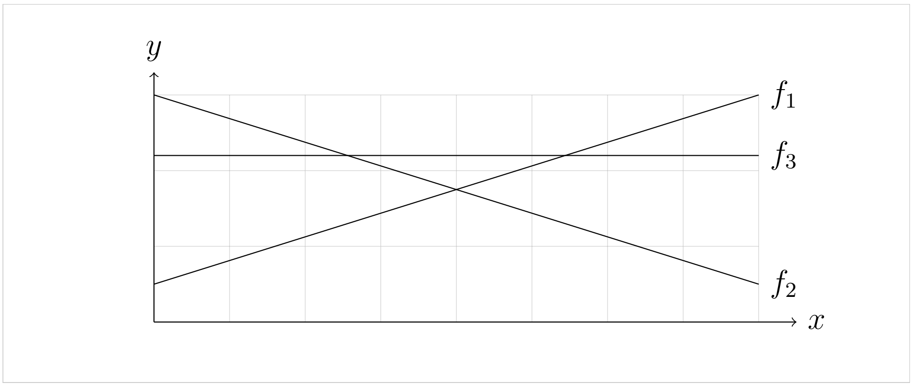


Figure 3.1: Three functions $f_1, f_2, f_3 : \mathbb{R} \to \mathbb{R}$. The GSPP queries paths corresponding to all functions f_1, f_2, f_3, because each of them is minimal. State-based routing shall ask only for f_1 and f_2, because f_3 does not *improve* the combination $\min(f_1, f_2)$.

Lemma 3.7. *Monotonicity and extensivity are both preserved under functional composition.*

Proof. Let $f : X \to Y,\ g : Y \to Z$ with $X, Y, Z \subseteq S$ and a preorder $\leq$ on S.
If both functions are monotone, then $f(x_1) \leq f(x_2)$ for all $x_1, x_2 \in X$ with $x_1 \leq x_2$ and $g(y_1) \leq g(y_2)$ for all $y_1, y_2 \in Y$ with $y_1 \leq y_2$. For $y_1 = f(x_1)$ and $y_2 = g(x_2)$ we have $(f \circ g)(x_1) = g(f(x_1)) \leq g(f(x_2)) = (f \circ g)(x_2)$ for all $x_1, x_2 \in X$ with $x_1 \leq x_2$. Therefore, $f \circ g$ is monotone.
If both functions are extensive, then $x \leq f(x)$ for all $x \in X$ and $y \leq g(y)$ for all $y \in Y$. For $y = f(x)$ we have $x \leq f(x) \leq g(f(x)) = (f \circ g)(x)$. Therefore, $f \circ g$ is extensive.

□

3.3.2 Definition

The first definition of state-based routing is used to search specific paths for a given starting state while the second definition is used to perform a profile search. In the following, graphs of vertices and (directed) edges $G = (V, E)$ are specified, such that each vertex is labelled with a set of possible states. These states denoted by S are used for example to describe the current battery charge in energy-optimal routing, the time spent so far on a route or any other preordered set. The edges are labelled with weights, which are state transformation functions. These weights must fulfill certain properties. When thinking of Dijkstra's Algorithm, negative edge weights were not allowed. This corresponds to extensivity of weights. Another important property coming from time-dependent routing (see [DW09]) is monotonicity.

Definition 3.8. *Given* $(G, S, \leq_S, \mathcal{S}, \mathbb{W}, \mathcal{W}')$*, where*

- $G = (V, E)$ *is a (directed) graph,*
- S *is a set of states,*
- $\leq_S$ *is a (partial) preorder on* S*,*
- $\mathcal{S} : V \to \mathcal{P}(S)$ *describes possible states at each vertex,*
- $\mathbb{W}$ *is the set of monotone and extensive weights* $S \rightsquigarrow S$*,*
- $\mathcal{W}' : E \to \mathbb{W}$ *is a weighting,*

such that

- $\mathcal{W}'(x, y)$ *is a weight* $\mathcal{S}(x) \to \mathcal{S}(y)$*,*
- *the extension of* $\mathcal{W}'$ *again is* $\mathcal{W} : \Gamma \to \mathbb{W}$ *given by*

$$\mathcal{W}(\gamma) = \mathcal{W}'(v_0, v_1) \circ \ldots \circ \mathcal{W}'(v_{k-1}, v_k)$$

  for all walks $\gamma = (v_0, \ldots, v_k) \in \Gamma$*,* $k \geq 1$*,*

then the state-based shortest path problem *(SBSPP) for given vertices* $x, y \in V$ *and initial state* $s \in \mathcal{S}(x)$ *is to find at least one corresponding path for each minimal element in* $\min(\mathcal{W}(\Pi_{x,y})(s))$ *except for equivalence.*

Remarks.

- Remember, that a minimal state $s \in S$ is a state that can not be improved, i.e. for all states $s' \in S$ we have $s' \leq_S s \to s \leq_S s$. This may also be described using the induced strict partial order by not having $s' <_S s$ (see Section 3.2.1).

- It is sensible to introduce a garbage state. When two vertices x, y are connected but a path from x to y is not of interest, because e.g. the battery charge is not sufficient, then a path yielding such a garbage state may be pruned. This is implicitely done for energy-optimal routing and not discussed further, but one may also introduce such a garbage state as part of above definition.

- Shortest path problems usually are defined having a total order or total preorder. An SBSPP with totally (pre-)ordered states is simpler in the sense, that $\min(\mathcal{W}(\Pi_{x,y}))$ contains only one state except for equivalence.

- Monotonicity here is quite strong because if two states $s_1, s_2 \in S$ are equivalent, i.e. $s_1 \cong_S s_2$ meaning $s_1 \leq_S s_2$ and $s_2 \leq_S s_1$, then their transformed states are also equivalent: $w(s_1) \cong_S w(s_2)$. Therefore, you may see SBSPP as routing on values of the quotient set of the equivalence relation induced by $\leq_S$.

- To show that another problem may be solved by algorithms developed for SBSPP, we want to reduce this problem to SBSPP by defining the tuple $(G, S, \leq_S, \mathcal{S}, \mathbb{W}, \mathcal{W}')$. Furthermore, it needs to be shown that the weight functions $\mathbb{W}$ are monotone and extensive.

- The extension was defined for the GSPP, such that $\mathcal{W}(\gamma) = \bigoplus_{i=1}^{k} \mathcal{W}'(v_{i-1}, v_i)$ for walks $\gamma = (v_0, \ldots, v_k) \in \Gamma$. When using function composition, it is important to notice the ordering. Since we transform first the starting state, then transform the transformed starting stated and so on, we have: $\mathcal{W}(\gamma)(s) =$

$\mathcal{W}'(v_{k-1}, v_k)(\ldots(\mathcal{W}'(v_0, v_1)(s)))$ for all starting states $s \in S$. In order to keep the ascending order we have defined the notation of functional composition $(f \circ g)(x)$ to be $g(f(x))$ for functions $f : X \to Y$ and $g : Y \to Z$.

In order to handle the state-based approach in a more elegant way, we will introduce routing policies. These are interesting for SBSPPs with partially preordered states as well as for profile routing, because they are used to collect all optimal paths.

Definition 3.9. *Let $G = (V, E)$ be a graph, then a* policy $m \subseteq \Pi_{x,y}$ *is a set of paths from x to y. The set of all policies is denoted by $M = \bigcup_{x,y \in V} \mathcal{P}(\Pi_{x,y})$. Let $\leq_M$ be a partial preorder on M satisfying $m_1 \leq_M m_2$ for all $m_1, m_2 \in M$ with $m_1 \supseteq m_2$.*

- *A path $\pi \in \Pi_{x,y}$ is said to* improve *a policy $m \subseteq \Pi_{x,y}$, if and only if $m \leq_M \{\pi\} \cup m$ is not satisfied.*
- *A policy $m \subseteq \Pi_{x,y}$ is said to be* optimal *(or* not shrinkable*), if and only if all paths $\pi \in m$ improve $m \setminus \{\pi\}$, i.e. no path $\pi \in m$ satisfies $m \leq_M m \setminus \{\pi\}$.*
- *A policy $m \subseteq \Pi_{x,y}$ is said to be* complete *(or* not improvable*), if and only if there is no path $\pi \in \Pi_{x,y}$ improving m, i.e. $m \leq_M \{\pi\} \cup m$ for all paths $\pi \in \Pi_{x,y}$.*
- *The* combination *m of policies $m_1, \ldots, m_k \subseteq \Pi_{x,y}$ means the union of both path sets*
$$m = \bigcup_{i=1,\ldots,k} m_i \subseteq \Pi_{x,y}.$$
- *The* concatenation *m of policies $m_1 \subseteq \Pi_{v_0,v_1}, \ldots, m_k \subseteq \Pi_{v_{k-1},v_k}$ means the set of elementwise path concatenations*
$$m = \{\pi_1 \circ \ldots \circ \pi_k \mid \pi_i \in m_i \text{ for } i = 1, \ldots, k\} \cap \Pi_{v_0,v_k}.$$

Remarks.

- We have $m \leq_M \emptyset$ and $\Pi_{x,y} \leq_M m$ for each policy $m \subseteq \Pi_{x,y}$. The policy $\emptyset$ is worse and the policy $\Pi_{x,y}$ is best in the sense, that a driver usually prefers any path over none and that taking all possible paths into account surely yields an optimal path.

- Since $\leq_M$ is reflexive, a path $\pi \in m$ of a policy m does not improve m. This means, that the policy containing all paths from x to y is complete.

- An optimal and complete policy does not necessarily have minimal cardinality among all complete policies. Finding a minimal and complete policy is NP-hard. More specifically, determining wether there is an optimal and complete policy of some specific cardinality $k > 0$, is NP-hard, because the 'set covering' problem, which is listed as one of Karp's 21 NP-complete problems in [Kar72], can be easily reduced to it (without proof).

- The reason for defining $\leq_M$ on the union of all policies is to provide algorithms with necessary comparisons. Dijkstra's Algorithm for example compares two weights even though they only share a common start vertex but not a common end vertex. Furthermore, for bidirectional searches we want to compare policies that share a common end vertex but not a common starting vertex.

- Regarding SBSPP we want to find a path π, such that $\mathcal{W}(\pi)(s)$ is minimal for some starting state $s \in \mathcal{S}(x)$. We may describe the partial preorder on policies by $m_1 \leq_M m_2$, if and only if $\mathcal{W}(\pi_1)(s) \leq_S \mathcal{W}(\pi_2)(s)$ for all paths $\pi_i \in m_i$ and policies $m_i \in \bigcup_{y \in V} M_{x,y}$, $i \in \{1, 2\}$. But if m_1 and m_2 do not start in a common vertex x, then we do not know, which path is 'better', because the ordering depends on an initial state $s \in \mathcal{S}(x)$ of a common vertex. That way, backward and bidirectional searches are difficult.

The aim is to show, how the relation $\leq_M$ may be induced from the relation $\leq_S$ on states in way, that policies with different start and end vertex are comparable. Doing so, we may search for an optimal and complete policy from x to y independent of a starting state. This is called a profile search and was described informally for time-dependent routing by Delling et al. in [DW09]. We formalize this search in terms of state-based routing in the following way.

Definition 3.10. *Given the structure of an SBSPP, i.e. the tuple $(G, S, \leq_S, \mathcal{S}, \mathbb{W}, \mathcal{W}')$, the partial preorder $\leq_M$ on policies $M = \bigcup_{x,y \in V} \mathcal{P}(\Pi_{x,y})$ is induced by functional comparison using $\leq_S$: Given any two policies $m_1 \subseteq \Pi_{x_1,y_1}$, $m_2 \subseteq \Pi_{x_1,y_2}$ then $m_1 \leq_M m_2$, if and only if $s_1 \leq_S s_2$ implies*

$$\forall \pi_2 \in m_2 \, \exists \pi_1 \in m_1 : \; \mathcal{W}(\pi_1)(s_1) \leq_S \mathcal{W}(\pi_2)(s_2)$$

for all $s_1 \in \mathcal{S}(x_1), s_2 \in \mathcal{S}(x_2)$.
Now given two vertices $x, y \in V$ the state-based profile routing problem (SBPRP) *is to find an* optimal and complete policy $m \subseteq \Pi_{x,y}$.

Remarks.

- The property $m_1 \leq_M m_2$ for all $m_1, m_2 \in M$ with $m_2 \subseteq m_1$ is fulfilled, because for all $\pi_2 \in m_2$ there is a $\pi_1 \in m_1$, namely $\pi_1 = \pi_2$, such that $\mathcal{W}(\pi_1)(s_1) \leq_S \mathcal{W}(\pi_2)(s_2)$ because of the monotonicity of weights, which is preserved under functional composition (see Lemma 3.7).

- The induced relation $\leq_M$ is indeed a preorder:
  - Reflexivity: $m \leq_M m$ for any $m \subseteq \Pi_{x,y}$ is given, because $m \subseteq m$ (see previous remark).
  - Transitivity: With $m_1 \leq_M m_2$ and $m_2 \leq_M m_3$, $m_i \subseteq \Pi_{x_i,y_i}$ and with any states $s_1 \leq_S s_2$ and $s_2 \leq_S s_3$, $s_i \in \mathcal{S}(x_i)$, $i \in \{1, 2, 3\}$, we have $\mathcal{W}(\pi_1)(s_1) \leq_S \mathcal{W}(\pi_2)(s_2)$ for some $\pi_1 \in m_1$ and all $\pi_2 \in m_2$

as well as $\mathcal{W}(\pi_2)(s_2) \leq_S \mathcal{W}(\pi_3)(s_3)$ for some $\pi_2 \in m_2$ and all $\pi_3 \in m_3$. Therefore, for all $\pi_3 \in m_3$, there is a $\pi_2 \in m_2$ and a $\pi_1 \in m_1$ with $\mathcal{W}(\pi_1)(s_1) \leq_S \mathcal{W}(\pi_2)(s_2)$ and $\mathcal{W}(\pi_2)(s_2) \leq_S \mathcal{W}(\pi_3)(s_3)$ following $\mathcal{W}(\pi_1)(s_1) \leq_S \mathcal{W}(\pi_3)(s_3)$ by transitivity of preorder $\leq_S$.

- By requiring $\leq_M$ to be a preorder and thus to be reflexive, the given definition induces monotonicity on weights. However, the edge functions are usually given by the problem and the preorder on M is derived from that. Therefore, it is sensible to explicitely require the edge functions to be monotone.

Again, these definitions generalize the simple SPP, which may be reduced to SBSPP (as well as SBPRP) in the following sense:

Example. Let $G = (V, E)$ be a graph and $C : E \to \mathbb{R}_{\geq 0}$ be additive edge costs. With $S = \mathcal{S}(v) = \mathbb{R}_{\geq 0}$ for all $v \in V$, we choose $\leq_S$ to be the usual comparison $\leq$ on reals. The weight functions $w_c \in \mathbb{W}$, $e \in E$ are (monotone and extensive) translation functions $w_c(x) = x + c$ for $c \in \mathbb{R}_{\geq 0}$, such that the weighting is given by $\mathcal{W}(e) = w_{C(e)}$ for all $e \in E$. Now given $x, y \in V$ and (an arbitrary) starting state $s = 0$, the SBSPP asks for a path $\pi^*_{x,y}$ with weight w_c, such that $w_c(s) = w_c(0) = c$ is minimal, if such a path exists. Therefore, the SBSPP is a generalization of SPP. Furthermore, the SBPRP solves the problem for all starting states, but because the optimal path is independent of the starting state, the computed policy always yields just the optimal path.

Furthermore, the longest path problem may not be directly transformed into a state-based model, because the edge weight functions are not extensive, i.e. the weights are not decreasing, but increasing the states. The same may also hold for negatively weighted edges in SPP, because if there are cycles of negative weight, then we can not use potential functions as described in [MS10].

So far we have seen the structure of the problem called SBSPP, but when developing algorithms we have an interest in bounding the complexity of all comprised modules. In most cases, it is necessary to refer to an actual implementation. Remember, that SBSPP with totally and with partially preordered states were differed explicitely. The former is usually less complex than the latter.

Definition 3.11. *An SBSPP is called* simple, *if and only if*

1. *the preorder $\leq_S$ is total,*
2. *the states S have constant descriptive complexity and may be compared by $\leq_S$ in constant time, and*
3. *the edge weights functions $\mathcal{W}(E)$ have constant descriptive complexity and may be computed in constant time.*

The SPP is a simple SBSPP. Simple SBSPPs may be directly solved using Dijkstra's Algorithm (see Section 4.2).
The following definition tries to bound the complexity of profile searches.

Definition 3.12. *An SBPRP is called* simple (in $\mathcal{O}(X)$), *if and only if*

1. *the underlying SBSPP is simple,*
2. *the complexity of policies $m_{x,y} \subseteq \Pi_{x,y}$ is bounded by $\mathcal{O}(X)$ in the worst case, i.e. the cardinality $|m_{x,y}|$ is in $\mathcal{O}(X)$ for all vertices $x, y \in V$ (thereby X may be a function of $|V|$ and $|E|$), and*
3. *the concatenation, combination and comparison of two policies m_1 and m_2 may be realized in time $O(|m_1| + |m_2|)$ in the worst case.*

For completeness the existance of an optimal and complete policy will be shown.

Lemma 3.13. *An optimal and complete policy from x to y does always exist.*

Proof. Given any non-optimal policy $m \subseteq \Pi_{x,y}$, then there is a path $\pi \in m$, such that

$$\forall s \in \mathcal{S}(x)\, \exists \pi' \in m : \mathcal{W}(\pi')(s) \leq_S \mathcal{W}(\pi)(s).$$

Thus we have $m \setminus \{\pi\} \leq_M m$. Therefore, any non-optimal policy can be reduced step by step to an optimal policy by removing the paths just described π, such that completeness would be conserved. Remember, that an empty policy is optimal by definition.

Now the policy $\Pi_{x,y}$ is complete and can be reduced to an optimal policy that remains complete. Therefore, there always exists an optimal and complete policy.

□

Since profile routing solves the state-based shortest path problem for all starting states (not just for one particular), a solution to SBPRP always describes a solution to the corresponding SBSPP. This is intuitive but a formal proof is given in the following.

Proposition 3.14. *Given $m \leq_M \emptyset \rightarrow m = \emptyset$, then a solution to SBPRP always comprises a correct solution to the corresponding SBSPP.*

Notice that it may be quite complex to determine the correct solutions of an SBSPP by the solution of an SBPRP if the found policy is set of many paths. For real road networks this cardinality is usually assumed to be small. This assumption is stated for example by Eisner et al. [EFS11].

Proof. Let $(G, S, \leq_S, \mathcal{S}, \mathbb{W}, \mathcal{W}')$ be the structure of an SBSPP and the corresponding SBPRP. Given a solution to the SBPRP, we have an optimal and complete policy $m \subseteq \Pi_{x,y}$ describing optimal paths from x to y.

If $m = \emptyset$, then y is not reachable from x. Otherwise the policy would not be complete, we had a path $\pi \in \Pi_{x,y}$ such that $\{\pi\} <_M \emptyset$ contradicting the presumption. Therefore, assume that $m \neq \emptyset$.

Given a starting state $s \in \mathcal{S}(x)$, let $\pi \in m$, such that $\mathcal{W}(\pi')(s) \leq_S \mathcal{W}(\pi)(s)$ follows $\mathcal{W}(\pi)(s) \leq_S \mathcal{W}(\pi')(s)$ for all $\pi' \in m$. Such a path always exists, because of the transitivity of $\leq_S$.
Since m is complete, there is no path $\pi' \in \Pi_{x,y}$ improving the policy m, i.e. we do not have $\{\pi'\} \cup m \leq_M m$ for any path $\pi' \in \Pi_{x,y}$.
This results in

$$\exists s_1, s_2 \in \mathcal{S}(x) : \; s_1 \leq_S s_2 \rightarrow \exists \pi_2 \in m \;\; \neg \exists \pi_1 \in \{\pi'\} \cup m : \; \mathcal{W}(\pi_1)(s_1) \leq_S \mathcal{W}(\pi_2)(s_2).$$

Because of monotonicity and equal domains, $s := s_1 = s_2$ may be chosen, which yields

$$\exists \pi_2 \in m \;\; \neg \exists \pi_1 \in \{\pi'\} \cup m : \; \mathcal{W}(\pi_1)(s) \leq_S \mathcal{W}(\pi_2)(s).$$

Since π' was chosen arbitrarily, no such path improves the resulting state, i.e. an optimal state is reached through a path $\pi \in m$. Therefore, an optimal state is reached from within the complete policy m.

$\square$

The presumption $m \leq_M \emptyset \rightarrow m = \emptyset$ is sensible, because any path is usually better than no path. On the other hand, when considering energy-optimal routing (Section 3.5) then paths may be worthless because they can not be traversed for any starting state. In that case, SBPRP yields an empty set representing no paths where SBSPP yields an arbitrary path. Because both cases are easily identified, we do not distinguish them explicitely.

3.4 Time-Dependent Shortest Path Problem

Time-dependency is modeled "by using functions for specifying edge weights" (Delling et al. in [DW09]). So the actual edge weight depends on the time of arrival (or previous path costs so far). This model is revised here, because it perfectly fits in the definition of state-based routing and thus is an interesting example.

Definition 3.15. *Let $T = \mathbb{R}$ be the set of departure times and let $\mathbb{F}$ be the function space of functions $f : T \to \mathbb{R}_{\geq 0}$ with $f(t_1) + t_1 \leq f(t_2) + t_2$ for all $t_1, t_2 \in T$, $t_1 \leq t_2$. Now given a graph $G = (V, E)$, a weighting $\mathcal{F} : E \to \mathbb{F}$, vertices $x, y \in V$ and starting time $t \in T$, the* time-dependent shortest path problem *(TDSPP) is to find a path π^*_{xy} from x to y with minimum weight with respect to $g : \Pi \times T \to \mathbb{R}$ defined recursively for a path $(v_0, \ldots, v_k)$ and starting time t by $g(v_0, t) = t$ and $g((v_0, \ldots, v_k), t) = g((v_0, \ldots, v_{k-1}), t) + \mathcal{F}(v_{k-1}, v_k)(g((v_0, \ldots, v_{k-1}), t))$.* [3]

The property $f(t_1) + t_1 \leq f(t_2) + t_2$ for all $t_1, t_2 \in T$ is called FIFO property (or non-overtaking property). It is used to ensure, that waiting does not provide any advantage. If this property was not fulfilled and if waiting was not allowed, then the decision version of TDSSPP would be NP-complete (Orda et al. in [OR90]). Notice that this property is fulfilled if and only if the function $g(t) = f(t) + t$ is monotonically increasing, which will be of interest in the state-based approach.

3.4.1 State-Based Time-Dependency

The following definition is an alternative definition of the same problem, but formulated as a state-based routing problem.

Definition 3.16. *Let $T = \mathbb{R}$ be the set of departure times and let $\mathbb{F}$ be the function space of functions $f : T \to \mathbb{R}_{\geq 0}$ with $f(t_1) + t_1 \leq f(t_2) + t_2$ for all $t_1, t_2 \in T$, $t_1 \leq t_2$. Now given a graph $G = (V, E)$, a weighting $\mathcal{F} : E \to \mathbb{F}$ and vertices $x, y \in V$ as well as a transformation $\tau : \mathbb{F} \to (T \to T)$ with $\tau(f)(t) = f(t) + t$ for all $t \in T$, the* time-dependent shortest path problem *(TDSPP2) is to solve the SBSPP given by $(G, T, \leq, \mathcal{S}, \tau(\mathbb{F}), \mathcal{F} \circ \tau)$, $\mathcal{S}(v) = T$ for all $v \in V$.*

The monotonicity and extensivity are easily checked:

[3]Delling et al. in [DW09], slightly changed for consistency. The periodicity of functions in $\mathbb{F}$ was omitted.

Remarks.

- Monotonicity: Let $t_1, t_2 \in T$ with $t_1 \leq t_2$, then $\tau(f)(t_1) = f(t_1) + t_1 \leq f(t_2) + t_2 = \tau(f)(t_2)$ for all functions $f \in \mathbb{F}$ by definition.

- Extensivity: Let $t \in T$, then $f(t) \geq 0$ (since $f : T \to \mathbb{R}_{\geq 0}$) and $t \leq t + f(t) = \tau(f)(t)$ for all functions $f \in \mathbb{F}$.

Theorem 3.17. *Both definitions of the time-dependent shortest path problem, TDSPP and TDSPP2, are equivalent.*

Proof. The only difference in both models is the definition of edge and path weights. Let $(v_0, \ldots, v_k)$ be a path in G. In TDSPP we have a recursive definition:

$$\begin{aligned} g(v_0, t) &= t, \\ g((v_0, \ldots, v_k), t) &= g((v_0, \ldots, v_{k-1}), t) + \mathcal{F}(v_{k-1}, v_k)(g((v_0, \ldots, v_{k-1}), t)). \end{aligned}$$

In TDSPP we have a composition of functions:

$$\mathcal{W}((v_0, \ldots, v_k)) = \mathcal{W}'(v_0, v_1) \circ \ldots \circ \mathcal{W}'(v_{k-1}, v_k).$$

We need to show, that $g((v_0, \ldots, v_k), t) = \mathcal{W}((v_0, \ldots, v_k))(t)$ for all $t \in T$. This is easily done by induction on k:

Base Let $k = 0$, then $g((v_0, \ldots, v_k), t) = g(v_0, t) = t$ and $\mathcal{W}((v_0, \ldots, v_k)) = \mathcal{W}(v_0) = \mathrm{id}_T$, so $\mathcal{W}((v_0, \ldots, v_k))(t) = t = g((v_0, \ldots, v_k), t)$ for all $t \in T$.

Hypothesis Let $x_t = g((v_0, \ldots, v_i), t) \overset{*}{=} \mathcal{W}((v_0, \ldots, v_i))(t)$ for all $t \in T$ for some fixed $i \in \mathbb{Z}_+$, $i < k$.

Step The induction step follows directly by using definitions:

$$\begin{aligned}
g((v_0,\dots,v_i,v_{i+1}),t) &= g((v_0,\dots,v_i),t) + \mathcal{F}(v_i,v_{i+1})(g((v_0,\dots,v_i),t)) \\
&\overset{*}{=} x_t + \mathcal{F}(v_i,v_{i+1})(x_t) \\
&= \tau(\mathcal{F}(v_i,v_{i+1}))(x_t) \\
&= \mathcal{W}'(v_i,v_{i+1})(x_t) \\
&\overset{*}{=} \mathcal{W}'(v_i,v_{i+1})(\mathcal{W}((v_0,\dots,v_i))(t)) \\
&= (\mathcal{W}((v_0,\dots,v_i)) \circ \mathcal{W}'(v_i,v_{i+1}))(t) \\
&= \mathcal{W}((v_0,\dots,v_i,v_{i+1}))(t)
\end{aligned}$$

We have seen now, that both weights g and $\mathcal{W}$ are equivalent. The TDSPP now asks for a path π^*_{xy}, such that g is minimal, whereas the TDSPP2 asks for an optimal path π^*_{xy} with minimal projected state $\mathcal{W}(\pi^*_{xy})(t)$, if such a path exists. Therefore, both models are equivalent.

□

Property 3.11 defines simple SBSPPs. The time-dependent routing problem is not simple, since the function space $\mathbb{F}$ was not constrained. Therefore, we can not decide yet, e.g. if composing weight functions is possible in constant time. Furthermore, the profile-search in time-dependent routing may also be formulated by a state-based approach as SBPRP, which follows from Theorem 3.17 and Proposition 3.14.

3.5 Energy-Optimal Path Problem

The following model was investigated by Sachenbacher et al. in [SLAH11] and [AHLS10]. It is the grounding for the energy-optimal stochastic models as well as the inspiration for state-based routing in general. The main difference to SPP allowing negative edge weights is the definition of path costs. Contrary to SPP, the path costs are not the sum of its edges but defined recursively in order to model for battery constraints. More precisely, the two following definitions are adapted from their model.

Definition 3.18. *The* energy costs $C_{KJ}(\gamma^k)$ of a walk $\gamma^k = (v_0, \dots, v_k)$, *given edge costs* $C : E \to \mathbb{R}$, *the battery capacity* $K \in \mathbb{R}$ *and its initial charge* $J \in \mathbb{R}$ *with* $0 \leq J \leq K$, *is defined recursively for* $i = 0, \dots, k$*:*

$$C_{KJ}(\gamma^i) = \begin{cases} K - J & \text{if } i = 0, \\ 0 & \text{if } i > 0 \text{ and } \Delta^i < 0, \\ \Delta^i & \text{if } i > 0 \text{ and } 0 \leq \Delta^i \leq K, \\ \infty & \text{if } i > 0 \text{ and } \Delta^i > K \end{cases}$$

where

$$\Delta^i = C_{KJ}(\gamma^{i-1}) + C(v_{i-1}, v_i) \qquad \text{for } i > 0.$$

Using this definition the energy optimal routing problem follows immediately:

Definition 3.19. *Given a graph* $G = (V, E)$, *vertices* $x, y \in V$, *edge costs* $C : E \to \mathbb{R}$, *such that there are no cycles of negative energy costs, battery capacity* $K \in \mathbb{R}$ *and initial charge* $J \in \mathbb{R}$ *with* $0 \leq J \leq K$, *the* energy-optimal path problem (EOPP) *is to find a path* π^*_{xy} *from* x *to* y *with minimum total energy cost* $C_{KJ}(\pi^*_{xy}) < \infty$, *if it exists.* [4]

Remarks.

- Negative edge costs are explicitely allowed to model for energy recuperation.
- Since paths by definition are walks (with distinct vertices), the definition for energy costs of walks may also be used to determine the energy costs of a path.

[4]Sachenbacher et al. in [SLAH11], slightly changed for consistency.

- It is sensible to assume, that each cycle has non-negative costs in sum. Algorithmically this has the advantage of not running cycles in order to gain energy. Furthermore, this assumption is based on thermodynamics: Assuming the system does not gain energy from outside (which is of course not true for solar vehicles, but these are ignored here), the battery charge can not increase by driving cycles. Furthermore, the vehicle actually loses energy due to friction and heating to the outside.

As was done with time-dependent routing problems we do now reformulate the energy-optimal routing problems as state-based routing problems. However, this is more difficult in various senses. First, the description of the state transformation functions will be more complex than how they were done in the previous section, actually just because there was no detailed specification of the function space there. Furthermore, the edge weights must be allowed to be negative, which contradicts extensivity of edge weights. In order to solve this problem, potential functions (Mehlhorn et al. in [MS10]) may be used in the same way as they were used by Sachenbacher et al. in [SLAH11] for energy-optimal routing. This is done by combining the battery charge and the potential energy to one combined energy state.

3.5.1 Battery and Energy Transformation Functions

The alternative definition of the energy-optimal shortest path problem requires the definition of simple battery transformation functions. It is similar to the edge cost function of Eisner et al. in [EFS11], but instead of chaining functions in a particular way, functional composition on state transformations is used here, which should be more intuitive and simple. Moreover, the domain is explicitely defined, i.e. the set of battery charges.

Definition 3.20. *Given a* battery capacity $K \in \mathbb{R}_{\geq 0}$, *the set of* battery charges *is*

$\mathbb{B}_K = [0, K] \cup \{-\infty\}$. *Thereby K referes to a fully charged battery, 0 to an empty battery and $-\infty$ to the case, that the battery was overstrained.*

There is just a slight difference between states 0 and $-\infty$ by the given description, but $-\infty$ is used as a garbage state in routing here. If at state 0 recharging the battery by recuperation is still possible. If at state $-\infty$ it does not matter which real number representing recuperation is added to it, the battery still remains overstrained.
Throughout this thesis, the same battery capacity level K is used, so $K \in \mathbb{R}_{\geq 0}$ is omitted for better readability.

Definition 3.21. *Let K be a battery capacity and $\mathbb{B} = \mathbb{B}_K$ be a set of battery charges. The set SBTF $=$ SBTF$_K$ of* simple[5] battery transformation functions *comprises functions $f_\infty : \mathbb{B} \to \mathbb{B}$ and $f_{a,b,c} : \mathbb{B} \to \mathbb{B}$ with $f_\infty(J) = -\infty$ for all $J \in \mathbb{B}$ and*

$$f_{a,b,c}(J) = \begin{cases} -\infty & \textit{for } J = -\infty \textit{ or } J \in [0, a), \\ J - c & \textit{for } J \in [a, b), \\ b - c & \textit{for } J \in [b, K], \end{cases}$$

for $c \in [-K, K]$, $a \in [0, \min(K + c, K)]$ and $b \in [\max(0, c), K]$.
Furthermore, the transformation $\tau_{SBTF} : \mathbb{R} \to$ SBTF projects

- *any $c \in [0, K]$ to the function $f_{c,K,c}$,*
- *any $c \in [-K, 0]$ to the function $f_{0,K+c,c}$,*
- *any $c > K$ to the function f_∞ and*
- *any $c < -K$ to the function $f_{0,0,-K}$.*

[5]These functions are called *simple* because combinations of SBTF yielding advanced battery transformation functions are introduced later. These functions will then be used as an implementation for optimal policies.

There are mainly two ways to illustrate simple battery transformation functions. The first one is as usual in a Cartesian coordinate system as can be seen in Figure 3.2, the other one is a mapping from domain to codomain as can be seen in Figure 3.3. The former is suited to illustrate comparisons, the latter is suited to illustrate functional compositions.

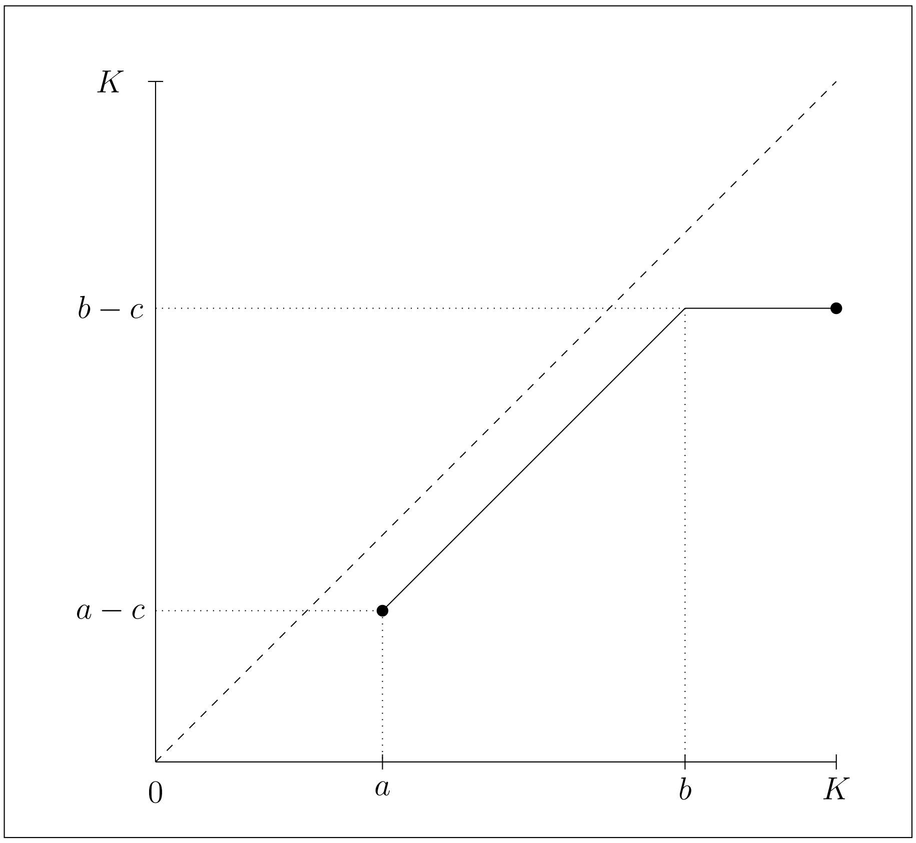


Figure 3.2: A simple battery transformation function $f_{a,b,c}$ is depicted in a Cartesian coordinate system. The value $-\infty$ of the ordinate is omitted for simplicity, all values (strictly) less than a point to $-\infty$.

The battery charge can be increased by applying an SBTF with negative costs $c < 0$, so these functions may be monotone, but not extensive. Therefore, the potential energy is combined with the current battery charge as one state $S = \mathbb{B} \times \mathbb{R}$. The transformation functions then are given by the following definition.

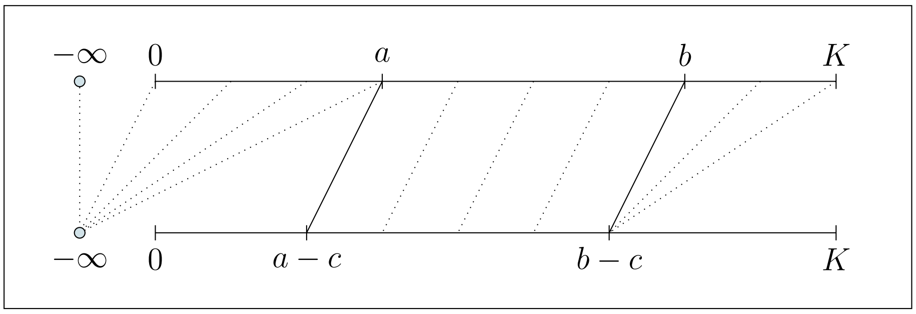


Figure 3.3: A simple battery transformation function $f_{a,b,c}$ is depicted as a mapping. The upper part illustrates the domain, the lower part illustrates the codomain. The dotted lines representing the projection of particular battery charges as well as the negative infinities will be omitted in the following figures.

Definition 3.22. *Let K be a battery capacity and $\mathbb{B} = \mathbb{B}_K$ be a set of battery charges. Let $S = \mathbb{B} \times \mathbb{R}$ where the first value represents the battery charge and the second value represents the potential energy or level of altitude. The set SETF = SETF$_K$ of* simple energy transformation functions *comprises functions $w_{h,f,h'} : \mathbb{B} \times \{h\} \to \mathbb{B} \times \{h'\}$ for all $h, h' \in \mathbb{R}$ and $f \in$ SBTF with*

$$w_{h,f,h'}(J, h) \mapsto (f(J), h'),$$

such that if $f = f_{a,b,c}$ we have

$$c \geq h' - h.$$

Furthermore, the transformation $\tau_{SETF} : \mathbb{R} \times \mathbb{R}_{\geq 0} \times \mathbb{R} \to$ SETF projects any (h, c', h') to the function $w_{h,\tau_{SBTF}(c'+h'-h),h'} \in$ SETF.

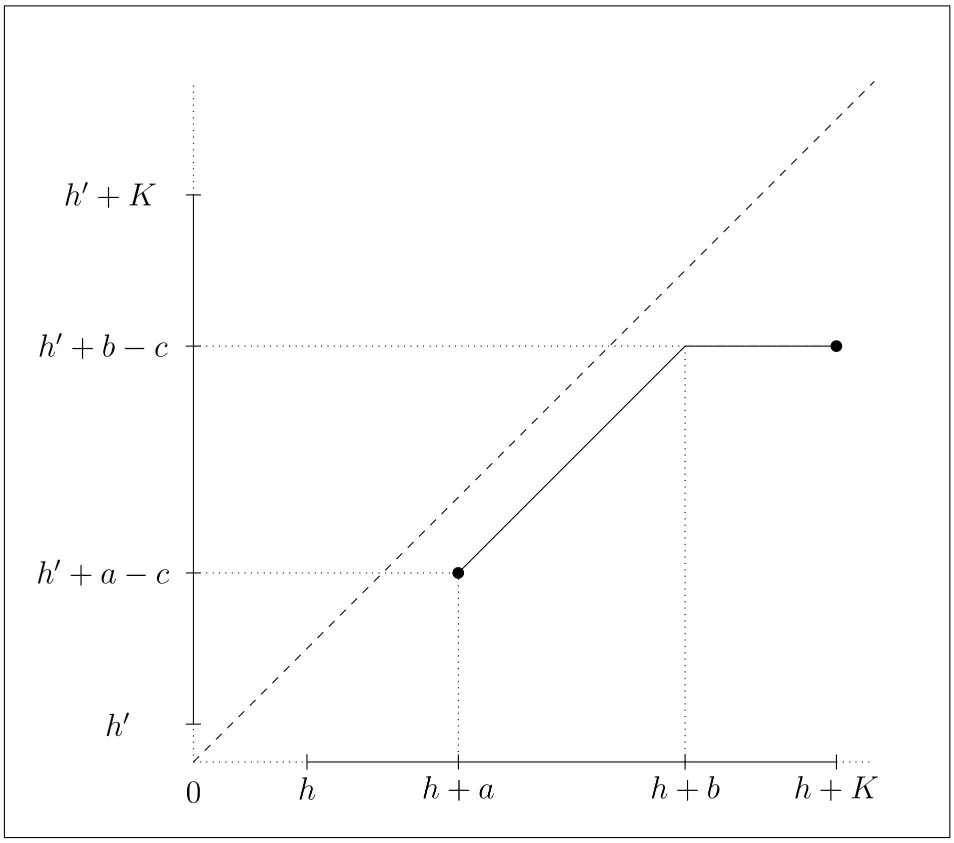


Figure 3.4: A simple energy transformation function $w_{h,f_{a,b,c},h'}$ is depicted in a Cartesian coordinate system. The sum $J + h$ of a state $(J + h) \in \mathbb{B} \times \mathbb{R}$ is used here. Again, all states strictly less than $h + a$ point to $-\infty$. Notice that the graph is always below or on the dashed line for SETFs, because of the constraint $c \geq h' - h$ of Definition 3.22. It is proven in Proposition 3.26.

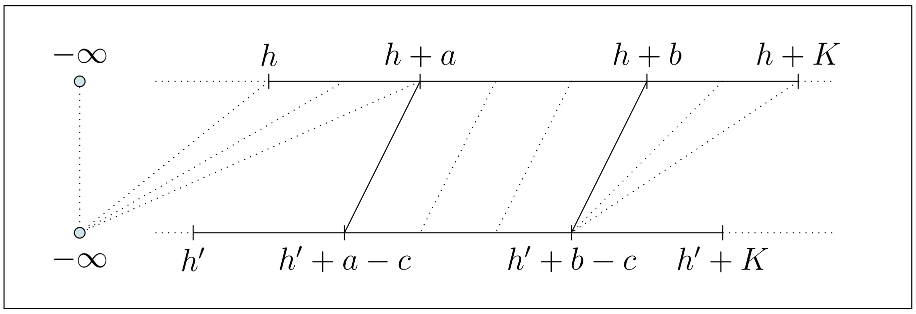


Figure 3.5: A simple energy transformation function $w_{h,f_{a,b,c},h'}$ is depicted. The upper part illustrates the domain, the lower part illustrates the codomain. The sum $J+h$ of a state $(J,h) \in B_K \times \mathbb{R}$ is used here. The battery intervals are translated by their respective potentials h and h'. Notice that all projections lean to the left, i.e. the projection always decreases values describing an energy loss in all cases. This is due to the constraint $c \geq h' - h$ of Definition 3.22 and will be proven in Proposition 3.26.

Remarks.

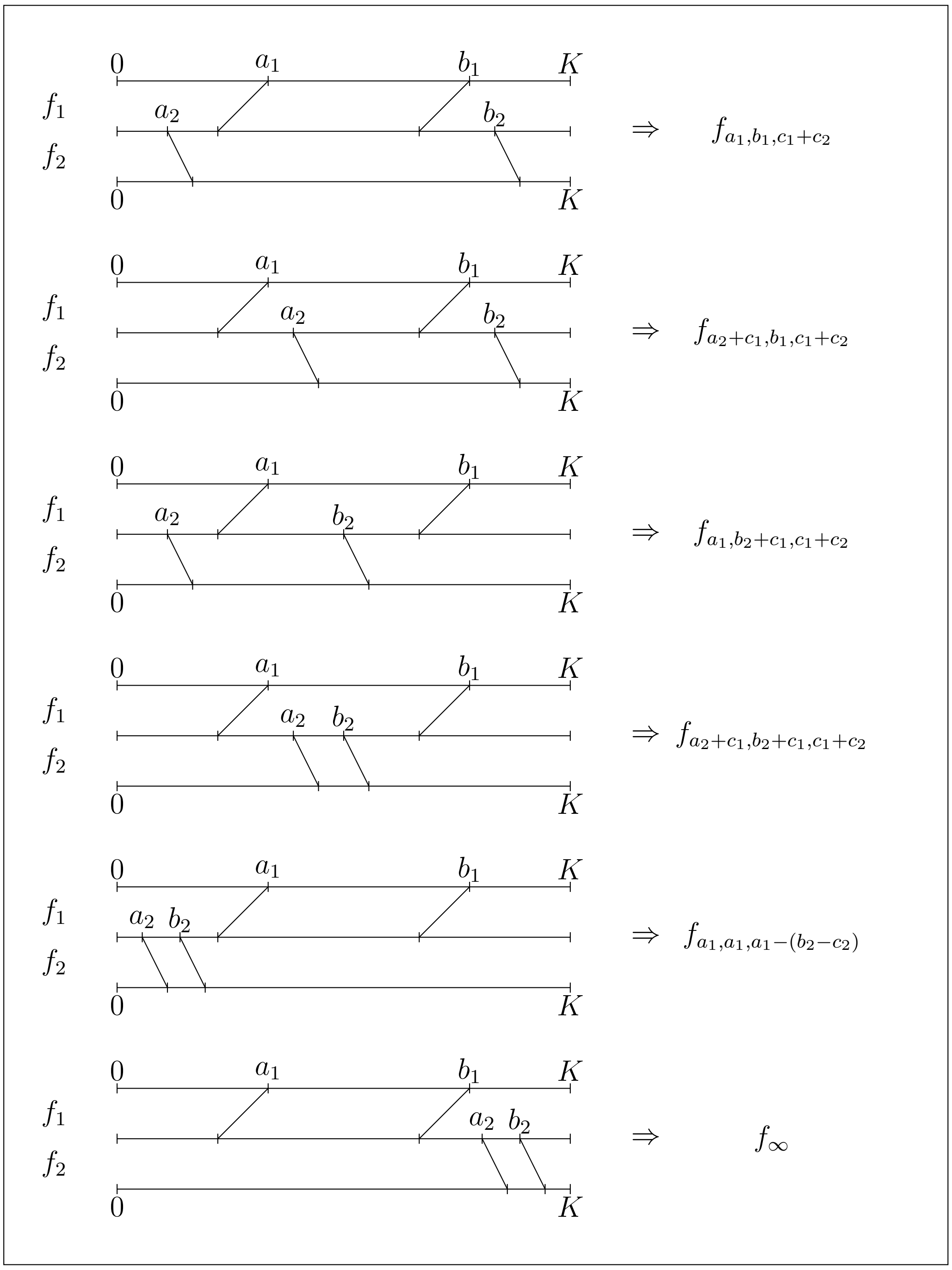
- The inequality $c \geq h' - h$ is required for extensivity and does not violate the transformation τ_{SETF}, because given a $c' \in \mathbb{R}_{\geq 0}$, we have $c = c' + h' - h \geq h' - h$. This is the equivalent of potential functions used in [SLAH11]. In other words, the costs in battery charge must be at least as big as the gain in potential energy.

- Again for illustration purposes, simple energy transformation functions may be depicted as shown in Figure 3.4 and 3.5.

For completeness the closure of SBTF and SETF under functional composition is shown.

Proposition 3.23. *SBTF and SETF are closed under functional composition.*

Proof. Given two functions $f_1, f_2 \in$ SBTF. If either is equal to f_∞, then $f_1 \circ f_2 = f_\infty$ is an SBTF. Otherwise, we have $c_i \in [-K, K]$, $a_i \in [0, \min(K + c_i, K)]$ and $b_i \in [\max(0, c_i), K]$ for $i \in \{1, 2\}$, such that $f_i = f_{a_i,b_i,c_i}$. If now $b_1 - c_1 < a_2$, then $f_1 \circ f_2 = f_\infty$ again is an SBTF. If $a_1 - c_1 \geq b_2$, then $f_1 \circ f_2 = f_{a_1,a_1,c}$ with



0
a_1
b_1
K
f_1
a_2
b_2
f_2
0
K
$\Rightarrow$ f_{a_1,b_1,c_1+c_2}
$\Rightarrow$ $f_{a_2+c_1,b_1,c_1+c_2}$
$\Rightarrow$ f_{a_1,b_2+c_1,c_1+c_2}
$\Rightarrow$ $f_{a_2+c_1,b_2+c_1,c_1+c_2}$
$\Rightarrow$ $f_{a_1,a_1,a_1-(b_2-c_2)}$
$\Rightarrow$ f_∞

Figure 3.6: Composition of two SBTFs: If both $f_1 \neq f_\infty$ and $f_2 \neq f_\infty$, then there may be six cases specifying the composition $f_1 \circ f_2$. In any case the composition yields an SBTF. Therefore, simple battery transformation functions are closed under functional composition.

$c = a_1 - f_2(f_1(a_1)) = a_1 - (b_2 - c_2)$ being an SBTF. Otherwise, we have $a_2 \leq b_1 - c_1$ and $b_2 < a_1 - c_1$, and the composition yields $f_{a,b,c} = f_1 \circ f_2$ with $c = c_1 + c_2$, $a = \max(a_1, a_2 + c_1)$ and $b = \min(b_1, b_2 + c_1)$, which is an SBTF (see Figure 3.6 for illustration). Therefore, SBTF is closed under functional composition.

Now composing two SETF, say $w_1 : \mathbb{B} \times \{h\} \to \mathbb{B} \times \{h'\}$, $w_2 : \mathbb{B} \times \{h'\} \to \mathbb{B} \times \{h''\}$, with $w_1 = w_{h,f_1,h'} \in$ SETF and $w_2 = w_{h',f_2,h''} \in$ SETF yields again an SETF $w : \mathbb{B} \times \{h\} \to \mathbb{B} \times \{h''\}$, namely $w = w_1 \circ w_2 = w_{h,f_1 \circ f_2,h''}$.

Furthermore, we need to check wether the constraint $c \geq h' - h$ for any $w_{h,f_{a,b,c},h'}$ is also valid after composition: If $f_i = f_{a_i,b_i,c_i}$ for $i \in \{1,2\}$, then $c_1 \geq h' - h$ and $c_2 \geq h'' - h'$, so if $f_{a,b,c} = f_1 \circ f_2$ (i.e. $f_1 \circ f_2 \neq f_\infty$), then

$$c = c_1 + c_2 \qquad \text{for} \quad a_1 - c_1 < b_2$$

or

$$c = a_1 - (b_2 - c_2) \geq c_1 + c_2 \qquad \text{for} \quad a_1 - c_1 \geq b_2.$$

In any case we have

$$c \geq c_1 + c_2 \geq h' - h + h'' - h' = h'' - h.$$

Therefore, the constraint $c \geq h' - h$ is preserved under functional composition.

□

The actual problem in energy-optimal routing is that SETF is not closed under the $\max$ operator. Otherwise routing would be as simple as SPP. Figure 3.7 shows that the maximization yields not an SETF in general. In Section 4.3 we will generalize these transformation functions by using the closure under the $\max$ operator.

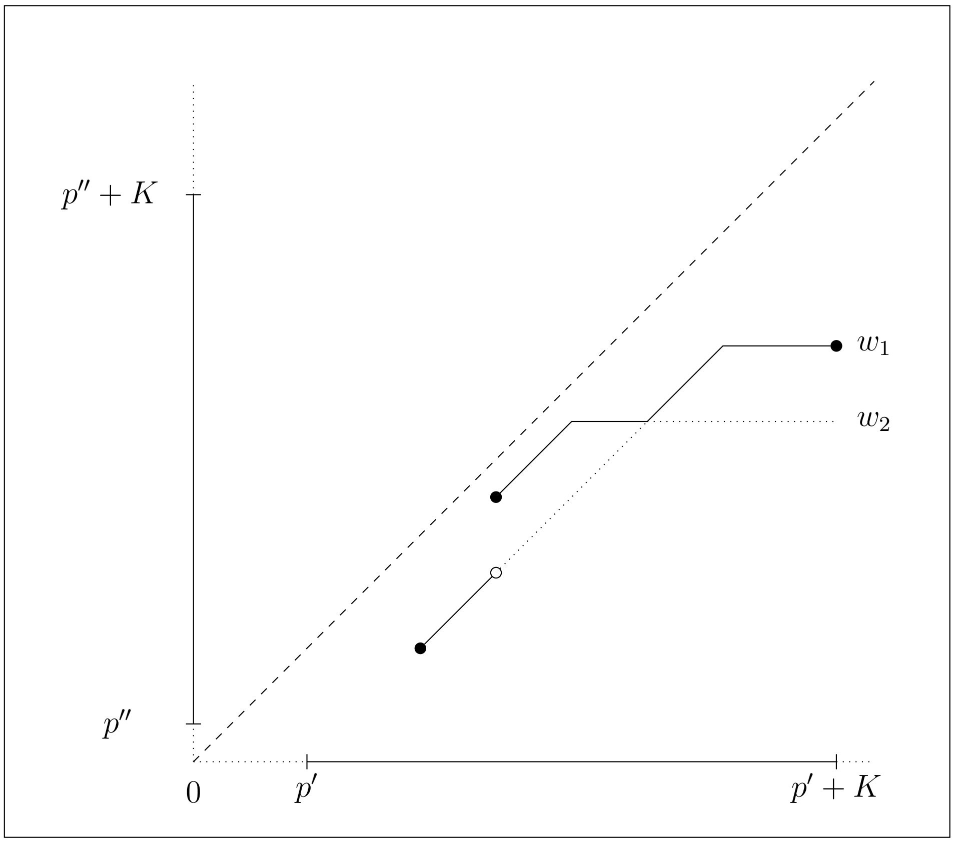


Figure 3.7: The maximum of two simple energy transformation functions w_1, w_2 may not yield an SETF again. Therefore, we need to keep track of multiple paths during routing. A *policy* decides, which path is optimal among a set of paths with corresponding SETFs. The closure of SETF under the $\max$ operator will be discussed in the next chapter.

3.5.2 State-Based Energy-Optimality

Now with Definition 3.22 the energy-optimal routing can be specified as follows.

Definition 3.24. *Let $G = (V, E)$ be a graph with potentials $H : V \to \mathbb{R}$ and edge costs $C : E \to \mathbb{R}_{\geq 0}$. The state space is $S = \mathbb{B} \times \mathbb{R}$ with the total preorder $\leq_S$ induced by the function $(J, h) \mapsto J + h$ with total order $\geq$ on $\overline{\mathbb{R}}$ (because we want to* maximize *the energy), and let $\mathcal{S}(v) = \mathbb{B} \times \{H(v)\}$ for each $v \in V$. With $\mathbb{W} =$ SETF the edge weights are given by $\mathbb{W}_e = \tau_{SETF}(\{H(x)\} \times \mathbb{R}_{\geq 0} \times \{H(y)\})$ and a weighting is given by $\mathcal{W}'(e) = \tau_{SETF}(H(x), c, H(y))$ for $e = (x, y) \in E$.*
Given vertices $x, y \in V$ and an initial battery charge $J \in \mathbb{B}$ the energy-optimal path problem (EOPP2) *is to solve the described SBSPP for $(G, S, \leq_S, \mathcal{S}, \mathbb{W}, \mathcal{W}')$.*

The monotonicity of weights is guaranteed:

Proposition 3.25. *The weight functions $\mathbb{W}$ of Definition 3.24 are monotone with respect to $\leq_S$ on $\mathbb{B}_K \times \mathbb{R}$.*

Proof. Simple battery transformation functions (SBTFs) are monotone: The function $f_\infty \in$ SBTF is monotone, because it is constant. Let $f_{a,b,c} \in$ SBTF and let $J_1, J_2 \in \mathbb{B}$, $J_1 \leq J_2$, then

- If $J_1 = -\infty$ or $J_1 \in [0, a)$, then $f(J_1) = -\infty \leq f(J_2)$ for all $J_2 \in \mathbb{B}_K$.
- If $J_1 \in [a, b)$, then $f(J_1) = J_1 - c$. Either $J_2 \in [a, b)$, then $f(J_1) = J_1 - c \leq J_2 - c = f(J_2)$, or else $J_2 \in [b, K]$, then $f(J_1) = J_1 - c \leq b - c = f(J_2)$.
- If $J_1 \in [b, K]$, then also $J_2 \in (b, K]$ and $f(J_1) = b - c = f(J_2)$.

So in any case, if $J_1 \leq J_2$ then $f(J_1) \leq f(J_2)$.
Now given a weight function $w = w_{h,f,h'} : \mathbb{B} \cup \{h\} \to \mathbb{B} \cup \{h'\}$, $w \in$ SETF, then two states $s_1, s_2 \in \mathbb{B} \cup \{h\}$ are described by $s_1 = (J_1, h)$ and $s_2 = (J_2, h)$, $J_1, J_2 \in \mathbb{B}$. Assume that $s_1 \leq_S s_2$, then $J_1 + h \geq J_2 + h$, i.e. $J_1 \geq J_2$. Since

$f \in$ SBTF is monotone, we have $f(J_1) \geq f(J_2)$ and $f(J_1) + h' \geq f(J_2) + h'$, i.e. $w(J_1, h) = (f(J_1), h') \leq_S (f(J_2), h') = w(J_2, h)$.

□

Furthermore, the weights are extensive, meaning that we can not gain energy by traversing edges with respect to the sum of battery charge and potential energy:

Proposition 3.26. *The weight functions $\mathbb{W}$ of Definition 3.24 are extensive with respect to $\leq_S$.*

Proof. Let $h, h' \in \mathbb{R}$, then with $w = w_{h,\infty,h'}$, we have $w(J, h) = (-\infty, h')$ constantly, therefore w is extensive. Otherwise let $c \in \mathbb{R}_{\geq 0}$, $h, h' \in \mathbb{R}$ and $w = w_{h,f,h'}$, $w : \mathbb{B} \cup \{h\} \to \mathbb{B} \cup \{h'\}$ with $f = f_{a,b,c} \in$ SBTF with $c \geq h' - h$, then $J \geq J - (c + h' - h) \geq f(J)$ following $J + h' \geq f(J) + h'$ and $(J, h) \leq_S (f(J), h') = w(J, h)$ for all states $(J, h) \in \mathbb{B} \cup \{h\}$.
In detail this is:

- If $J = -\infty$ or $J \in [0, a)$, then $f(J) = -\infty$, so $J + h \geq f(J) + h'$ and $(J, h) \leq_S (f(J), h')$.
- If $J \in [a, b)$, then $f(J) = J - (c + h' - h) = (J + h) - (c + h')$, so with $c \geq 0$ (by definition $c \in \mathbb{R}_{\geq 0}$) we have $J + h \geq J + h - c = (J + h) - (c + h') + h' = f(J) + h'$ and $(J, h) \leq_S (f(J), h)$.
- If $J \in [b, K]$, then $f(J) = K$ and $J - (c + h' - h) \geq K$, because $J \geq b = K + c + h' - h$. Therefore, $J + h \geq K + c + h' \geq K + h' = f(J) + h'$, so we have $(J, h) \leq_S (f(J), h)$.

In all of these cases we have seen, that $(J, h) \leq_S (f(J), h') = w(J, h)$. Therefore, SETFs are extensive.

□

The following proves, that both definitions of energy-optimal routing are equivalent.

Theorem 3.27. *Both definitions of the enery-optimal path problem, EOPP and EOPP2, are equivalent, insofar as EOPP2 may be reduced to EOPP instantly, while EOPP may be reduced to EOPP2 by computing a potential function $H : V \to \mathbb{R}$.*

Proof. To prove this theorem, it needs to be shown, that given an initial battery charge $J \in [0, K]$ and two vertices $x, y \in V$, the energy costs of a walk (actually we only need to show this for paths) of Definition 3.18 are equivalent to those given by the corresponding energy transformation functions of Definition 3.22.

Given a graph $G = (V, E)$ and an edge cost function $C : E \to \mathbb{R}$, such that there are no cycles of negative energy costs, then there exists a potential function (Mehlhorn et al. in [MS10]) $H : V \to \mathbb{R}$, such that an optimal path with edge costs C is also an optimal path with respect to edge costs $C'(e) = C(e) - H(x) + H(y) \geq 0$ for all edges $e = (x, y) \in E$ and vice versa.

Now given a walk $\gamma = (v_0, \ldots, v_k) \in \Gamma_{x,y}$ the energy costs are defined recursively:

$$C_{KJ}(\gamma^i) = \begin{cases} K - J & \text{if } i = 0, \\ 0 & \text{if } i > 0 \text{ and } \Delta^i < 0, \\ \Delta^i & \text{if } i > 0 \text{ and } 0 \leq \Delta^i \leq K, \\ \infty & \text{if } i > 0 \text{ and } \Delta^i > K \end{cases}$$

where

$$\Delta^i = C_{KJ}(\gamma^{i-1}) + C(v_{i-1}, v_i) \qquad \text{for } i > 0.$$

The corresponding energy transformation function w is given by

$$w = \tau_{\text{SETF}}(H(v_0), C'(v_0, v_1), H(v_1)) \circ \ldots \circ \tau_{\text{SETF}}(H(v_{k-1}), C'(v_{k-1}, v_k), H(v_k)),$$

which by definition gives

$$\begin{aligned}
&= w_{H(v_0),\tau_{\text{SBTF}}(C'(v_0,v_1)+H(v_1)-H(v_0)),H(v_1)} \circ \ldots \circ \\
&\quad w_{H(v_{k-1}),\tau_{\text{SBTF}}(C'(v_{k-1},v_k)+H(v_k)-H(v_{k-1})),H(v_k)} \\
&= w_{H(v_0),\tau_{\text{SBTF}}(C'(v_0,v_1)+H(v_1)-H(v_0))\circ\ldots\circ\tau_{\text{SBTF}}(C'(v_{k-1},v_k)+H(v_k)-H(v_{k-1})),H(v_k)} \\
&= w_{H(v_0),\tau_{\text{SBTF}}(C(v_0,v_1))\circ\ldots\circ\tau_{\text{SBTF}}(C(v_{k-1},v_k)),H(v_k)}
\end{aligned}$$

Because the second parameter is constant, we just need to optimize the first parameter. We will prove now, that maximizing this parameter (as is done by EOPP2) minimizes energy costs. For convenience, let

$$f_k := \tau_{\text{SBTF}}(C(v_0, v_1)) \circ \ldots \circ \tau_{\text{SBTF}}(C(v_{k-1}, v_k))$$

Namely we will prove by induction on the length k of walk γ, that

$$C_{KJ}(\gamma^k) = K - f_k(J).$$

Base Let $k = 0$, i.e. $\gamma = (v_0)$. The energy costs are given by $C_{KJ}(\gamma) = K - J$, whereas the battery transformation function is the identity function (because γ^0 is an empty path). Therefore, $C_{KJ}(\gamma^0) = K - J = K - \text{id}_{\mathbb{B}}(J) = K - f_0(J)$.

Hypothesis For some $0 \leq i < k$ and all $J \in \mathbb{B}$, let $C_{KJ}(\gamma^i) = K - f_i(J)$.

Step We have $\Delta^{i+1} = C_{KJ}(\gamma^i) + C(v_i, v_{i+1})$. By case distinction:

- If $\Delta^{i+1} > K$, then $C_{KJ}(\gamma^{i+1}) = \infty$. Furthermore, we have $f_i(J) = K -$

$C_{KJ}(\gamma^i)$ by induction hypothesis. Therefore,

$$\begin{aligned} f_{i+1}(J) &= (f_i \circ \tau_{\text{SBTF}}(C(v_i, v_{i+1})))(J) \\ &= \tau_{\text{SBTF}}(C(v_i, v_{i+1}))(f_i(J)) \\ &= \tau_{\text{SBTF}}(C(v_i, v_{i+1}))(K - C_{KJ}(\gamma^i)) \\ &= -\infty, \end{aligned}$$

because

$$K - C_{KJ}(\gamma^i) - C(v_i, v_{i+1}) = K - \Delta^{i+1} < 0.$$

- If $\Delta^{i+1} < 0$, then $C_{KJ}(\gamma^{i+1}) = 0$. Furthermore, we have $f_i(J) = K - C_{KJ}(\gamma^i)$ by induction hypothesis. Therefore,

$$f_{i+1}(J) = \ldots = \tau_{\text{SBTF}}(C(v_i, v_{i+1}))(K - C_{KJ}(\gamma^i)) = K,$$

because

$$K - C_{KJ}(\gamma^i) - C(v_i, v_{i+1}) = K - \Delta^{i+1} > K.$$

- Otherwise, if $\Delta^{i+1} \in [0, K]$, then $C_{KJ}(\gamma^{i+1}) = \Delta^{i+1}$. Furthermore, we have $f_i(J) = K - C_{KJ}(\gamma^i)$ by induction hypothesis. Therefore,

$$f_{i+1}(J) = \ldots = \tau_{\text{SBTF}}(C(v_i, v_{i+1}))(K - C_{KJ}(\gamma^i)) = K - \Delta^{i+1},$$

because

$$K - C_{KJ}(\gamma^i) - C(v_i, v_{i+1}) = K - \Delta^{i+1} \in [0, K].$$

Therefore, in all cases we have $C_{KJ}(\gamma^{i+1}) = K - f_{i+1}(J)$.

Finally, since both cost functions are equivalent, insofar that EOPP minimizes costs, whereas EOPP2 maximizes energy left after traversal, both models query for the same paths. Therefore, they are equivalent except for the potential function, that is explicitely required in EOPP2, but only indirectly given in EOPP.

$\square$

This shows that the energy-optimal routing problem may be redefined from a state-based perspective. It remains to show that this problem is simple with respect to Definition 3.11 and 3.12. The first one, refering to the optimal path problem, is easily proven as follows. The second one depends particularly on the implementation, which is why the simplicity of energy-optimal profile routing will be discussed in the next chapter.

Proposition 3.28. *The energy-optimal path problem EOPP2 is simple with respect to Definition 3.11.*

Proof. We need to check three statements:

1. The preorder $\leq_S$ is total.

   Since $\leq_S$ is induced by the total order $\geq$ on the sum of both components, the induced preorder is also total.

2. The states S must have constant descriptive complexity and may be compared by $\leq_S$ in constant time.

   The descriptive complexity of a state $s \in S = \mathbb{B} \times \mathbb{R}$ is constant, because its representation just requires two numbers. They are compared by using $\geq$ on the sum of their two components, which is done in constant time.

3. The edge weights functions $\mathbb{W}_e$ have constant descriptive complexity and may be computed in constant time

   Weight functions are energy transformation functions $\mathbb{W} =$ SETF, which can be described by two numbers h, h' and a function $f \in$ SBTF, which itself can be described either by f_∞ or by $f_{a,b,c}$ with three numbers a, b, c. Therefore, the descriptive complexity is constant.

The computation of a weight function $w \in$ SETF for a given argument $(J, h) \in \mathbb{B} \times \{h\}$ may be computed in constant time, because it comprises just an SBTF with a (constantly sized) case distinction.

□

3.6 Shortest Weight-Constrained Path Problem

The driver may be interested in minimizing energy use for Green Routing while having constraints to the time needed to reach a destination or to the traveling distance. Without those constraints driving directions will probably lead through slow-paced regions, because driving fast is less energy-efficient. The corresponding model is described by Joksch in [Jok66] and its decision version is listed in [GJ90] as [ND30]:

Definition 3.29. *Given a graph $G = (V, E)$, additive edge lengths $L : E \to \mathbb{Z}_+$ and additive edge weights $W : E \to \mathbb{Z}_+$, vertices $x, y \in V$, maximum length $L_{\max} \in \mathbb{Z}_+$ and maximum weight $W_{\max} \in \mathbb{Z}_+$, the* shortest weight-constrained path problem *(SWCPP) is to determine, wether there is a path from x to y in G with total length $L_{\max}$ or less and total weight $W_{\max}$ or less?* [6]

3.6.1 Intractability

Unfortunately the NP-complete problem *Partition* can be reduced to this path problem. It is one of Karp's 21 NP-complete problems [Kar72] and defined as follows:

[6]Garey in [GJ90], p. 214, slightly changed for consistency.

Definition 3.30. *Given a tuple $(c_1, c_2, \dots, c_k) \in \mathbb{Z}^k$, the* partition problem *(PP) is to determine, wether there is a set $I \subseteq \{1, \dots, k\}$ such that $\sum_{i \in I} c_i = \sum_{i \notin I} c_i$?* [7]

For the sake of completeness a reduction is given in the proof of the following theorem.

Theorem 3.31. *The decision version of the shortest weight-constrained path problem is NP-complete.*

Proof. Obviously SWCPP is an element of NP, because given a nondeterministic solution, a path π^*_{xy} from x to y, it is easy to verify, if the given total limits $L_{\max}$ and $W_{\max}$ are satisfied.
To show the NP-hardness, we reduce PP to SWCPP in polynomial time: Given an instance of PP, a tuple $(c_1, c_2, \dots, c_k) \in \mathbb{Z}^k$ with minimum value[8] $c_{\min} = \min_{i=1,\dots,k} c_i$, the instance of SWCPP is given by $G = (V, E)$ with

$$\begin{aligned}
V &:= \left\{v_0, v_1^l, \dots, v_k^l, v_1^w, \dots, v_k^w, v_1, \dots, v_k\right\}, \\
E &:= \left\{\left\{v_i^l, v_i\right\}, \left\{v_i^w, v_i\right\}, \left\{v_{i-1}, v_i^l\right\}, \left\{v_{i-1}, v_i^w\right\} \mid i = 1, \dots, k\right\}, \\
L(e) &:= \begin{cases} 1 + c_i - c_{\min} & \text{if } v_i^l \in e, \\ 1 - c_{\min} & \text{otherwise,} \end{cases} \\
W(e) &:= \begin{cases} 1 + c_i - c_{\min} & \text{if } v_i^w \in e, \\ 1 - c_{\min} & \text{otherwise,} \end{cases} \\
L_{\max} &:= W_{\max} := 2k \cdot (1 - c_{\min}) + \sum_{i=1}^{k} c_k, \quad x := v_0 \quad \text{and} \quad y := v_k.
\end{aligned}$$

By definition a path does not visit any vertex twice. Therefore, any path π_{xy} is of the form $(v_0, v_1^{\alpha_1}, v_1, \dots, v_k^{\alpha_k}, v_k)$ with α_i being either l or w for $i = 1, \dots, k$. The

[7][Kar72], p. 97, slightly changed for consistency.
[8]The only reason to use the minimum value of all c_i is, that PP is defined on integers, while SWCPP is defined on positive integers. Therefore, we need $c_{\min}$ to translate the values into the space of natural numbers.

index set is defined as $I := \{i \in \{1, \dots, k\} \mid \alpha_i = l\}$. The length of the path then is

$$\begin{aligned} L(\pi_{xy}) &= \sum_{i=1}^{k} L(v_{i-1}, v_i^{\alpha_i}) + L(v_i^{\alpha_i}, v_i) = 2 \cdot \sum_{i=1}^{k} L(v_i, v_i^{\alpha_i}) \\ &= 2k \cdot (1 - c_{\text{min}}) + 2 \cdot \sum_{v_i^{\alpha_i} = v_i^l} c_i = 2k \cdot (1 - c_{\text{min}}) + 2 \cdot \sum_{i \in I} c_i. \end{aligned}$$

The weight of the path similarly is

$$W(\pi_{xy}) = 2k \cdot (1 - c_{\text{min}}) + 2 \cdot \sum_{i \notin I} c_i.$$

Notice, that

$$L(\pi_{xy}) + W(\pi_{xy}) = 4k \cdot (1 - c_{\text{min}}) + 2 \cdot \sum_{i=1}^{k} c_i = L_{\max} + W_{\max},$$

i.e. the sum of length and weight of any path is constant. Therefore, with $L(\pi_{xy}) \leq L_{\max}$ and $W(\pi_{xy}) \leq W_{\max}$ it follows $L(\pi_{xy}) = L_{\max} = W_{\max} = W(\pi_{xy})$ and

$$\sum_{i \in I} c_i = \frac{L(\pi_{xy})}{2} - k \cdot (1 - c_{\text{min}}) = \frac{W(\pi_{xy})}{2} - k \cdot (1 - c_{\text{min}}) = \sum_{i \notin I} c_i.$$

Thus we know:

- If there is a path found in SWCPP, then there is a valid partition.

- If there is a valid solution $I \subseteq \{1, \dots, k\}$, then I induces a solution to the constructed instance of SWCPP.

$\square$

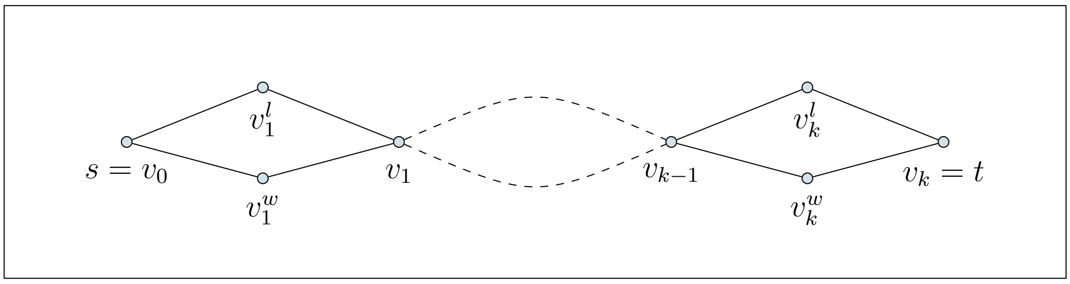


Figure 3.8: Reduction of *Partition* to SWCPP

3.6.2 State-Based Perspective

The two previously presented models, namely the time-dependent and the energy-optimal routing problem, were both described positively as state-based problems. The NP-complete weight-constrained path problem may as well be described as an SBSPP, but one with partially preordered states. Since the states are partially preordered, we need to keep record of all minimal states found during routing. The number of optimal paths, and thus the size of an optimal policy, may grow exponentially in the number of edges.

The intuitive definition of the state-space of SWCPP would be $S = \mathbb{Z}_+^2 \cup \{(0,0)\}$, representing both the distance and the weight from the start to some vertex. Let $H : E \to \mathbb{Z}_+$ and $L : E \to \mathbb{Z}_+$ denote the weighting (because W will be used for edge weight functions) and the length of edges, as well as $H_{\max}$ and $L_{\max}$ the maximum weight and maximum length. The weight functions are $w_{h,l} \in W'$ with $w_{h,l} : S \to S$ and $w_{h,l}(h', l') = (h' + h, l' + l)$. The weighting $\mathcal{W}' : E \to W'$ is given by $\mathcal{W}'(e)(h, l) = w_{H(e),L(e)}$.

Now the definition of $\leq_S$ is given by $(h_1, l_1) \leq_S (h_2, l_2)$, if and only if $h_1 \leq h_2 \wedge l_1 \leq l_2$ or $h_2 > H_{\max} \vee l_2 > L_{\max}$, such that all states violating the constraints are equivalent and greater than non-violating states. This relation is reflexive and transitive, but neither total nor symmetric.

The problem resulting from the NP-completeness is the number of minimal paths, which may grow exponentially with respect to $|V|$. This can be seen in Figure 3.9 using binary representation, such that the cardinality of $\min(W(\Pi(x, y)))$ is equal to $\left|\left\{(z, (2^k - 1) - z) \mid 0 \leq z < 2^k\right\}\right| = 2^k$.

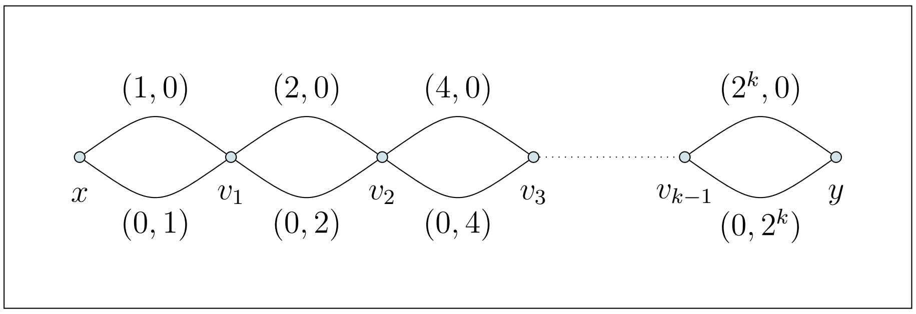


Figure 3.9: Given $k+1$ vertices (with $2k$ vertices inbetween), the number of minimal paths with respect to $\leq_S$ and $H_{\max}, L_{\max} > 2^k$ grows exponentially, because each weighting $(z, (2^{k+1} - 1) - z)$ for $0 \leq z < 2^{k+1}$ is described by one path through the graph. All of these weightings are minimal, because there is no other weighting with $z' \leq z$ and $w' = (2^{k+1} - 1) - z' \leq (2^{k+1} - 1) - z = w$ at the same time.

Nevertheless the corresponding SBPRP resulting from the state-based perspective of weight-constrained path finding may still be useful. There are policies $m \subseteq \Pi_{x,y}$, such that $\pi \in \Pi_{x,y}$ with $\mathcal{W}(\pi) = w_{h,l}$ improves m, if and only if for all paths $\pi' \in m$ with $\mathcal{W}(\pi') = w_{h',l'}$ we have $h < h'$ or $l < l'$ as well as $h \leq H_{\max}$ and $l \leq L_{\max}$. So, even though this problem is intractable, all algorithms developed for SBPRP may still be applied and may probably still yield a better runtime on real road networks.

3.7 Stochastic Shortest Path Problems

Uludag et al. [UUN+09] investigated a stochastic routing model looking for a path that satisfies a given constraint with highest probability. This was done in the generic context of Quality-of-Service routing, so it may possibly be adopted to use for vehicle routing. Instead of a deterministic edge weighting function, in-

dependent random variables are used as edge weights. These random variables could represent the time needed to take a connection. Different acceleration and deceleration behaviours induce randomness. Furthermore, we can use congestion predictions or traffic lights in a stochastic manner.
The following model is adapted from [UUN$^+$09], but since it does not actually compute a shortest path, but a most likely successful path, it is called *most likely successful* routing within this thesis.
Of course you may argue, that successful paths must also be shortest paths, but those two aspects are just indirectly connected. You could rather say, that these aspects behave similar to Pareto optimality on the success and the expected length.

Definition 3.32. *Given a graph $G = (V, E)$, vertices $x, y \in V$, a probability space $(\Omega, \mathcal{F}, P)$, additive, independent random variables with known cumulative distribution functions $C : E \to (\Omega \to \mathbb{R}_{\geq 0})$ representing edge costs and maximum costs $C_{\max} \in \mathbb{R}_{\geq 0}$, the* most likely successful path problem (MLSPP) *is to find a path $\pi^*_{x,y}$ from x to y in G satisfying $C_{\max}$ with maximum probability, if it exists.* [9]

The problem may also be described as

$$\pi^*_{x,y} \in \operatorname*{arg\,max}_{\pi_{x,y} \in \Pi_{x,y}} P(C(\pi_{x,y}) \leq C_{\max}).$$

This model however is not complete, as we do not intuitively know how to represent the cumulative distribution functions. By their given algorithm Uludag et al. indirectly suggest to use observation points on the Laplace transformation of the cumulative distribution function. These are evaluated by a modified Gaussian quadrature. Obviously there are various interesting numerical problems involved this way.
The NP-hardness of this problem was shown by Guérin et al. in [GO99], who

[9]Uludag et al. in [UUN$^+$09], slightly changed for consistency. The costs are explicitely defined to be non-negative here.

investigated stochastic routing in the context of computer networks. Furthermore, they have shown, that determining if $P(C(\pi_{x,y}) \leq C_{\max}) \geq P_{\min}$ for a given path $\pi_{x,y}$ and some $P_{\min} \in [0,1]$ is NP-hard by reduction from the K-th largest subset problem listed in [GJ90] as problem [SP20]. It is not even known wether [SP20] is in NP or not.

As described in the previous model, one could search for a path, that fulfills a given constraint with maximum probability. Probably the more intuitive problem for stochastic routing would be to search for a path with least expected costs.

Definition 3.33. *Given a graph $G = (V, E)$, vertices $x, y \in V$, a probability space $(\Omega, \mathcal{F}, P)$ and additive, independent random variables with known cumulative distribution functions $C : E \to (\Omega \to \mathbb{R}_{\geq 0})$ representing edge costs, the* expected shortest path problem (ESPP) *is to find a path $\pi^*_{x,y}$ in G from x to y with least expected costs, if it exists.*

The problem may also be described as

$$\pi^*_{x,y} \in \operatorname*{arg\,min}_{\pi_{x,y} \in \Pi_{x,y}} \mathbb{E}(C(\pi_{x,y})).$$

Because by choosing a different weighting function $C' : E \to \mathbb{R}_{\geq 0}$ with $C' = C \circ \mathbb{E}$ an optimal solution to the expected shortest path problem is directly given by solving the corresponding SPP with edge costs C'. This is due to the linearity of the expected value operator.

However, the presented model remains interesting for further extensions. For example finding a solution turns out to be difficult as soon as any constraints are introduced, such as battery constraints from energy-optimal routing. Furthermore, this model provides the basics for routing with recourse.

3.7.1 State-Based Perspective

This thesis tries to combine both energy-optimality with stochastic aspects. If stochastic routing can be described in a state-based manner, then it can probably be combined with the state-based energy-optimal approach. Therefore, this section handles an one step towards stochastic energy-optimal routing algorithms.

As it has been already noticed, the ESPP can be translated into an SPP, so it is easy to translate it to a state-based model. This section concentrates on the MLSPP. Notice that the state space now contains random variables. In practice one would use the corresponding cumulative distribution functions (CDF) as well as the probability density functions (PDF), if the variables are continuous.

There are some obstacles when formulating the MLSPP as an SBSPP. First, [UUN$^+$09] makes heavy use of a backward approach. This means to have a final state and inverse state transformation functions.

The intuitive approach may be to say that the distance of an empty path (at the initial vertex) is 0 and that random variables are added to this initial state for each edge traversed. When multiple paths meet then those less than one of the remaining are omitted, depending on $\leq_S$. A natural approach for $\leq_S$ is comparing CDFs. Given two states $s_1, s_2 : \Omega \to \mathbb{R}$, then $s_1 \leq_S s_2$ if and only if $P(s_1 \leq x) \geq P(s_2 \leq x)$ for all $x \in \mathbb{R}$. This is the approach used later in this thesis.

The other approach is a backward approach and discussed in [UUN$^+$09]. The presented algorithm starts at the target vertex with the state 0 and uses a pull strategy [10] to update the information for vertices backwards. The states at each vertex are the sum of the variables from traversed edges. At the starting vertex the algorithm chooses iteratively the appropriate successor corresponding to the

[10]The pull strategy does essentially the same thing as the push strategy, which is used by Dijkstra's algorithm. The slight difference is the processing of a node, where you draw the information from predecessors instead of pushing information to the successors. See Sniedovich [Sni06] for details. Notice that in backward searches the predecessors actually are the successors of a vertex and vice versa.

optimal success value.

The latter approach is used in the following in order to stick to the definition of [UUN+09].

The state space consists of random variables and the edge weights are additions of further random variables. For practical reasons convolutions of PDFs (and CDFs with derivation) are used to sum up two random variables. Since convolutions of PDFs are given only for stochastically independent variables, all variables should be chosen to be independent. But requiring all states and edge weights to be independent is not possible, because after the traversal of an edge, the edge weight and the resulting state, i.e. the sum of the previous state and the edge variable, are not (at least not always) independent. So to avoid invalid states, cycles must be avoided explicitely.

Formally, this can be done by adding a list of vertices to each state that have already been visited so far. This would be $S = \{\Omega \to \mathbb{R}\} \times \mathcal{P}(V^*) \cup \{\perp\}$ for some garbage state $\perp$. Then for edge $e = (x, y) \in E$ the weight $w_e : \mathcal{S}(x) \to \mathcal{S}(y)$ is given by $w_e(s) = \perp$ for $s = (X, (v_1, \ldots, v_k)) \in \mathcal{S}(x)$ if and only if y is among the vertices $v_1, \ldots, v_k$. Otherwise, the random variable would be added to the first component and vertex y would be enlisted in the second component.

Because of the extensivity of weights cycles are avoided anyway, so these enriched states are just formally needed. Therefore, stochastic independency of all random variables representing states and edge weights is assumed.

In the previously described problem, MLSPP, a path with the maximum probability of success of having costs less or equal than $C_{\max}$ is queried. So if $P(s_1 \leq C_{\max}) \geq P(s_2 \leq C_{\max})$, then we prefer s_1 over s_2, i.e. $s_1 \leq_S s_2$. But states need to be compared at all times during routing, in order to use all paths that may cause a higher success in the future. An extension to $s_1 \leq_S s_2$ if and only if $P(s_1 \leq x) \geq P(s_2 \leq x)$ for all $x \leq C_{\max}$ is the natural approach also described in [UUN+09]. Obviously, this relation is not total, so the following definition is an

SBSPP with partially preordered states.

Definition 3.34. *Given a graph $G = (V, E)$, a probability space $(\Omega, \mathcal{F}, P)$, independent random variables with known cumulative distribution functions $C : E \to (\Omega \to \mathbb{R}_{\geq 0})$ representing edge costs and maximum costs $C_{\max} \in \mathbb{R}_{\geq 0}$.*
The state space describes random variables $S = \{\Omega \to \mathbb{R}\}$, $\mathcal{S}(v) = S$ for all $v \in V$, such that $s_1 \leq_S s_2$ if and only if $P(s_1 \leq z) \geq P(s_2 \leq z)$ for all $z \leq C_{\max}$ and $s_1, s_2 \in S$. The weights $w \in \mathbb{W} = \mathbb{W}_e$ then are functions $w : (\Omega \to \mathbb{R}) \to (\Omega \to \mathbb{R})$, such that $w_c(s) = s + c$ for states $s : \Omega \to \mathbb{R}$ and costs $c : \Omega \to \mathbb{R}_{\geq 0}$.
The transposed graph *is $G^T = (V, E^T)$ with $(y, x) \in E^T$, if and only if $(x, y) \in E$. Then a weighting $\mathcal{W} : E^T \to \mathbb{W}$ is given by $\mathcal{W}(y, x) = w_{C(x,y)}$ for each $(x, y) \in E$.*
Given vertices $x, y \in V$ the most likely successful path problem (MLSPP2) *is to find an optimal path from y to x in the transposed graph G^T for the given starting state $s = 0$ (i.e. $s : \Omega \to \mathbb{R}$ with $s(\omega) = 0$ for all $\omega \in \Omega$).*

Remarks.

- Again we need to check for monotonicity and extensivity of weights. Moreover, the similarities and differences between MLSPP and MLSPP2 need to be discussed in the following.
- As it has already been mentioned, the reason to use the transposed graph is that the MLSPP has a target state and is designed to be a backward search. In order to define an appropriate SBSPP, the edges need to be reversed. Finding a path from y to x in G^T means to find a path from x to y in G.

Proposition 3.35. *Assuming stochastic independence for edge costs $c = C(x, y)$ and any state $s \in \mathcal{S}(y)$, the weight function $w = w_c = \mathcal{W}(y, x)$ is monotone.*

Proof. Let $s_1, s_2 \in \mathcal{S}(y)$, $s_1 \leq_S s_2$, such that s_1 and c as well as s_2 and c are stochastically independent. Because of $s_1 \leq_S s_2$ we have $P(s_1 \leq z) \geq P(s_2 \leq z)$ for all $z \leq C_{\max}$.

With $c: \Omega \to \mathbb{R}_{\geq 0}$, i.e. $c \geq 0$, we have

$$P(s_1 + c \leq z) = \int_{-\infty}^{z} P(s_1 \leq z', c = z - z')\mathrm{d}z'$$

Stochastic independence yields

$$= \int_{-\infty}^{z} P(s_1 \leq z') \cdot P(c = z - z')\mathrm{d}z'$$

And because $s_1 \leq_S s_2$:

$$\leq \int_{-\infty}^{z} P(s_2 \leq z') \cdot P(c = z - z')\mathrm{d}z'$$

Again, stochastic independence yields

$$= \int_{-\infty}^{z} P(s_2 \leq z', c = z - z')\mathrm{d}z'$$

And because $c \geq 0$, we have

$$= P(s_2 + c \leq z)$$

Therefore, $P(s_1 + c \leq z) \geq P(s_2 + c \leq z)$ for all $z \leq C_{\max}$ and $w(s_1) \leq_S w(s_2)$.

□

Proposition 3.36. *The weights $w_{C(x,y)}$ for any $(x, y) \in E$ are extensive.*

Proof. This follows immediately from the definition $C: E \to (\Omega \to \mathbb{R}_{\geq 0})$. Let

$s \in \mathcal{S}(y)$ be a state and let $c = C(x, y)$ for some edge $(x, y) \in E$.

$$\begin{aligned} P(s \le z) &= \int_{-\infty}^{z} P(s = z')\mathrm{d}z' \\ &\ge \int_{-\infty}^{z} P(s = z', c \le z - z')\mathrm{d}z' \\ &= P(s + c \le z) \end{aligned}$$

Then $P(s \le z) \ge P(s + c \le z)$ for all $z \le C_{\max}$.

□

This model is not simple, neither with respect to Definition 3.11 nor in common sense. Nevertheless the math behind it was already studied by Uludag et al. in [UUN$^+$09], whereas the algorithms do only need to solve an SBSPP (or the corresponding SBPRP) as well as finding numerically stable solutions to the convolution of CDFs (combined with a derivation). The former will be discussed in the next chapter, the latter is a trade-off between accuracy and time/space requirements.

Last but not least, the similarities and differences between MLSPP and its state-based version are as follows:

An obvious but only formal difference is the explicit use of random variables as states while the original problem uses these just indirectly. Furthermore, the problem is modelled explicitely as a backward approach, whereas the original problem just indirectly leads to a backward search. Starting with a target ending state and choosing the path having highest success to reach the target with costs less or equal than $C_{\max}$ is the exact same behaviour in both models, except for notational differences.

3.8 Stochastic Energy-Optimal Path Problems

The aim of the following models is to combine the stochastic aspects of the previous section with energy-optimality as discussed in Section 3.5. After that, a state-based approach is discussed, so that algorithms for SBSPP and SBPRP can be applied to stochastic energy-optimal problems without thinking.

First, a function handling the case distinction in the definition of energy costs (Definition 3.18) is introduced. Then, a specification of stochastic energy costs and the stochastic energy-optimal problem follow in a natural way.

Definition 3.37. *Given a battery capacity K, the* Battery Constraining Function *(BCF) $B_K : \mathbb{R} \cup \{\infty\} \to [0, K] \cup \{\infty\}$ is given by*

$$B_K(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } 0 \leq x \leq K, \\ \infty & \text{if } x > K. \end{cases}$$

Given random variables as edge costs $C : E \to (\Omega \to \mathbb{R})$ and a random variable $J : \Omega \to [0, K]$ describing the initial charge, the stochastic energy costs of a walks *$\gamma^k = (v_0, \ldots, v_k)$ are random variables $C_{KJ} : \Gamma \to (\Omega \to \mathbb{R})$ defined recursively by*

$$C_{KJ}(\gamma^k)(\omega) = \begin{cases} K - J(\omega) & \text{if } k = 0, \\ B_K(C_{KJ}(\gamma^{k-1})(\omega)) + C(v_{k-1}, v_k)(\omega) & \text{if } k > 0. \end{cases}$$

Remarks.

- The intermediate step of defining a Δ is not needed here, because of the battery constraining function. We could have applied it also in Definition

3.18. Then we would have there:

$$C_{KJ}(\gamma^k) = \begin{cases} K - J & \text{if } k = 0, \\ B_K(C_{KJ}(\gamma^{k-1})) + C(v_{k-1}, v_k) & \text{if } k > 0. \end{cases}$$

If $\omega \in \Omega$ is fixed or if Ω is a singleton, then both definitions are equivalent.

The stochastic routing models are given as follows.

Definition 3.38. *Given a graph $G = (V, E)$, vertices $x, y \in V$, a probability space $(\Omega, \mathcal{F}, P)$, maximum costs $C_{\max} \in \mathbb{R}_{\geq 0}$, independent random variables with known cumulative distribution functions $C : E \to (\Omega \to \mathbb{R}_{\geq 0})$ representing edge costs and $J : \Omega \to \mathbb{B}$ representing the initial charge, the* most likely energy-sufficient path problem *(MLESPP) is to find a path $\pi^*_{x,y}$, such that the probability $P(C_{KJ}(\pi^*_{x,y}) \leq C_{\max})$ is maximal among all paths from x to y, if it exists.*

The problem may also be described as

$$\pi^*_{x,y} \in \operatorname*{arg\,max}_{\pi_{x,y} \in \Pi_{x,y}} P(C_{KJ}(\pi_{x,y}) \leq C_{\max}).$$

A corresponding definition of expected value routing is more difficult in terms of energy costs. This is because costs of ∞ may occur as soon as the vehicle runs out of energy. If the probability is positive to do so, then the expected costs are infinite. Resulting from this, an optimal route among all paths never running out of energy is queried, independent of the actual outcome. If such a path does not exist, then an arbitrary path would be chosen.
In order to overcome that disadvantage, conditional expectation values may be used. The following model queries a path that has least expected costs, given that these costs are finite.

$$\pi^*_{x,y} \in \operatorname*{arg\,min}_{\pi_{x,y} \in \Pi_{x,y}} \mathbb{E}(C_{KJ}(\pi_{x,y}) \mid C_{KJ}(\pi_{x,y}) < \infty).$$

Since C_{KJ} is a function with codomain $[0, K] \cup \{\infty\}$, this *conditional* expected value is always finite, namely in $[0, K]$. This definition would prefer high varianced path costs, because those outcomes resulting in infinite costs are just ignored. When discussing the state-based approach, ignoring these outcomes with negative infinite results will break the monotonicity.
Another approach is to combine the previous equation with a minimal success probability $P_{\min}$.

$$\pi^*_{x,y} \in \underset{\substack{\pi_{x,y} \in \Pi_{x,y} \\ P(C_{KJ}(\pi_{x,y}) < \infty) \geq P_{\min}}}{\arg\min} \mathbb{E}(C_{KJ}(\pi_{x,y}) \mid C_{KJ}(\pi_{x,y}) < \infty).$$

Since ∞ here is used as a constant, it might be generalized to some $C_{\max}$ as it was done for the MLESPP. This results in the following equation.

$$\pi^*_{x,y} \in \underset{\substack{\pi_{x,y} \in \Pi_{x,y} \\ P(C_{KJ}(\pi_{x,y}) \leq C_{\max}) \geq P_{\min}}}{\arg\min} \mathbb{E}(C_{KJ}(\pi_{x,y}) \mid C_{KJ}(\pi_{x,y}) < \infty).$$

To be consistent, the model should always deliver a path, so if there is no path satisfying the success constraint but there are paths in the graph from x to y, then the model is allowed to return an arbitrary path.
This model is summarized in the following definition.

Definition 3.39. *Given a graph $G = (V, E)$, vertices $x, y \in V$, a probability space $(\Omega, \mathcal{F}, P)$, an energy constraint $C_{\max}$, a minimum success probability $P_{\min}$ and independent random variables with known cumulative distribution functions $C : E \to (\Omega \to \mathbb{R}_{\geq 0})$ representing edge costs and $J : \Omega \to \mathbb{B}$ representing the initial charge, the* expected energy-optimal path problem *(EEOPP) is to find a path $\pi^*_{x,y}$ with least expected costs $\mathbb{E}(C_{KJ}(\pi^*_{x,y}) \mid C_{KJ}(\pi^*_{x,y}) < \infty)$ among all paths from x to y with a probability at least $P_{\min}$ to have costs less or equal to $C_{\max}$. If no such path exists but y is reachable from x, then the problem is to find an arbitrary path from x to y. Otherwise, no path is returned.*

This model introduces expected value routing with energy-optimal aspects. An interesting alternative would be a combination of both the expected value and the success probability. A linear combination would be given by:

$$\pi^*_{x,y} \in \operatorname*{arg\,min}_{\pi_{x,y}\in\Pi_{x,y}} \alpha \cdot P(C_{KJ}(\pi_{x,y}) \leq C_{\max}) + \beta \cdot \mathbb{E}(C_{KJ}(\pi_{x,y}) \mid C_{KJ}(\pi_{x,y}) < \infty),$$

where α should be negative and β should be positive.
In general it is possible to use any bivariate function f, monotically decreasing in the first component and monotonically increasing in the second, such that above equation is written as:

$$\pi^*_{x,y} \in \operatorname*{arg\,min}_{\pi_{x,y}\in\Pi_{x,y}} f(P(C_{KJ}(\pi_{x,y}) \leq C_{\max}), \mathbb{E}(C_{KJ}(\pi_{x,y}) \mid C_{KJ}(\pi_{x,y}) < \infty)).$$

The derived model is formalized in the following definition:

Definition 3.40. *Given a graph $G = (V, E)$, vertices $x, y \in V$, a probability space $(\Omega, \mathcal{F}, P)$, an energy constraint $C_{\max}$, a bivariate function $f : [0,1] \times [0,K] \to \overline{\mathbb{R}}$ monotonically decreasing in the first component and monotonically increasing in the second, and independent random variables with known cumulative distribution functions $C : E \to (\Omega \to \mathbb{R}_{\geq 0})$ representing edge costs and $J : \Omega \to \mathbb{B}$ representing the initial charge, the* stochastic energy-optimal path problem *(SEOPP) is to find a path $\pi^*_{x,y}$ minimizing $f(P(C_{KJ}(\pi_{x,y}) \leq C_{\max}), \mathbb{E}(C_{KJ}(\pi_{x,y}) \mid C_{KJ}(\pi_{x,y}) < \infty))$ among all paths from x to y, if it exists.*

Both the MLESPP and EEOPP can be modeled as SEOPPs, as will be shown in the following.

Proposition 3.41. *The stochastic energy-optimal path problem comprises both MLESPP and EEOPP.*

Proof. This proof is straight-forward. An MLESPP is modeled by an SEOPP by choosing $f(x, y) = -x$, i.e. ignoring the expected value and maximizing the success.

An EEOPP is defined for some minimal success P_{min}. To model an EEOPP by an SEOPP, we choose

$$f(x,y) = \begin{cases} y & \text{if } x \geq P_{\text{min}}, \\ \infty & \text{if } x < P_{\text{min}}. \end{cases}$$

Using this definition for f, paths with success less than P_{min} are dismissed, because we try to minimize f. Among all paths with success at least P_{min}, we rate them path by their expected costs. If there is no path satisfying the success constraint, then all paths are valued equally, so an arbitrary path may be chosen. If y is not reachable from x at all, then SEOPP does not yield a path, as it is specified in EEOPP.

□

3.8.1 State-Based Perspective

Reformulating the above models as a state-based problem, as it has been done for energy-optimal routing and time-dependent routing, is difficult. This is because of the target functions, that need to be reflected in the comparison $\leq_S$. Just comparing success rates or expected values does not yield correct results, because the details of the CDFs are ignored. Furthermore, using the conditional expectation in the above manner will break the monotonicity of edge weights.

A backward approach, as it has been described in the previous section for the most likely successful path problem, is not possible either, e.g. because negative edge weights are explicitely allowed.

Instead the following model picks up the initial idea of the forward approach. Thereby the states being random variables are compared by their CDFs. Stochastic independency is assumed at all times, otherwise a similar formal approach as shown in the previous section may be realized, where weight functions yield a garbage state on the occurence of stochastic dependency.

In order to use BTFs and ETFs, these functions need to be redefined using random variables.

Definition 3.42. *Let K be a battery capacity, $\mathbb{B} = \mathbb{B}_K$ be a set of battery charges and let $(\Omega, \mathcal{F}, P)$ be a probability space. The set $SSBTF = SSBTF_K$ of* simple stochastic battery transformation functions *comprises functions $f : (\Omega \to \mathbb{B}) \to (\Omega \to \mathbb{B})$, such that there are corresponding $f' : \Omega \to SBTF_K$ with $f(J) = J'$ for random variables $J, J' : \Omega \to \mathbb{B}$ if and only if $J'(\omega) = f'(\omega)(J(\omega))$ for all $\omega \in \Omega$. Furthermore, the transformation $\tau_{SSBTF} : (\Omega \to \mathbb{R}) \to SSBTF$ projects any $c : \Omega \to \mathbb{R}$ to the function corresponding to $\omega \mapsto \tau_{SBTF}(c(\omega))$.*

This definition is just an elementwise application of SBTFs for each sample $\omega \in \Omega$. The reason to use $f : (\Omega \to \mathbb{B}) \to (\Omega \to \mathbb{B})$ instead of directly using $f' : \Omega \to \text{SBTF}$ is to have the codomain equaling the domain, such that functional composition can still be applied. Stochastic energy transformation functions may be redefined analogously or on top of stochastic battery transformation functions. The latter approach will be used in the following.

Definition 3.43. *Let K be a battery capacity, $\mathbb{B} = \mathbb{B}_K$ be a set of battery charges and let $(\Omega, \mathcal{F}, P)$ be a probability space. Let $S = (\Omega \to \mathbb{B}) \times \mathbb{R}$ where the first value represents the battery charge and the second value represents the potential energy or level of altitude. The set $SSETF = SSETF_K$ of* simple stochastic energy transformation functions *comprises functions $w_{h,f,h'} : (\Omega \to S) \to (\Omega \to S)$ for all $h, h' \in \mathbb{R}$ and $f \in SSBTF$ with $w_{h,f,h'}(J, h) \mapsto (f(J), h')$, such that the SBTFs of the corresponding $f'(\omega), \omega \in \Omega$ have costs of at least $h' - h$, i.e. $f'(\omega)(x) \leq x - h' + h$ for all $x \in \mathbb{B}$.*

This definition again is just an elementwise application of SETFs for each sample $\omega \in \Omega$. A proof is left out because this proposition is not necessary for the remaining discussion.

The definition of state-based stochastic, energy-optimal routing uses SSETFs as edge costs. Thereby different target functions, such as the probability of success or the average energy consumption is not handled directly. Instead the CDFs are used to compare two states. This probably yields more minimal states than a specialized model, but monotonicity and extensivity are not fulfilled in most of these cases. Using the CDFs allows to specify a valid SBSPP while solutions to above models are always a subset of the found solutions.

Definition 3.44. *Let $G = (V, E)$ be a graph with potentials $H : V \to \mathbb{R}$, let $(\Omega, \mathcal{F}, P)$ be a probability space, and let $C : E \to (\Omega \to \mathbb{R}_{\geq 0})$ be independent random variables with known cumulative distribution functions representing edge costs.*
The state space $S = \{\Omega \to \mathbb{B}\} \times \mathbb{R}$ describes pairs of battery states and potentials. Let $(J_1, h_1) \leq_S (J_2, h_2)$, if and only if $P(J_1 + h_1 \leq z) \leq P(J_2 + h_2 \leq z)$ for all $z \in \overline{\mathbb{R}}$ (because energy shall be maximized). Furthermore, let $\mathcal{S}(v) = \mathbb{B} \times \{H(v)\}$, $v \in V$ and let $\mathbb{W} = SSETF$ with the weighting given by $\mathcal{W}'(e) = \tau_{SSETF}(H(x), C(e), H(y))$ for $e = (x, y) \in E$.
Given vertices $x, y \in V$ the stochastic energy-optimal path problem *(SEOPP2) is to solve the described SBSPP for $(G, S, \leq_S, \mathcal{S}, \mathbb{W}, \mathcal{W}')$.*

As it was done for the deterministic model some properties must be proven before actually using the SBSPP to model stochastic energy-optimal routing. The following proofs are similar to those given in Sections 3.5 and 3.7.

Proposition 3.45. *Assuming stochastic independence for edge costs $c := C(x, y)$ and battery charges $J : \Omega \to \mathbb{B}$, the edge weights $w := \mathcal{W}(x, y)$ are monotone for all $(x, y) \in E$.*

Proof. Let f be the SSBTF corresponding to w. If f is monotone, then w is also monotone, because w again is just a translation by the potential $h := H(x)$ at x and by $h' := H(y)$ at y.
Let $(J_1, h), (J_2, h) \in \mathcal{S}(x)$, $(J_1, h) \leq_S (J_2, h)$, such that J_1 and c as well as J_2 and c are stochastically independent.

By definition $(J_1, h) \leq_S (J_2, h)$ follows $P(J_1 + h \leq z) \leq P(J_2 + h \leq z)$ for all $z \in \overline{\mathbb{R}}$. We want to show, that $P(f(J_1) + h' \leq z) \leq P(f(J_2) + h' \leq z)$ for all $z \in \overline{\mathbb{R}}$. Because both the premise and the conclusion are compared using the same translation for both states (h for premise and h' for conclusion), these values can be ignored here. So it needs to be shown, that $P(J_1 \leq z) \leq P(J_2 \leq z)$ for all $z \in \overline{\mathbb{R}}$ implies $P(f(J_1) \leq z) \leq P(f(J_2) \leq z)$ for all $z \in \overline{\mathbb{R}}$.

Because of 3.35 we already know, that $P(J_1 \leq z) \leq P(J_2 \leq z)$ for all $z \in \overline{\mathbb{R}}$ implies $P(J_1 - c \leq z) \leq P(J_2 - c \leq z)$ for all $z \in \overline{\mathbb{R}}$, as long as J_1 and c as well as J_2 and c are independent.

There are three cases for $z \in \overline{\mathbb{R}}$ now. Let f_ω be the SBTF described by $\tau_{\text{SBTF}}(c(\omega))$. Then $\omega \mapsto f_\omega$ is f' describing f by definition.

- For $z < 0$ we have $P(J \leq z) = P(J = -\infty)$ because $J : \Omega \to \mathbb{B}$ with $\mathbb{B} = \{-\infty\} \cup [0, K]$.

$$\begin{aligned}
P(f(J_1) = -\infty) &= P(\{\omega \in \Omega \mid f_\omega(J_1(\omega)) = -\infty\}) \\
&= P(\{\omega \in \Omega \mid J_1(\omega) < c(\omega)\}) \\
&= P(\{\omega \in \Omega \mid J_1(\omega) - c(\omega) < 0\}) \\
&= P(J_1 - c < 0) \leq P(J_2 - c < 0) \\
&= \ldots = P(f(J_2) = -\infty)
\end{aligned}$$

- For $z \in [0, K]$ we similarly have

$$\begin{aligned}
P(f(J_1) \leq z) &= P(\{\omega \in \Omega \mid f_\omega(J_1(\omega)) \leq z\}) \\
&= P(\{\omega \in \Omega \mid J_1(\omega) - c(\omega) \leq z\}) \\
&= P(J_1 - c \leq z) \leq P(J_2 - c \leq z) \\
&= \ldots = P(f(J_2) \leq z)
\end{aligned}$$

- For $z > K$ we obviously have $P(f(J_1) \leq z) = 1 = P(f(J_2) \leq z)$.

Therefore, all edge weights are monotone as long as stochastic independency of battery charges and edge costs are given.

□

Proposition 3.46. *The edge weights $w := \mathcal{W}(x, y)$ are extensive for all $(x, y) \in E$.*

Proof. This follows immediately from the definition $C : E \to (\Omega \to \mathbb{R}_{\geq 0})$. Let $(J, h) \in \mathcal{S}(x)$ be a state, let $c = C(x, y)$ and let f be the SSBTF corresponding to w. By definition we have $f'(\omega)(x) \leq x - h' + h$ for f' corresponding to f.

$$\begin{aligned} P(J + h \leq z) &= P(\{\omega \in \Omega \mid J(\omega) + h \leq z\}) \\ &= P(\{\omega \in \Omega \mid J(\omega) - h' + h + h' \leq z\}) \end{aligned}$$

Because for all $\omega \in \Omega$ with $J(\omega) - h' + h + h' \leq z$ we also have $f'(\omega)(J(\omega)) + h' \leq z$.

$$\begin{aligned} &\leq P(\{\omega \in \Omega \mid f'(\omega)(J(\omega)) + h' \leq z\}) \\ &= P(f(J) + h' \leq z) \end{aligned}$$

Therefore, the edge weights are extensive.

□

Unfortunately SEOPP2 is not comparable with SEOPP due to the conditional expectation. Nevertheless MLESPP may be compared to SEOPP2 in the sense, that MLESPP yields a path that is also found among all optimal solutions of SEOPP2.

Proposition 3.47. *Given an optimal path π_1 from x to y with respect to MLESPP, the state-based model SEOPP2 finds at least one path π_2 from x to y, that yields the same success value.*

Proof. Let $z_i = P(C_{KJ}(\pi_i) \leq C_{\max})$ be the success values of the paths π_i, $i \in \{1,2\}$. Obviously $z_1 \leq z_2$, by contradiction MLESPP would otherwise have chosen the path π_2. It remains to show $z_2 \leq z_1$.

Assume that $z_2 > z_1$. Then π_1 is not queried by SEOPP2 directly (otherwise there would be another path π_2' found by SEOPP2 equaling π_1 in its success value). Then with f_i being the SSETFs corresponding to π_i, we have $P(f_2(J) + H(y) \leq z) \leq P(f_1(J) + H(y) \leq z)$ for all $z \in \overline{\mathbb{R}}$ and $J : \Omega \to \mathbb{B}$. Because $H(y)$ just represents a translation, $P(f_2(J) \leq z) \leq P(f_1(J) \leq z)$ also holds for all $z \in \overline{\mathbb{R}}$ and $J : \Omega \to \mathbb{B}$.

In Theorem 3.27 we have already seen, that $C_{KJ}(\pi) = K - f(J)$ for all paths π in the deterministic case, where C_{KJ} are the edge costs and f is the corresponding SBTF. This also holds for the stochastic definitions, because for each $\omega \in \Omega$ we have $C_{KJ}(\pi)(\omega) = C_{KJ(\omega)}(\pi)$ and $K - f(J)(\omega) = K - f'(\omega)(J(\omega))$ (for f' corresponding to f in Definition 3.42) given indirectly by their respective definitions.

Therefore, $P(f_2(J) \leq z) \leq P(f_1(J) \leq z)$ for all $z \in \overline{\mathbb{R}}$ implies $P(K - C_{KJ}(\pi_2) \leq z) \leq P(K - C_{KJ}(\pi_1) \leq z)$ for all $z \in \overline{\mathbb{R}}$, which is the same as $P(C_{KJ}(\pi_2) \leq z) \geq P(C_{KJ}(\pi_1) \leq z)$ for all $z \in \overline{\mathbb{R}}$. Due to this we also have a higher success on π_2, i.e. $z_2 \leq z_1$, which contradicts the assumption $z_2 > z_1$. Therefore, a path with the same success value is found by SEOPP2.

$\square$

A counterexample easily shows that $P(J_1 \leq z) \leq P(J_2 \leq z)$ for all $z \in \overline{\mathbb{R}}$ of two random variables $J_1, J_2 : \Omega \to \mathbb{B}$ representing battery charges does not imply $\mathbb{E}(J_1 \mid J_1 \geq 0) \geq \mathbb{E}(J_2 \mid J_2 \geq 0)$.

Example. Let $(\Omega, \mathcal{F}, P)$ be the uniform distribution over two outcomes $\Omega = \{\omega_1, \omega_2, \omega_3\}$. Let $J_1(\omega_1) = 0$ and $J_1(\omega_2) = K$, then $\mathbb{E}(J_1) = \mathbb{E}(J_1 \mid J_1 \geq 0) = 0.5 \cdot K$. Let $J_2(\omega_1) = -\infty$ and $J_2(\omega_2) = K$, then $\mathbb{E}(J_2 \mid J_2 \geq 0) = K$. Obviously

$\mathbb{E}(J_2 \mid J_2 \geq 0) > \mathbb{E}(J_1 \mid J_1 \geq 0)$, but comparing the CDFs yields:

$$\begin{aligned}
P(J_1 \leq z) = 0 \leq 0.5 = P(J_2 \leq z) \qquad &\text{for} \quad z < 0,\\
P(J_1 \leq z) = 0.5 = P(J_2 \leq z) \qquad &\text{for} \quad 0 \leq z < K,\\
P(J_1 \leq z) = 1 = P(J_2 \leq z) \qquad &\text{for} \quad K \leq 0.
\end{aligned}$$

Therefore, $P(J_1 \leq z) \leq P(J_2 \leq z)$ for all $z \in \overline{\mathbb{R}}$ does not imply $\mathbb{E}(J_1 \mid J_1 \geq 0) \geq \mathbb{E}(J_2 \mid J_2 \geq 0)$.

Even though the expected shortest path problem can not (directly) be translated into a state-based approach, the model SEOPP2 does take into account more than just the success value. Since all optimal CDFs are queried by SEOPP2, a choice function may be designed that has similar properties to the comparison of expected values but is consistent with the comparison of CDFs. It is left as an open question here.

4 Algorithm Design

In the previous chapter different models for vehicle routing were discussed. The most simple version, the shortest path problem, comes with a lot of useful algorithms, beginning with Dijkstra's algorithm, the A^*-search, their bidirectional versions and some more advanced acceleration techniques such as contraction hierarchies or efficient graph partitionings (see for example [Gei08], [DGRW11]).

The advanced models such as energy-optimal routing and stochastic routing were remodeled as state-based problems. Therefore, finding algorithms solving the state-based routing problem is sufficient to find solutions to the previously discussed models. The aim now is to reformulate different shortest path algorithms to solve state based profile routing problems. Because the inverse of edge weight functions are missing or do not exist in general, bidirectional searches can not be applied to SBSPP. Instead, the associative functional composition in SBPRP is used.

Dijsktra's Algorithm and the A^*-search can be easily adapted to state-based routing, especially to SBSPP with totally preordered states. Concerning energy-optimal routing this corresponds exactly to the work of Sachenbacher et al. in [SLAH11]. The main idea to use these algorithms to partially preordered states and to profile routing lies in the use of *partial preorder queues* (PPQ) which will be introduced here. In short the elements in the queue are topologically sorted, but we keep track of all minimal elements instead of just one. This enables us to use Dijkstra's Algorithm and the A^*-search for SBSPP with partially preordered states, for example to solve the weight-constrained shortest path problem (even though the runtime may grow exponentially). Furthermore, the partial preorder queues may be used to improve other algorithms, such as contraction hierarchies, as we will see later.

Figure 4.1 summarizes the three parts to a complete algorithm. First the partial preorder queue will be discussed in its syntax and semantics. An efficient implementation remains open. Then the basics of an implementation of the state-based routing models are presented in terms of generic datatypes using Java. An implementation of the policies in energy-optimal profile routing are discussed in Section 4.3, where simple energy transformation functions are generalized to represent energy-optimal policies. Since most acceleration techniques may be generalized to solve profile routing problems by using PPQs, contraction hierarchies were chosen as an example here. Thereby, the most important terms of shortcuts and witnesses are translated to partially preordered policies.

4.1 Partial Preorder Queue

The partial preorder queue is the main utility to enable known shortest path algorithms to be applied to SBSPP with partially preordered states and to state-based profile routing SBPRP.

Usually a queue - a min-priority queue here - manages a subset of set X sorted by a relation $\leq$ on X and provides at least two methods, one for adding objects, one for extracting the least element (or the top priority element). If $\leq$ is a total preorder then there may be multiple minimal elements. These minimal elements are equivalent in the sense that $x_1 \leq x_2$ and $x_2 \leq x_1$ for all minimal elements $x_1, x_2 \in X$. In state-based routing we have seen that such equivalent states always yield equivalent resulting states in routing because of the monotonicity of weights. If $\leq$ is not a total relation but a partial preorder, then there may be multiple minimal elements that are not equivalent, i.e. we may neither have $x_1 \leq x_2$ nor $x_2 \leq x_1$

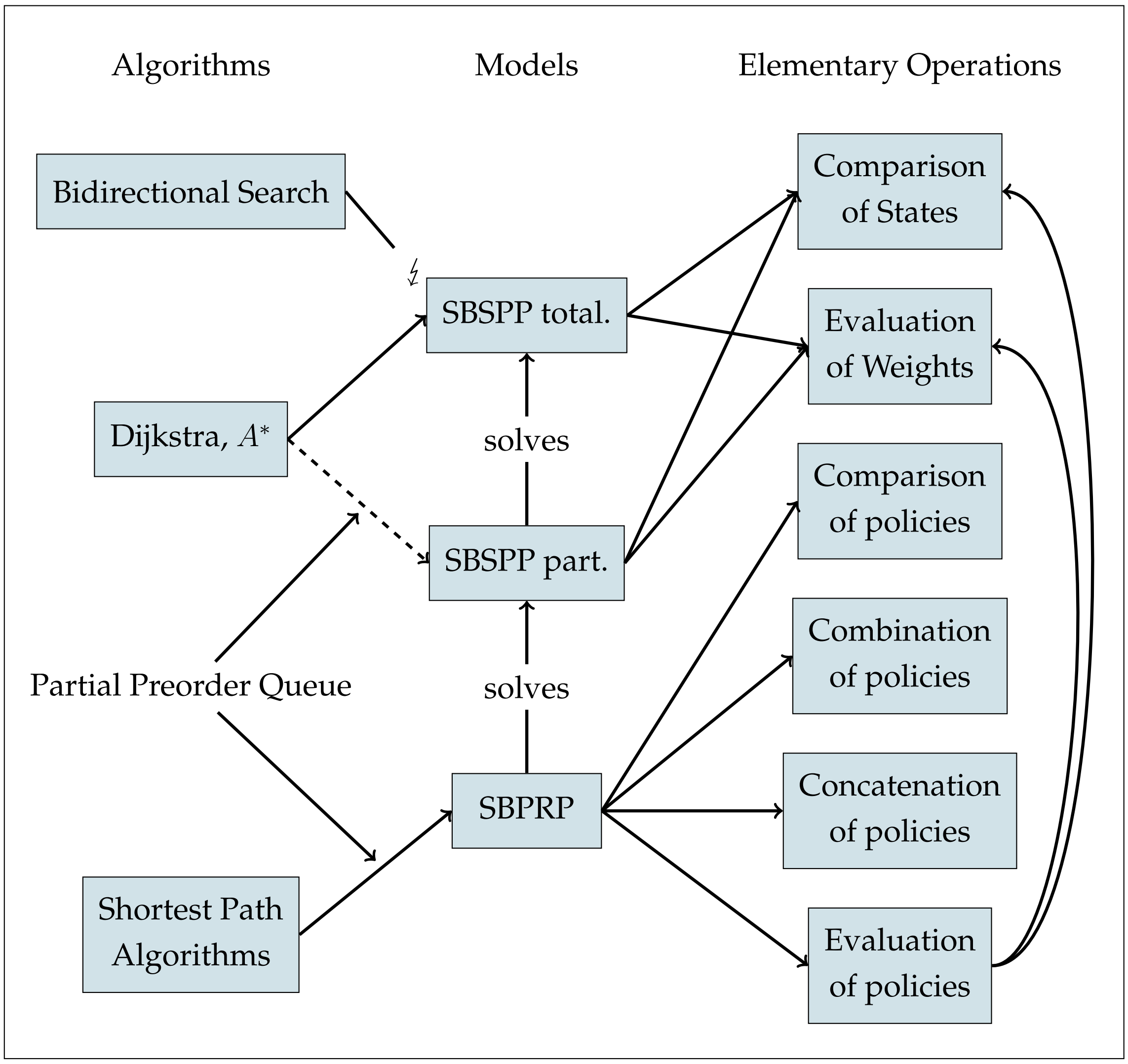


Figure 4.1: An algorithmic solution to state-based routing consists of three layers. The middle one represents the models where profile routing (SBPRP) comprises state-based shortest path problems. The right side represents elementary operations used by the model requiring specific implementations. The left side represents actual algorithms. The bidirectional search is not applicable to state-based routing, but we can reuse Dijkstra's Algorithm and A^*-search for an SBSPP with totally preordered states. Together with a partial preorder queue these may also be applied to SBSPP with partially preordered states. Most shortest path algorithms can be generalized using partial preorder queues to solve profile routing problems. Examples are given for Dijkstra's Algorithm (Section 4.2) and Contraction Hierarchies (Section 4.4).

for elements $x_1, x_2 \in X$. For state-based routing this means, that we keep track of both possibilities. Therefore, it is sensible to extend the definition of $\min$ to partial preorders (see also Section 3.2.1). This behaviour is described by

$$\min_{\leq} X = \{x \in X \mid \forall x' \in X : \; x' \leq x \rightarrow x \leq x'\} \, .$$

The function $\min_{\leq}$ yields a set of minimal elements, such that it is equivalent to the usual definition if $\leq$ is total.

Another important feature for the partial preorder queue is changing the value of an element already enqueued. The values need to be identified somehow. These identifications are usually called keys which are valued by an element of set X. Therefore, the queue is also a mapping from stored keys to values.

4.1.1 Syntax

The signature of an abstract data type $\Sigma = (\mathbb{S}, \mathbb{C}, \mathbb{F})$ is described by a set of sorts $\mathbb{S}$, a family of constants $\mathbb{C} = (\mathbb{C}^{\mathbf{s}})_{\mathbf{s} \in \mathbb{S}}$ and a family of operations $\mathbb{F} = (\mathbb{F}^{\mathbf{s_1} \times \ldots \times \mathbf{s_n} \rightarrow \mathbf{s_{n+1}} \times \ldots \times \mathbf{s_{n+m}}})_{\mathbf{s_1}, \ldots, \mathbf{s_{n+m}} \in \mathbb{S}}$, where $\mathbf{s_1}, \ldots, \mathbf{s_n}$ are the argument sorts and $\mathbf{s_{n+1}}, \ldots, \mathbf{s_{n+m}}$ are the result sorts.

Furthermore, axioms may be described by (Σ, X)-terms in first order logic using a family of countable variables $X = (X^{\mathbf{s}})_{\mathbf{s} \in \mathbb{S}}$.

In Figure 4.2 the specification of partial preorder queues is shown. It has four constructing operations for the sort **ppq** representing the queues, namely 'empty', 'insert', 'change' and 'pull', all of which we described in the introduction of this section. Furthermore, the sorts **key** and **value** represent the identification and the value enqueued. The crucial operation here is the operation $\leq$ comparing two values and yielding a bool. Axiom 1 (reflexivity and transitivity) enforces it to be a partial preorder.

Axiom 2 describes the properties of the operation 'in' with respect to all four con-

specification PARTIAL_PREORDER_QUEUE =

based on			BOOL, SET
sorts			**ppq**, **key**
constants	empty	:	**ppq**
operations	$\leq$	:	**value** $\times$ **value** $\to$ **bool**
	insert, change	:	**ppq** $\times$ **key** $\times$ **value** $\to$ **ppq**
	pull	:	**ppq** $\to$ **ppq** $\times$ **key** $\times$ **value**
	front	:	**ppq** $\to$ **set**
	isEmpty	:	**ppq** $\to$ **bool**
	in	:	**key** $\times$ **ppq** $\to$ **bool**
	get	:	**ppq** $\times$ **key** $\to$ **value**
variables	P, Q	:	**ppq**
	a, b	:	**key**
	x, y, z	:	**value**

axioms

1. Reflexivity ($x \leq x$). Transitivity ($x \leq y \ \land\ y \leq z \ \to\ x \leq z$).
2. a in empty is false.
   a in insert(P, a, x).
   b in insert(P, a, x) is b in P for $a \neq b$.
   b in change(P, a, x) is b in P.
   a in Q is false for pull$(P) = (Q, k, a)$.
   b in Q is b in P for pull$(P) = (Q, k, a)$ with $a \neq b$.
   isEmpty(P), if and only if there is no key a with a in P.
3. get(insert$(P, a, x), a)$ is x.
   get(insert$(P, a, x), b)$ is get(P, b) for $a \neq b$.
   get(change$(P, a, x), a)$ is x, if a in P.
   get(change$(P, a, x), b)$ is get(P, b) for $a \neq b$.
   get(Q, b) is get(P, b) for pull$(P) = (Q, k, a)$ with $a \neq b$.
4. $x \in$ front(P), if and only if there is key a with a in P, $x =$ get(P, a) and for all keys b we have $y =$ get$(b) \ \land\ b$ in $P \to (y \leq x \to x \leq y)$.
5. pull$(P) = (Q, a, x)$, if a in P, get$(P, a) = x$, not(a in Q), and $x \in$ front(P), as well as b in $P = b$ in Q and get$(P, b) =$ get(Q, b) for $b \neq a$.

end

Figure 4.2: Syntax and axioms of partial preorder queues. For readability an axiom yielding a **bool** x is a formula $x =$ true.

structors. Basically, this operation yields ′true′, if and only if the corresponding key was inserted and not removed afterwards. The operation ′change′ on a queue has no influence on the ′in′ operation. ′isEmpty′ is true, if and only if there is no key ′in′ the queue.

Axiom 3 then describes the operation ′get′ with respect to all but one constructors, such that ′get′ always yields the value which was assigned to the key by either ′insert′ or ′change′. The search for a key in an empty queue is not specified. This would be usually caught by some kind of exception mechanism.

Axiom 4 describes the ′front′ of a queue. A value x is part of the front if there is a corresponding key a in the queue, such that $y \leq x \rightarrow x \leq y$ for all values y comprised by the queue (having a corresponding key b inside the queue). This is equivalent to the mathematical definition of $\min_{\leq}$ of the introduction.

Axiom 5 describes the operation ′pull′. The specification is not deterministic here, the result may be any pair of key and value as long as the value is an element of the front. Furthermore, the other elements may not be affected by ′pull′.

4.1.2 Semantics

Algebras interpret signatures by providing explicit descriptions of the terms introduced in the signature. Thereby a Σ-algebra consists of an interpretation of the sorts, of the constants, and of the functions, such that no axiom is violated.

Besides the ground-term algebra, there is a simple algebra A on partial mappings directly derived from our preliminary thoughts about the queue. For BOOL and SET we may use the natural algebras on boolean values and sets. Let K be the set of all keys and V be the set of all possible values, the interesting interpretation then is the definition of the algebra for PARTIAL_PREORDER_QUEUE. Thereby, a queue is a partial mapping $K \rightsquigarrow V$ represented by a subset of pairs $K \times V$, such that any key appears at most once in the queue. For every partial preorder queue

q, key a and value x, we have:

$$\begin{aligned}
\mathbf{ppq}^A &= \{K \rightsquigarrow V\} \subseteq \mathcal{P}(K \times V), \\
\text{empty}^A &= \emptyset, \\
\text{insert}^A(q,a,x) &= (q \setminus (\{a\} \times V)) \cup \{(a,x)\}, \\
\text{change}^A(q,a,x) &= \begin{cases} \text{insert}^A(q,a,x) & \text{if } a \text{ in } q, \\ q & \text{otherwise}, \end{cases} \\
\text{pull}^A(q) &= (q \setminus \{(a,x\}, a, x) \quad \text{with} \quad x \in \text{front}^A(q) \quad \text{for} \quad q \neq \emptyset, \\
\text{front}^A(q) &= \min_{\leq}(\{x \in V \mid (a,x) \in q\}), \\
\text{isEmpty}^A(q) &= \begin{cases} \text{true}^A & \text{if } q = \emptyset, \\ \text{false}^A & \text{otherwise}, \end{cases} \\
\text{in}^A(a,q) &= \begin{cases} \text{true}^A & \text{if } q \cap (\{a\} \times V) \neq \emptyset, \\ \text{false}^A & \text{otherwise} \end{cases} \\
\text{get}^A(q,a) &= x \quad \text{for} \quad (a,x) \in q.
\end{aligned}$$

An efficient implementation remains open for further research and is not part of this thesis. The implementation may be specialized to different situations, e.g. the queue may often use the 'insert', 'change' and 'pull' methods, while 'front' is used rarely. This is a reasonable assumption because the main reason to use the front is to know when exactly to stop a search. Besides of just skipping the stop-condition check one may use different heuristics here, for example the average 'distance'.

Furthermore, the values may fulfill additional properties. One example is totality (i.e. $x \leq y \lor y \leq x$ for all values x, y) which yields a simple priority queue except for anti-symmetry playing no role for ordering elements in a priority queue anyway.

4.2 Model Implementation

In Figure 4.1 the development of algorithms was illustrated with three layers, the middle one representing the models. This section is for showing an implementation of the models SBSPP and SBPRP in the context of Java 1.6. All data types will be represented by using appropriate interfaces. They may also be realized as templates as in C++ or by any other type inheritance mechanism. Additional constraints that are not verified explicitely in the program but used for routing are added as commentaries.

In the following partial and total preorders are explicitely distinguished, because the specified algorithms shall only be applied to problems they are designed for. By using unique types for each model the compiler can check syntactically, wether a routing problem is solved by an appropriate algorithm.

- **Preorders**: The model layer should provide a generic interface for comparing two objects. Thereby total and partial preorders are differed. Since algorithms for problems with partial preorders can also solve problems with total preorders it is sensible to use type inheritance in this place such that partial preorders are more general than total preorders. The interfaces may be seen as concretizations of the abstract data type ELEMENT (described in the previous section and listed in Appendix A).

```java
interface PartialPreorder <K> {
  // This relation must be reflexive and transitive.
  public boolean lessEqual(K other);
}
interface TotalPreorder <K> extends PartialPreorder<K>{
  // The relation lessEqual must be total.
}
```

  A class implementing this interface is supposed to set K to the implementing class itself.

- **States**: States are values, i.e. not implementing any methods.

```
interface State {}
```

The only reason to use this interface is to identify which objects are states and which are not.

- **Weights**: Edge weights are mappings from input states to output states.

```
interface Weight
   <S extends State & Preorder<S>> {
  // The evaluate function must be monotone and extensive.
  public S evaluate(S state);
}
```

Even though weight composition will be included in the concatenation of policies, an explicit interface may be of interest for later use.

```
interface WeightFactory
   <S extends State & Preorder<S>,
    W extends Weight<S>> {
  public W compose(W w1, W w2);
}
```

- **Vertex** and **Edge**: These are required for graphs. The only reason for an interface is again to identify vertices and edges.

```
interface Vertex {}
interface Edge {}
```

- **Graph**: In this case, a read-only directed graph.

```
interface Graph
   <V extends Vertex,
    E extends Edge> {
  // Vertex operations
  public int getOrder();
  public Set<V> getVertices();

  // Edge operations
  public int getSize();
  public boolean isAdjacent(V x, V y);
  public E getEdge(V x, V y);
  public Set<V> getSuccessors(V x);
  public Set<V> getPredecessors(V x);
}
```

For more complex algorithms the graph class of course needs to be extended, but efficient data structures are part of the algorithms themselves and not included in the routing model.

- **Policy**: A mapping from states to paths, i.e. lists of vertices.

```
interface Policy
   <S extends State & Preorder<S>,
    V extends Vertex> {
  public List<V> evaluate(S state);
}
```

A factory is used to construct policies:

```
interface PolicyFactory
   <S extends State & Preorder<S>,
    W extends Weight<S>,
    V extends Vertex,
    M extends Policy<S, V> & Preorder<M>> {
  public M createEmpty(V x, V y);
  public M createFromEdge(V x, V y, W w);
  public M createSingleton(V x);
  public M compose(M m1, M m2);
  public M combine(M m1, M m2);
}
```

- **Algorithm**: An algorithm implementation then would realize one of the following interface.

```
interface SBSPPTotalAlgorithm
   <S extends State & TotalPreorder<S>,
    V extends Vertex> {
  public List<V> searchPath(V x, V y, S s);
}
interface SBSPPPartialAlgorithm
   <S extends State & PartialPreorder<S>,
    V extends Vertex>
    extends SBSPPTotalAlgorithm<S, V> {
  // The preorder on states must be total.
}
interface SBPRPAlgorithm
   <S extends State & PartialPreorder<S>,
    V extends Vertex,
    M extends Policy<S, V> & PartialPreorder<M>> {
  public M searchPolicy(V x, V y);
}
```

Notice that the only difference between the interfaces SBSPPTotalAlgorithm and SBSPPPartialAlgorithm is the comment about the totality of the preorder. The reason to define distinct interfaces is to keep algorithms for both models seperate. Otherwise one could apply an algorithm for SBSPP with totally preordered states to a problem with partially preordered states. Due to the inheritance relation it is possible to apply the more general algorithm with a partial preorder to problems with a total preorder.

- **Partial Preorder Queue**: Priority queues are needed in almost any routing algorithm. As we have seen in the previous section, we want to use partial preorder queues for SBSPP and SBPRP. A Java 1.6 interface[1] is derived directly from the definition of the signature.

```java
interface PartialPreorderQueue<K, V extends PartialPreorder<V>> {
  public void insert(K k, V v);
  public void change(K k, V v);
  public java.util.Map.Entry<K, V> pull();
  public Set<V> front();
  public boolean isEmpty();
  public boolean in(K k);
  public V get(K k);
}
interface TotalPreorderQueue<K, V extends TotalPreorder<V>>
    extends PartialPreorderQueue<K, V>{
}
```

  If a given routing problem can be formulated as an SBSPP with totally preordered states, then a simple priority queue may also be used. In contrast to the partial preorder queue, this is called a total preorder queue even though it is equivalent to usual priority queues.

  The constructor 'empty' is best placed in a factory:

```java
interface PartialPreorderQueueFactory
   <K, V extends PartialPreorder<V>,
    Q extends PartialPreorderQueue<K, V>> {
  public Q createEmpty();
}

```

[1]Because a method in Java 1.6 may only return one result, i.e. no tuples, the type 'java.util.Map.Entry' is used as a workaround here. In any way it makes sense to do so because the given definition of a partial preorder queue actually is a mapping, such that the partial preorder queue could also implement the 'java.util.Map' interface.

```
interface TotalPreorderQueueFactory
   <K, V extends TotalPreorder<V>,
    Q extends TotalPreorderQueue<K, V>> {
  public Q createEmpty();
}
```

4.2.1 Example: Dijkstra's Algorithm

As a first example of how all three layers in Figure 4.1 may work together, the following listing shows Dijkstra's Algorithm in the style of A^* (starting with a queue only containing the start vertex) solving an SBSPP with totally preordered states.

```
class DijkstrasAlgorithm
   <S extends State & TotalPreorder<S>,
    W extends Weight<S> & Edge,
    V extends Vertex,
    G extends Graph<V, W>,
    Q extends TotalPreorderQueue<V, S>,
    QF extends TotalPreorderQueueFactory<V, S, Q>>
    implements SBSPPTotalAlgorithm<S, V> {
  private G graph;
  private Q queue = null;
  private QF queueFactory;
  private Map<V, S> states = new TreeMap<V, S>();
  private Map<V, V> predecessor = new TreeMap<V, V>();
  public DijkstrasAlgorithm(G graph, QF queueFactory) {
    this.graph = graph;
    this.queueFactory = queueFactory;
  }
  public List<V> searchPath(V x, V y, S s) {
    if (x == y)
      return Collections.singletonList(x);
    states.clear();
    predecessor.clear();
    states.put(x, s);
    queue = queueFactory.createEmpty();
    queue.insert(x, s);
    while (!queue.isEmpty()) {
      V currVertex = queue.pull().getKey();
      S currState = states.get(currVertex);
      if (states.containsKey(y) && states.get(y).lessEqual(currState))
        break; // Because of the totality of the (pre-)order
      for (V succNode : graph.getSuccessors(currVertex)) {
        S succState = states.get(succNode);
        W edgeWeight = graph.getEdge(currVertex, succNode);
        S sum = edgeWeight.evaluate(currState);
        // If weight is improved this way
        if (succState == null
            || (sum.lessEqual(succState)
                && !succState.lessEqual(sum))) {
          states.put(succNode, sum);
          predecessor.put(succNode, currVertex);
          queue.insert(succNode, sum);
        }
      }
    }
    return getPath(x, y);
  }
  private List<V> getPath(V x, V y) {
    List<V> res = new LinkedList<V>();
    V pred = y;
    while (pred != x) {
      res.add(0, pred);
      pred = predecessor.get(pred);
    }
```

```
      res.add(0, pred);
      return res;
    }
  }
```

This programm shows an example of an algorithm applied to one of the defined models, namely the SBSPP with totally preordered states. Therefore, it is possible to solve time-dependent routing and energy-optimal routing both with the same given algorithm. An extension to A^* can be easily implemented by using an appropriate queue.

As a minimal example on how to use the algorithm the simple shortest path problem is presented in terms of the given interfaces (see example from Section 3.3):

```
class Distance implements State, TotalPreorder<Distance> {
  double value;
  public Distance(double value) {this.value = value;}
  public boolean lessEqual(Distance other) {
    return this.value <= other.value;
  }
}
class DistanceWeight implements Edge, Weight<Distance> {
  double costs;
  public DistanceWeight(double costs) {this.costs = costs;}
  public Distance evaluate(Distance state) {
    return new Distance(state.value + this.costs);
  }
}
```

A further simple example is energy-optimal routing as described in Section 3.5.

```
class Energy implements State, TotalPreorder<Energy> {
  double h; // Potential.
  double j; // Charge.
  public Energy(double h, double j) {this.h = h; this.j = j;}
  public boolean lessEqual(Energy other) {
    return this.h + this.j <= other.h + other.j;
  }
}
class EnergyWeight implements Edge, Weight<Energy> {
  static double K = ...;
  double h1, h2;
  double costs;
  public EnergyWeight(double h1, double h2, double costs) {
    this.h1 = h1;
    this.h2 = h2;
    this.costs = costs;
  }
  public Energy evaluate(Energy state) {
    if (state.h != h1)
      throw new RuntimeException(...);
    double j2 = state.j - (costs + h2 - h1);
    if (j2 < 0)
      j2 = Double.NEGATIVE_INFINITY;
    else if (j2 > K)
      j2 = K;
    return new Energy(j2, h2);
  }
}
```

This shows that implementing SBSPP with totally preordered states is straightforward. An example for an SBSPP with states not totally preordered is the weight-constrained shortest path problem as described in Section 3.6. We skip an implementation of Dijkstra's Algorithm for that problem because it is rather technical. Nevertheless, the state-based profile routing is an interesting concept, so an adaptation of Dijkstra's Algorithm to solve SBPRP will be shown next.

```
class DijkstrasSBPRP
   <S extends State & PartialPreorder<S>,
    W extends Weight<S> & Edge,
    V extends Vertex,
    M extends Policy<S, V> & PartialPreorder<M>,
    MF extends PolicyFactory<S, W, V, M>,
    G extends Graph<V, W>,
    Q extends PartialPreorderQueue<V, M>,
    QF extends PartialPreorderQueueFactory<V, M, Q>>
    implements SBPRPAlgorithm<S, M, V> {
  private G graph;
  private Q queue = null;
  private QF queueFactory;
  private MF policyFactory;
  private Map<V, M> policies = new TreeMap<V, M>();
  public DijkstrasSBPRP(G graph, QF queueFactory, MF policyFactory) {
    this.graph = graph;
    this.queueFactory = queueFactory;
    this.policyFactory = policyFactory;
  }
  public M searchPolicy(V x, V y) {
    if (x == y)
      return policyFactory.createSingleton(x);
    policies.clear();
    queue = queueFactory.createEmpty();
    for (V succVertex : graph.getSuccessors(x)) {
      W edgeWeight = graph.getEdge(x, succVertex);
      M edgePolicy = policyFactory.createFromEdge(x,
          succVertex, edgeWeight);
      policies.put(succVertex, edgePolicy);
      queue.insert(succVertex, edgePolicy);
    }
    while (!queue.isEmpty()) {
      // Check for break condition
      if (policies.containsKey(y)) {
```

```java
        M my = policies.get(y);
        Set<M> front = queue.front();
        boolean doBreak = true;
        for (M m : front)
          if (!my.lessEqual(m))
            doBreak = false;
        if (doBreak)
          break;
      }
      // Expand
      V currVertex = queue.pull().getKey();
      M currPolicy = policies.get(currVertex);
      for (V succVertex : graph.getSuccessors(currVertex)) {
        M succPolicy = policies.get(succVertex);
        W edgeWeight = graph.getEdge(currVertex, succVertex);
        M edgePolicy = policyFactory.createFromEdge(currVertex,
            succVertex, edgeWeight);
        M sum = policyFactory.compose(currPolicy, edgePolicy);
        // If policy is improved this way
        if ((succPolicy == null)
            || (sum.lessEqual(succPolicy)
                && !succPolicy.lessEqual(sum))) {
          M combination = policyFactory.combine(succPolicy, sum);
          policies.put(succVertex, combination);
          queue.insert(succVertex, combination);
        }
      }
    }
    return policies.get(y);
  }
}
```

Proofs of correctness are omitted here because they are rather technical and the reasoning is already given informally in the previous chapter. The main concept of the proof would be to build a tree of optimal policies, as it was done in the introduction for the shortest path problem (see Proposition 1.3).

There are two important aspects that need to be taken care of when proving correctness. The first one is the break condition, the second one is the combination of policies.

The break condition 'doBreak' becomes true, if and only if there is no policy left in the front of the queue, that could possibly improve the currently computed policy to the target. In other words, the tentative policy found to the target must be less or equal to all other policies in the queue.

The combination of policies and thus enqueuing another vertex is done, if the policy using the traversed edge 'improves' the current policy. By using the induced strict order as it is commonly done in the original version of Dijkstra's Algorithm (which is correct for total preorders), we completely miss those policies in which paths actually are combined to form new policies.

4.3 Energy-Optimal Policies

The simple battery transformation functions (SBTF) discussed in the previous chapter in Section 3.5 are closed under functional composition. Up to this point we have no substantial differences between the two definitions of energy path costs and battery transformation functions because energy path costs can be composed in a similar fashion which was also done by Eisner et al. in [EFS11].

Since SBTFs are not extensive, simple energy transformation functions (SETF) are used in the following. These were monotone and extensive. Furthermore, implementations of comparison, combination, concatenation and evaluation of policies are needed as illustrated in Figure 4.1.

Basically a policy is a set of paths. Therefore using the representation of sets, the combination and concatenation of policies is intuitive (see Definition 3.9). Furthermore, evaluation of policies is also simple because it just requires to compute the resulting state for each path and choose one path with an optimal resulting state. An exact implementation of the comparison of policies is more difficult. Eisner et al. do this by evaluating the path weight functions at all 'break points' (interval endpoints), which they assume to be correct. A formal proof (for energy transformation functions) is given in Proposition 4.12.

4.3.1 Definition

A more compact representation of policies in energy-optimal routing is given by a generalization of SBTF and SETF. Both are closed under functional composition as shown by Proposition 3.23, while they are not closed under the elementwise $\max$-operator representing combination. The following definition describes the closure of both transformation functions under $\max$.

Definition 4.1. *Let K be a battery capacity and $\mathbb{B} = \mathbb{B}_K$ be a set of battery charges. The set BTF $=$ BTF$_K$ of* battery transformation functions *comprises all functions $f : \mathbb{B} \to \mathbb{B}$ corresponding to a family of $k \in \mathbb{N}_0$ triples $(a_i, b_i, c_i)_{i=1,\dots,k}$ with*

- $a_1 \geq 0$, $b_k \leq K$, $c_1 \leq a_1$, $c_k \geq b_k - K$,
- $a_i \leq b_i$ *for* $i = 1, \dots, k$,
- $a_i < a_{i+1}$, $b_i \leq a_{i+1}$, $b_i - c_i \leq a_{i+1} - c_{i+1}$ *for* $i = 1, \dots, k-1$,

such that

$$f(J) = \begin{cases} -\infty & \text{if } k = 0 \text{ or else } J < a_1, \\ J - c_i & \text{if } k > 0 \text{ and } J \in [a_i, b_i), i = 1, \dots, k, \\ b_i - c_i & \text{if } k > 0 \text{ and } J \in [b_i, a_{i+1}), i = 1, \dots, k-1, \\ b_k - c_k & \text{if } k > 0 \text{ and } J \geq b_k. \end{cases}$$

Furthermore, the transformation $\tau_{BTF} : \mathbb{R} \to BTF$ is equivalent to τ_{SBTF}.

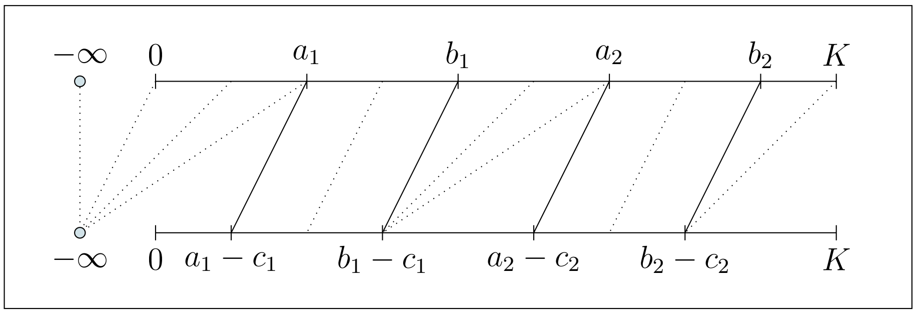


Figure 4.3: Battery transformation functions (BTFs) are used to describe the change of battery charge when traversing edges or paths in a graph. The contraction of vertices results in the combination of different BTFs. We make routing choices based on the intervals, which will be enriched by additional information about the actual paths.

We may illustrate battery transformation functions in the same way we did for SBTF, as is shown in Figure 4.3. The energy transformation functions now are derived from BTF in the same way as SETF was derived from SBTF.

Definition 4.2. *Let K be a battery capacity and $\mathbb{B} = \mathbb{B}_K$ be a set of battery charges. Let $S = \mathbb{B} \times \mathbb{R}$, where the first value represents the battery charge and the second value represents the potential energy or level of altidude. The set $ETF = ETF_K$ of* energy transformation functions *comprises function $w_{h,f,h'} : \mathbb{B} \times \{h\} \to \mathbb{B} \times \{h'\}$ for all $h, h' \in \mathbb{R}$ and $f \in BTF$, with $w_{h,f,h'}(J, h) \mapsto (f(J), j'')$, such that we have $c_i \geq h' - h$ for all triples (a_i, b_i, c_i), $i = 1, \ldots, k$ corresponding to f. Furthermore, the transformation $\tau_{ETF} : R \times \mathbb{R}_{\geq 0} \times \mathbb{R} \to ETF$ is equivalent to τ_{SETF}.*

4.3.2 Properties

In order to use BTF and ETF various properties needs to be proven now. Both definitions must be well-defined, the questionable part is the set of properties describing the family of triples corresponding to BTF as well as the question, if SBTF $\subseteq$ BTF and SETF $\subseteq$ ETF actually hold. Furthermore, monotonicity and extensivity of ETF as well as closure under functional composition and the $\max$ operator needs to be shown. Another interesting property is that BTF and ETF do not contain any more elements than the closure under $\max$ of their respective sets SBTF and SETF. This means, that BTF and ETF actually are the closures of SBTF and SETF under $\max$. Two other important aspects are the descriptive complexity and the comparison of ETF.

When it comes to above definitions, one may doubt the required properties of the family of triples $(a_i, b_i, c_i)_{i=1,...,k}$ describing a BTF. The idea behind choosing these properties is to have multiple intervals on the domain projecting values in the same way as is done by SBTFs. Of course, if the intervals specified by $[a_i, b_i)$ (requiring $a_i \leq b_i$ for $i = 1, \ldots, k$) are subsets of the domain $\mathbb{B}$, then $a_i \geq 0$ and $b_i \leq K$, but together with $b_i \leq a_{i+1}$ for $i = 1, \ldots, k-1$ the inequalities $a_1 \geq 0$ and $b_k \leq K$ are sufficient. Moreover, the intervals should be disjoint such that the case distinction in Definition 4.1 is unambiguously.

Another point is to avoid duplicate intervals, thus $a_i < a_{i+1}$ is chosen for $i = 1, \ldots, k-1$. Notice that b_i may be equal to b_{i+1}, i.e. $a_i < b_i = a_{i+1} = b_{i+1}$. So empty intervals are allowed even though they can not be dismissed, because they are still used for the third (and fourth) case in their case distinction. The idea behind this are singletons, a path always yielding either negative infinity or another constant value. This is represented by an SBTF $f_{a,b,c}$ with $a = b$ (see Figure 3.6).

Furthermore, c_i describe the costs for battery charges in the interval $[a_i, b_i)$. These costs must be chosen such that the function is monotone and inside the codomain $\mathbb{B}$. Monotonicity is guaranteed by $b_i - c_i \leq a_{i+1} - c_{i+1}$ for $i = 1, \ldots, k-1$ as will be proven in Proposition 4.10. Finally the image is kept inside $[0, K]$ by requiring $c_1 \leq a_1$ and $c_k \geq b_k - K$.

Proposition 4.3. *SBTF $\subseteq$ BTF.*

Proof. SBTF comprises functions f_∞ and $f_{a,b,c}$ with $f_\infty(J) = -\infty$ and

$$f_{a,b,c}(J) = \begin{cases} -\infty & \text{for } J = -\infty \text{ or } J \in [0, a), \\ J - c & \text{for } J \in [a, b), \\ b - c & \text{for } J \in [b, K], \end{cases}$$

for all $J \in \mathbb{B}$. By definition, f_∞ corresponds to the BTF with $k = 0$ intervals. Furthermore, $f_{a,b,c}$ corresponds to the BTF with one triple (a, b, c), which can be seen directly by comparing both case distinctions.
The only difference in the definitions of SETF and ETF is the use of SBTF for SETF and BTF for ETF. Because SBTF is contained in BTF, the inclusion SETF $\subseteq$ ETF is obvious.

□

Because $w_{h,f,h'} \in$ ETF for any $f \in$ BTF, we also have $w_{h,f,h'} \in$ ETF for any $f \in$ SBTF, i.e. an SETF:

Corollary 4.4. *SETF $\subseteq$ ETF.*

Comparing BTF and the closure of SBTF under the $\max$ operator consists of two directions. Inclusion is shown easily, but equality requires the closure of BTF under functional composition and the $\max$ operator as shown in Propositions 4.7 and 4.8.

Proposition 4.5. *BTF is a subset of the closure of SBTF under the* $\max$ *operator.*

Proof. Let f be a BTF with corresponding triples $(a_i, b_i, c_i)_{i=1,\ldots,k}$. If $k \leq 1$, then f is an SBTF. Otherwise, let $f' = \max(f_1, \ldots, f_k)$ for SBTFs $f_i = f_{a_i,b_i,c_i}$, $i = 1, \ldots, k$. Now, f equals f', which can be seen by the following case distinction.

- If $J \in [a_i, b_i), i = 1, \ldots, k$, then $f(J) = J - c_i = f_i(J)$. Furthermore, for

$f_j, j = i+1, \ldots, k$ we have $f_j(J) = -\infty$ and for $f_j, j = 1, \ldots, k-1$ we have

$$f_j(J) = f_j(b_j) = b_j - c_j \leq \ldots \leq a_i - c_i \leq f_i(J).$$

- If $J \in [b_i, a_{i+1}], i = 1, \ldots, k-1$, then $f(J) = b_i - c_i = f_i(J)$. Furthermore, for $f_j, j = i+1, \ldots, k$ we have $f_j(J) = -\infty$ and for $f_j, j = 1, \ldots, k-1$ we have

$$f_j(J) = f_j(b_j) = b_j - c_j \leq \ldots \leq a_i - c_i \leq f_i(J).$$

- If $J \geq b_k$, then $f(J) = b_k - c_k = f_k(J)$. Furthermore, for $f_j, j = 1, \ldots, k-1$ we have

$$f_j(J) = f_j(b_j) = b_j - c_j \leq \ldots \leq a_k - c_k \leq f_k(J).$$

Therefore, $f(J) = (\max(f_1, \ldots, f_k))(J) = f'(J)$ in any case.

□

The following corollary is an immediate result from the previous proposition.

Corollary 4.6. *ETF is a subset of the closure of SETF under combination* ($\max$ *operator).*

The closures need a little more attention, because we would also like to derive simple algorithms for both operations. This is only done for BTF, because both closures (composition and $\max$ operator) for ETFs are directly bound to the closures of BTFs.

Proposition 4.7. *BTF is closed under functional composition.*

Proof. Let f_1, f_2 be two BTFs with corresponding triples $(a_i^1, b_i^1, c_i^1)_{i=1,\ldots,k_1}$ and $(a_j^2, b_j^2, c_j^2)_{j=1,\ldots,k_2}$ and let $g = f_1 \circ f_2$. If $k_1 = 0$ or $k_2 = 0$, then $g(J) = (f_1 \circ f_2)(J) = -\infty$ for all $J \in \mathbb{B}$, which is a BTF. So assume, that $k_1, k_2 > 0$ in the following.
To find the corresponding tuples for g, we are interested in $g([a_i^1, b_i^1])$ for all $i = 1, \ldots, k_1$, because for $J < a_1^1$ we have $g(J) = -\infty$, for $J \in (b_i^1, a_{i+1}^1)$, $i = 1, \ldots, k-1$ we have $g(J) = g(b_i^1)$ and for $J > b_{k_1}^1$ we have $g(J) = g(b_{k_1}^1)$.

There are five cases.

- The interval $[a_j^2, b_j^2]$ is included in the image of $[a_i^1, b_i^1]$ under f_1.

  Let $a_j^2 \geq a_i^1 - c_i^1$ and $b_j^2 \leq b_i^1 - c_i^1$, then $f_1^{-1}([a_j^2, b_j^2]) = [a_j^2 + c_i^1, b_j^2 + c_i^1]$ and $g(J) = J + c_i^1 + c_j^2$ for all $J \in [a_j^2 + c_i^1, b_j^2 + c_i^1]$. Therefore, we introduce the triple

  $$(a_{ij}, b_{ij}, c_{ij}) = (a_j^2 + c_i^1, b_j^2 + c_i^1, c_i^1 + c_j^2).$$

- The interval $[a_j^2, b_j^2]$ has elements only less than the image of $[a_i^1, b_i^1]$ under f_1.

  Let $a_j^2, b_j^2 < a_i^1 - c_i^1$ and $a_{j+1}^2 > a_i^1 - c_i^1$, then for all $J \in [a_i^1, a_{j+1}^2 + c_i^1)$ we have $g(J) = f_2(J - c_i^1) = f_2(b_j^2) = b_j^2 - c_j^2$. Therefore, we introduce a singleton

  $$(a_{ij}, b_{ij}, c_{ij}) = (a_i^1, a_i^1, a_i^1 - b_j^2 + c_j^2).$$

- The interval $[a_j^2, b_j^2]$ has elements inside and less than the image of $[a_i^1, b_i^1]$ under f_1.

  Let $a_j^2 < a_i^1 - c_i^1$ and $b_j^2 + c_i^1 \in [a_i^1, b_i^1)$, then for all $J \in [a_i^1, b_j^2 + c_i^1]$ we have $g(J) = f_2(J - c_i^1) = J - c_i^1 - c_j^2$. Therefore, we introduce a triple

  $$(a_{ij}, b_{ij}, c_{ij}) = (a_i^1, b_j^2 + c_i^1, c_i^1 + c_j^2).$$

- The interval $[a_j^2, b_j^2]$ has elements in and greater than the image of $[a_i^1, b_i^1]$ under f_1.

  Let $b_j^2 > b_i^1 - c_i^1$ and $a_j^2 + c_i^1 \in [a_i^1, b_i^1]$, then for all $J \in [a_j^2 + c_i^1, b_i^1]$ we have $g(J) = f_2(J - c_i^1) = J - c_i^1 - c_j^2$. Therefore, we introduce a triple

  $$(a_{ij}, b_{ij}, c_{ij}) = (a_j^2 + c_i^1, b_i^1, c_i^1 + c_j^2).$$

- The interval $[a_j^2, b_j^2]$ is a superset of the image of $[a_i^1, b_i^1]$ under f_1.

  Let $a_j^2 < a_i^1 - c_i^1$ and $b_j^2 > b_i^1 - c_i^1$, then for all $J \in [a_i^1, b_i^1]$ we have $g(J) =$

$f_2(J - c_i^1) = J - c_i^1 - c_j^2$. Therefore, we introduce a triple

$$(a_{ij}, b_{ij}, c_{ij}) = (a_i^1, b_i^1, c_i^1 + c_j^2).$$

The introduced intervals shall be lexically ordered, first by index i and secondly by index j. Since above case distinction is exhaustive, the introduced intervals are sufficient to represent g as a battery transformation function. Notice that there is no case required for intervals $[a_j^2, b_j^2]$ having all elements greater than the image of $[a_i^1, b_i^1]$ under f_1, because for all $J \in [a_i^1, b_i^1]$ the inequality $f_1(J) < a_j^2$ holds and the battery charge is insufficient to traverse paths with higher battery charge requirements in f_2.

$\square$

Proposition 4.8. *BTF is closed under the* $\max$ *operator*

Proof. Let f_1, f_2 be two BTFs with corresponding triples $(a_i^1, b_i^1, c_i^1)_{i=1,\ldots,k_1}$ and $(a_j^2, b_j^2, c_j^2)_{j=1,\ldots,k_2}$ and let $h = \max(f_1, f_2)$. If $k_1 = 0$, then $f_1(J) = -\infty \leq f_2(J)$ for all $J \in \mathbb{B}$, so $h = f_2$. Analogously, if $k_2 = 0$ otherwise, then $f_2(J) = -\infty \leq f_1(J)$ for all $J \in \mathbb{B}$, so $h = f_1$. So assume that $k_1, k_2 > 0$ in the following.

Notice that for all $J \in \mathbb{B}$ not included in any interval $[a_i^1, b_i^1)$ of f_1 nor in any interval $[a_j^2, b_j^2)$ of f_2 we either have $h(J) = -\infty$ for

$$J < \min(a_1^1, a_1^2)$$

or we have $h(J) = \max(f_1(b_i^1), f_2(b_j^2))$ for

$$\begin{aligned}
&J \in [\max(b_i^1, b_j^2), \min(a_{i+1}^1, a_{j+1}^2)),\\
&J \in [\max(b_{k_1}^1, b_j^2), a_{j+1}^2),\\
&J \in [\max(b_i^1, b_{k_2}^2), a_{i+1}^1),\\
&J \in [\max(b_{k_1}^1, b_{k_2}^2), K].
\end{aligned}$$

Therefore, there are no triples needed to describe the behaviour of h at these values of J. We are merely interested in the behaviour of h for any $J \in [a_i^1, b_i^1]$ or $J \in [a_j^2, b_j^2]$.
Assume that $J \in [a_i^1, b_i^1]$ for some $i = 1, \ldots, k_1$. We introduce necessary triples coming from f_1 with costs c_i^1 here. These may be interupted by other triples of f_2 having less costs, but these are ignored, because triples derived from f_2 can be evaluated analogously.
If there is a triple (a_j^2, b_j^2, c_j^2) with $c_j^2 < c_i^1$ and $a_j^2 \leq b_i^1$ and $b_j^2 - c_j^2 > a_i^1 - c_i^1$, then (a_j^2, b_j^2, c_j^2) is said to collide with (a_i^1, b_i^1, c_i^1), because we then have $f_2(J) > f_1(J)$ for some $J \in [a_i^1, b_i^1]$.
We are now looking for those subintervals of $[a_i^1, b_i^1]$ with costs c_i^1, that do not collide with any (a_j^2, b_j^2, c_j^2), which can be easily found by iterating through all given triples of f_2.

$\square$

With the last two propositions the closure of ETF follows immediately:

Corollary 4.9. *ETF is closed under functional composition and the* $\max$ *operator.*

The combination of paths yields a policy which may be described by ETFs. The path yielding a better final state is chosen, i.e. having highest battery charge at the end vertex. In order to use these functions monotonocity and extensivity need to be assured, as it was done for the functional composition in Lemma 3.7.

Proposition 4.10. *BTFs and ETFs are monotone.*

Proof. Since BTFs are the closure of SBTFs under the $\max$ operator, it is enough to show, that monotonicity is preserved under this operator:
Let $J_1 \geq J_2$ for $J_1, J_2 \in \mathbb{B}$. If f_1, f_2 are monotone BTFs, then $f_1(J_1) \geq f_1(J_2)$ and $f_2(J_1) \geq f_2(J_2)$. Furthermore $f_1(J_1) \geq \max(f_1(J_2), f_2(J_2))$ and $f_2(J_1) \geq \max(f_1(J_2), f_2(J_2))$. Therefore, $\max(f_1(J_1), f_2(J_1)) \geq \max(f_1(J_2), f_2(J_2))$.
Monotonicity of ETFs follows immediately from the monotonicity of BTFs.

□

For extensivity the method is the same:

Proposition 4.11. *ETF is extensive.*

Proof. Since ETFs are the closure of SETFs under the $\max$ operator, it is enough to show, that extensivity is preserved under this operator.
Let w_1, w_2 be two extensive ETFs, then with $s \leq_S w_1(s)$ and $s \leq_S w_2(s)$ we have $s \leq_S \max(w_1(s), w_2(s))$. Therefore, $\max(w_1, w_2)$ is extensive.

□

With the closure of ETF policies can now be mapped to ETFs, such that the ETF always yields energy state of an optimal path of the policy. For a policy m this is the ETF from the elementwise $\max$ operator of all path weights

$$w = \max \{\mathcal{W}(\pi) \mid \pi \in m\}.$$

Proposition 4.12. *The elementwise comparison $w_1 \leq_S w_2$ of two ETFs w_1 and w_2 can be realized in time $\mathcal{O}(k_1 + k_2)$ in the worst case, where k_1 and k_2 are the number of triples corresponding to the BTFs comprised by w_1 and w_2.*

Proof. Remember that a policy m_1 is 'better' than m_2 if $m_1 \leq_M m_2$. Because a higher energy state is better, we compare two ETFs elementwise here by $w_1 \leq_S w_2$, where w_i corresponds to m_i, $i \in 1, 2$. In the following, let $w_i = w_{h_i, f_i, h'_i} \in$ ETF with

$w_i : S(h_i) \to S(h'_i)$, $i \in \{1,2\}$, such that $f_1, f_2 \in$ BTF correspond with the triples $(a^1_i, b^1_i, c^1_i)_{i=1,\dots,k_1}$ and $(a^2_j, b^2_j, c^2_j)_{j=1,\dots,k_2}$.

By definition $m_1 \leq_M m_2$, if and only if $s_1 \leq_S s_2$ implies

$$\forall \pi_2 \in m_2 \, \exists \pi_1 \in m_1 : \, \mathcal{W}(\pi_1)(s_1) \leq_S \mathcal{W}(\pi_2)(s_2)$$

for all $s_1 \in \mathcal{S}(x_1), s_2 \in \mathcal{S}(x_2)$.

The definition translates to the following with respect to ETFs: If and only if $w_1(J_1, h_1) \leq_S w_2(J_2, h_2)$ for all $J_1, J_2 \in \mathbb{B}$ with $J_1 + h_1 \geq J_2 + h_2$, then $m_1 \leq_M m_2$. The negation is to find some $J_1, J_2 \in \mathbb{B}$ with $J_1 + h_1 \geq J_2 + h_2$, such that $f_1(J_1) + h'_1 < f_2(J_2) + h'_2$.

Because of monotonicity of f_1, we want to check for the least possible value of J_1, i.e. $J_1 = \min(J_2 + h_2 - h_1, K)$ if this value is greater or equal 0. If there are such J_1 and J_2, we will show for all four cases of $J_2 \in \mathbb{B}$, that we only need to compare particular function values to find appropriate J_1 and J_2.

- If $J_2 < a^1_2$, then $f_2(J_2) = -\infty$. So there is no J_1 with $f_1(J_1) + h_1 < -\infty$.

- If $J_2 \in [a^2_j, b^2_j)$, $j \in \{1, \dots, k_2\}$, then $f_2(J_2) = J_2 - c^2_j$. In this case, we need to check all triples (a^1_i, b^1_i, c^1_i) colliding with (a^2_j, b^2_j, c^2_j). If there is a triple (a^1_i, b^1_i, c^1_i) with $f_1(a^1_i) + h'_1 < f_2(a^1_i + h_1 - h_2) + h'_2$, where $a^1_i + h_1 - h_2 \in \mathbb{B}$, then $w_1 \leq_S w_2$ does not hold. Otherwise, for all $J_1 \in [a^1_i, a^1_{i+1})$ we have

$$\begin{aligned} f_1(J_1) + h'_1 &\leq f_1(a^1_i) + (J_1 - a^1_i) + h'_1 \\ &< f_2(a^1_i + h_1 - h_2) + h'_2 + (J_1 - a^1_i) \\ &\leq f_2(J_1 + h_1 - h_2). \end{aligned}$$

- If $J_2 \in [b^2_j, a^2_{j+1})$, $j \in \{1, \dots, k_2 - 1\}$, then $f_2(J_2) = b^2_j - c^2_j$. Now, J_1 can be chosen to be the least possible value, and we only need to check wether

$$f_1(\min(b_j^2 + h_2 - h_1, K)) + h_1' < b_j^2 - c_j^2 + h_2'.$$

If so, then $w_1 \leq_S w_2$ does not hold.

- If $J_2 \geq b_{k_2}^2$, then $f_2(J_2) = b_{k_2}^2 - c_{k_2}^2$. Again, J_1 can be chosen to be the least possible value, and we only need to check wether

$$f_1(\min(b_{k_2}^2 + h_2 - h_1, K)) + h_1' < b_{k_2}^2 - c_{k_2}^2 + h_2'.$$

If so, then $w_1 \leq_S w_2$ does not hold.

All in all we only need to compute particular function values. This may be done by traversing both families of triples in parallel to keep a runtime of $\mathcal{O}(k_1 + k_2)$.

□

4.3.3 Simplicity

The complexity of the model still needs to be discussed. The idea is to show simplicity in terms of Definition 3.12 and to use the property of simplicity to bound time and space consumption. Simplicity of the underlying SBSPP was already shown in Proposition 3.28. Furthermore, the cardinality of an optimal policy grows at most linearly in the number of edges. This still needs to be proven but since experimental results indicate that this cardinality is small for real road networks (see [EFS11]) a proof is left as an open question for now.

Conjecture 4.13. *The complexity of an optimal policy $m \in M_{x,y}$ in the SBPRP based on energy-optimal routing EOPP2 is linear in $|E|$ for any two vertices $x, y \in V$ in the worst case.*

Together with the previous propositions for closure under functional composition (Proposition 4.7), closure under combination (Proposition 4.8), where algorithms in $\mathcal{O}(k_1 + k_2)$ can be derived directly, and comparison in $\mathcal{O}(k_1 + k_2)$ (Proposition 4.12), all requirements for simplicity in $\mathcal{O}(|E|)$ with respect to Definition 3.12 are satisfied.

4.4 Contraction Hierarchies

There are different acceleration techniques used for the shortest path problem. One of these are contraction hierarchies (CH) introduced by Geisberger in [Gei08]. Eisner et al. have shown, that CHs can be applied to energy-optimal routing in [EFS11]. We want to use a similar strategy to generalize CHs to use partial preorder queues, such that they may solve SBPRPs. The reason, why SBSPPs are not considered here, is the necessary backward search for contraction hierarchies.
The following three steps sum up CHs here, as they were introduced in [Gei08]. Thereby, a lot of details will be omitted, because they are not necessary to adapt CHs to the given problems. The first two steps are precalculations, the third step actually performs a query.

1. **Node Ordering:**
   All vertices are sorted by a relation $\leq$, indicating the 'importance' of vertices ascending. The algorithm is correct for arbitrary orders, but the runtime heavily depends on the order. Different heuristics are linearly combined to determine an ordering. The most important priority term, says Geisberger, is the 'edge difference'. This is the difference of graph sizes before and after a simulated contraction of a vertex.

2. **Construction:**
   The construction of a CH is done by consecutively contracting vertices ordered by $\leq$. For a current vertex v, let x be a predecessor and y be a successor

of v. A 'shortcut' edge from x to y with additional information pointing to v is introduced if we do not find a 'witness' path, i.e. a path from x to y shorter than the direct path (x, v, y).

3. **Bidirectional Search:**
   A bidirectional search is performed on the contracted graph. Thereby, both searches are allowed to traverse only these edges, that lead to vertices of higher importance (described by $\leq$). That means, the forward search may use edges (x, y) with $x \leq y$ and the backward search may use edges (x, y) with $y \leq x$. The stop criterion here is *not* to have a common vertex in both search spaces, but to abort as soon as all keys in both search queues have higher distances than some tentative shortest path determined so far.

Notice that a node ordering itself is independent of the graph structure, edge costs and additional information. These may be used as heuristics, but since any node ordering is correct, we may ignore this step for now. For real road networks we may use a node ordering from common contraction hierarchies, because 'important' nodes will hardly become unimportant for slighlty extended problems.
In order to extend CHs to solve SBPRP we need to describe how and when to create shortcut edges and how to perfom bidirectional searches.

4.4.1 Shortcuts and Witnesses

When contracting a vertex in a graph, we need to introduce shortcuts, if the contracted vertex is part of an optimal path. In case of SBPRP we want to introduce shortcuts, if the contracted vertex is necessary for an optimal and complete policy (see Definition 3.9). In order to save edges and to reduce the size of the search graph, we may omit these shortcuts, if we find witnesses. A witness with respect to state-based routing then is a path, that yields better resulting states for all input states than the direct path via the contracted vertex.

Due to the extensivity and monotonicity of weights, we can use the same reasoning for correctness here as is given in [Gei08].

In common contraction hierarchies, when contracting a vertex v an edge (x, y) is replaced by its shortcut (x, y) together with information to the contracted vertex, such that the actual path can be easily unfolded. This is done only, if the costs of the path (x, v, y) is strictly less than the path (x, y). Otherwise (x, y) would be a witness. To get the same behaviour for state-based routing, we want to use policies representing a set of paths. The same formulation can then be used: If we have two policies $m_{x,v}$ from x to v and $m_{v,y}$ from v to y as well as a policy $m_{x,y}$ from x to y, we introduce a shortcut replacing (x, y) by the policy $m = m_{x,v} \cup m_{v,y}$, if $m \leq_M m_{x,y}$ and not $m_{x,y} \leq_M m$. If we have $m_{x,y} \leq_M m$, we do not need to create a shortcut, because $m_{x,y}$ always yields better resulting states.

But there is a third case: If both policies are incomparable, i.e. neither $m_{x,y} \leq_M m$ nor $m \leq_M m_{x,y}$, then both policies are required, because either policy may yield a resulting state for some particular input state strictly better than for the other policy. In this case, we want to merge both policies by combination.

For energy-optimal routing, this combination corresponds to the $\max$ operator (see Figure 3.7). Eisner et al. introduce multiple edges in this place to represent policies, but for energy-optimal routing we can also use energy transformation functions, as we have seen in Section 4.3. Even though, they say, that profile searches are "relatively time- and space-consuming" [EFS11], they are missing reasonings or experimental results here.

Another point we have not yet addressed is the reference to the contracted vertex. For common contraction hierarchies, a shortcut edge references to the unique contracted vertex. For state-based routing we use multiple edges or policies, so we may have different 'sources'. We may get a shortcut that is the merging of two shortcuts, both with distinct references. In this case, we need to keep track of all such vertices. In the worst case, we then need to check all of these reference in order to unfold a path for a specific starting state. In the best case, we have a mapping $\mathcal{S}(x) \to \bigcup_{v \in V} M_{x,v} \times M_{v,y}$ from starting states to both subpolicies of a contracted vertex v. For energy-optimal routing this is simple, because we just need to keep track of the contracted vertices for each triple (a_i, b_i, c_i) representing the ETF (see Definition 4.1 and 4.2).

4.4.2 Bidirectional Search

The bidirectional search is based on concatenating, combining and comparing policies. Because concatenation is based on functional composition of state transformation functions and because function composition is associative, the bidirectional search works in the same way as it has been described by Eisner et al. [EFS11]. Details of bidirectional searches will be omitted here, but the stop criteria is an important aspect to discuss.

Bidirectional searches usually stop when both search spaces meet somewhere inbetween at vertex v. This is reasonable because the search space around the starting vertex x contains all vertices of some distance d to x and the search space around the end vertex y contains as well all vertices of distance d to y. All other paths from x to y are longer than $2 \cdot d$.

Regarding contraction hierarchies it is not a valid assumption to stop the bidirectional search when both search spaces meet. This is described in detail by Geisberger [Gei08]. The search can be stopped as soon as both search queues do not contain a vertex of distance less than of a tentative shortest path from start to end found so far.

Because SBPRP uses partial preorders the bidirectional search may be stopped only if a complete and optimal policy was found. This is due to the definition of state-based profile routing. The implementation of the elementary operations of concatenation and combination need to ensure, that only optimal policies are constructed in the search graph. So in theory the optimality of policies can be assumed to be given at all times. A complete policy is found as soon as a tentative policy can not be improved by any other policy in the remaining search queues. In order to check this criterion the partial preorder queues must provide a method to deliver all minimal elements in the queue, because each minimal element may probably improve the tentative policy found so far. This method is given by 'front', which is discussed in Section 4.1.

5 Conclusions

Green Routing can be approached in many different ways. Chapter 3 presents different models handling different aspects of Green Routing. On of those models is the state-based shortest path problem using an arbitrary set of states and having edges labelled with state transformation functions. This model is capable of comprising most other routing problems, the only required properties are monotonicity and extensivity with respect to a partial preorder on the set of states.

Two types of state-based routing is presented, the SBSPP and the SBPRP. The former queries for optimal path for a specific starting state. If the preorder on the states is total, then Dijkstra's Algorithm and A^* may be applied. Otherwise a partial preorder queue is required to solve the problem. The latter queries for a complete policy describing optimal paths for all possible starting states. The size of such policies heavily influence the efficiency of algorithms designed for the SBPRP.

This thesis focused on proving that the energy-optimal path problem introduced by Sachenbacher et al. in [SLAH11] can be reformulated as a state-based problem. Simple battery transformation functions were introduced to represent the change of the battery charge after traversing edges. These were monotone and closed under functional composition. Unfortunately they were not extensive which is required for the state-based approach. A potential function naturally given by the altitudes of the locations represented by vertices are used similarly as in [SLAH11]. Using such potential functions yields simple energy transformation functions.

Section 3.8 presents a model combining battery constraints from energy-optimal routing with stochastic aspects. Unfortunately these models do not perfectly fit into the state-based model, but finding a path with optimal cumulative distribution functions (CDFs) is easily done in theory. In practice approximations are needed to compute the CDFs and their convolutions (together with a derivation to keep the result a CDF). Different approximation techniques still need to be implemented and tested.

In order to use the state-based energy-optimal path problem for profile routing a representation of policies is required. Eisner et al. [EFS11] have done this by using multiple edges such that they could apply the acceleration technique of contraction hierarchies introduced by Geisberger [Gei08]. Another approach using battery and energy transformation functions was presented, but a comparison of time and space efficiency still needs to be done. Eisner et al. assume their approach to be more efficient in both aspects.

A Java 1.6 implementation of the model layer consisting of generic interfaces was presented in Section 4.2. These interfaces are implemented to realize specific models, such as the energy-optimal path problem. Algorithms can be designed on top of this structure on interfaces as presented in the respective section.

Different acceleration techniques are known for solving the shortest path problem. In order to generalize these to solve state-based problem, an important datatype - the partial preorder queue - is presented in Section 4.1. Using this type of priority queue contraction hierarchies are generalized to solve the state-based profile routing problem. This generalization is done analogously to the approach of [EFS11]. Other techniques might be generalized in a similar way.

5.1 Open Problems

There are different aspects not yet discussed in full detail. Section 3.8 presents a model combining battery constraints from energy-optimal routing with stochastic aspects. It may be worth to combine all four different aspects of time-dependency, energy-optimality, stochastic routing and the shortest weight-constrained path problem representing multiple target functions (see Chapter 3). This may probably be a sensible model for multimodal routing, i.e. routing not only on electric vehicles but on a set of different types of transportations overcomming the problem of a limited reach of eletric vehicles.

Furthermore, in this chapter we have seen, that the conditional expectation can not be used in a state-based approach directly. Nevertheless SEOPP2 does take more information into account than MLESPP itself. It was left as an open question to design a choice function comparing CDFs not only by a specific value (e.g. the success for MLESPP) but in a way similar to the expectation operator still consistent with the comparison of CDFs.

Another important aspect is fleet routing and fleet scheduling. So far routing was considered in an egoistic manner to find optimal paths just for one vehicle. Providing routing directions to balance the load in order to avoid the upcomming of congestion completely may be more efficient than giving egoistic routing directions. Nevertheless, energy-optimal routing may indirectly cause the same effect in a self-organizing way.

Section 4.1 introduces partial preorder queues. There are already different implementations for priority queues using a total (pre)orders, but efficient implementations for partial preorder queues are still open for research. As it has been already mentioned in the discussion of these queues, an efficient implementation might exploit additional information. One such information could the average rate different methods are used.

Contraction hierarchies solving the state-based profile routing problem were presented in Section 4.4. Other acceleration techniques may be generalized in a similar fashion using partial preorder queues. An interesting technique for profile routing could be transit node routing discussed by Bast et al. in [BFM$^+$07].

As previously mentioned defining a distinguished garbage state when modeling state-based path problem is natural in most cases. This was also implicitely done for energy-optimal routing where $-\infty$ was such a garbage state. This state represents that the target is not reachable through the given path even though this path actually connects start and end vertex.
The use of kinetic energy was not further discussed because it is not possible to implement kinetic energy in the state-based models directly. Each road junction would need to be expanded in order to model additional energy costs due to breaking and acceleration at the road junctions. These costs can be seen as a property of pairs of adjacent edges which is not handled by the presented models, but is an interesting problem if such space-consuming expansions can be saved by generalizing shortest path algorithms or the state-based model even further.

5.2 Future Work

A thorough implementation of contraction hierarchies solving state-based profile routing is still open for future work. If this algorithms shows to be efficient in a series of tests, then it will be imported into the GreenNav system.
The implementation is to be evaluated with respect to time and space consumption and correctness. In case of a stochastic model where approximations should be used for effiency, approximation quality needs to be evaluated. Thereby different discretization techniques should be compared.
Another aspect for test is the actual benefit of using a complex model. Kono et al. [KFKN08] save about nine percent of energy by not taking the shortest but energy-efficient route in their experiment, but more empirical data should be analyzed for a differentiated point of view. Another approach is to directly compare energy-consumption on shortest and time-efficient paths to energy-efficient paths.

A Abstract Data Types

For completeness the signatures of booleans and sets together with axiomatic properties are presented here as they are formulated by Stümpel [Stü11]. They are used for a formal specification of partial preorder queues in Section 4.1. The axioms are formulated using first-order logic.

```
specification BOOL =
 sorts                                 bool
 constants              true     :     bool
                        false    :     bool
 operations             not      :     bool → bool
                        and      :     bool × bool → bool
                        or       :     bool × bool → bool
                        implies  :     bool × bool → bool
                        is       :     bool × bool → bool
 variables              x        :     bool
 axioms
                     true        ≠     false,
                not(true)        =     false,
               not(false)        =     true,
               true and x        =     x,
              false and x        =     false,
                true or x        =     true,
               false or x        =     x,
          false implies x        =     true,
           true implies x        =     x,
                   x is y        =     (x implies y) and (y implies x).
end
```

Figure A.1: The syntax and axioms of datatype describing boolean values. Notice that the infix notation is used for boolean operators.

specification SET =

based on			BOOL
sorts			**set**, **value**
constants	$\emptyset$	:	**set**
operations	$\{\cdot\}$	:	**value** $\rightarrow$ **set**
	$\cup$	:	**set** $\times$ **set** $\rightarrow$ **set**
	$\cap$	:	**set** $\times$ **set** $\rightarrow$ **set**
	$\setminus$	:	**set** $\times$ **set** $\rightarrow$ **set**
	$\subseteq$	:	**set** $\times$ **set** $\rightarrow$ **bool**
	$\equiv$	:	**set** $\times$ **set** $\rightarrow$ **bool**
	$\in$	:	**value** $\times$ **set** $\rightarrow$ **bool**
variables	A, B	:	**set**
	x, y	:	**value**

axioms

- $x \in \emptyset$ = false,
- $x \in \{x\}$ = true,
- $x \in A \cup B = x \in A \vee x \in B$,
- $x \in A \cap B = x \in A \wedge x \in B$,
- $x \in A \setminus B = x \in A \wedge \neg(x \in B)$,
- $A \subseteq B$ = true, if and only if $x \in A \rightarrow x \in B$ = true for all values x,
- $A \equiv B = A \subseteq B \wedge B \subseteq A$.

end

Figure A.2: The syntax and axioms of an abstract datatype describing sets. The operation $\{\cdot\}$ is used to describe singleton sets.

References

[AHLS10] Andreas Artmeier, Julian Haselmayr, Martin Leucker, and Martin Sachenbacher, *The Shortest Path Problem Revisited: Optimal Routing for Electric Vehicles*, KI 2010: Advances in Artificial Intelligence, vol. 6359, Springer Berlin / Heidelberg, 2010, pp. 309–316.

[BFM$^+$07] Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes, *In Transit to Constant Time Shortest-Path Queries in Road Networks*, Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithmics and Combinatorics, SIAM, 2007, pp. 46–59.

[CW03] Ken Caldeira and Michael E. Wickett, *Oceanography: Anthropogenic carbon and ocean pH*, Nature 425 (2003), pp. 365–368.

[DGRW11] Daniel Delling, A.V. Goldberg, I. Razenshteyn, and R.F. Werneck, *Graph Partitioning with Natural Cuts*, Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International, 2011.

[Dij59] Edsger W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik **1** (1959), no. 1, pp. 269–271.

[DW09] Daniel Delling and Dorothea Wagner, *Time-Dependent Route Planning*, Robust and Online Large-Scale Optimization, LNCS, Springer, 2009.

[EFS11] Jochen Eisner, Stefan Funke, and Sabine Storandt, *Optimal Route Planning for Electric Vehicles in Large Networks*, AAAI, 2011.

[Gei08] Robert Geisberger, *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks*, Diploma thesis, Karlsruhe Institute of Technology, 2008.

[GJ90] Michael R. Garey and David S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1990.

[Gle10] Peter H. Gleick[1], *Climate Change and the Integrity of Science*, Science **328** (2010), no. 5979, pp. 689–690.

[GO99] Roche A. Guérin and Ariel Orda, *QoS routing in networks with inaccurate information: theory and algorithms*, IEEE/ACM Trans. Netw. **7** (1999), pp. 350–364.

[Jok66] Hans C. Joksch, *The shortest route problem with constraints*, Journal of Mathematical Analysis and Applications (1966), pp. 191–197.

[Kar72] Richard M. Karp, *Reducibility Among Combinatorial Problems*, Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, eds.), Plenum Press, 1972, pp. 85–103.

[KFKN08] Toshiaki Kono, Takumi Fushiki, Asada Katsuya, and Kenjiro Nakano, *Fuel Consumption Analysis and Prediction Model for "Eco" Route Search*, ITS Connections: Saving Time. Saving Lives, 15th World Congress on Intelligent Transport Systems and ITS America's 2008 Annual Meeting, 2008.

[Klu10] Sebastian Kluge, *On the computation of fuel-optimal paths in time-dependent networks*, Ph.D. thesis, Technische Universität München, 2010.

[1]Full citation available at `http://www.sciencemag.org/content/328/5979/689/suppl/DC1`.

[Moo09] Frances C. Moore, *Climate Change and Air Pollution: Exploring the Synergies and Potential for Mitigation in Industrializing Countries*, Sustainability 1 (2009), pp. 43–54.

[MS10] Kurt Mehlhorn and Peter Sanders, *Algorithms and Data Structures: The Basic Toolbox*, 1st ed., Springer Publishing Company, Incorporated, 2010.

[Neu10] Sabine Neubauer, *Planung energieeffizienter Routen in Straßennetzwerken*, Diploma thesis, Karlsruhe Institute of Technology, 2010.

[OR90] Ariel Orda and Raphael Rom, *Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length*, J. ACM **37** (1990), pp. 607–625.

[PT96] George H. Polychronopoulos and John N. Tsitsiklis, *Stochastic shortest path problems with recourse*, Networks **27** (1996), no. 2, pp. 133–143.

[Sch02] Bernd S. W. Schröder, *Ordered Sets: An Introduction*, Birkhäuser, 2002.

[Sch08] Dominik Schultes, *Route Planning in Road Networks*, Ph.D. thesis, Karlsruhe Institute of Technology, 2008.

[SLAH11] Martin Sachenbacher, Martin Leucker, Andreas Artmeier, and Julian Haselmayr, *Efficient Energy-Optimal Routing for Electric Vehicles*, AAAI Conference on Artificial Intelligence, Special Track on Computational Sustainability, AAAI, 2011, to appear.

[Sni06] Moshe Sniedovich, *Dijkstra's Algorithm Revisited: the Dynamic Programming Connexion*, Journal of Control and Cybernetics, 35(3) (2006), pp. 599–620.

[Stü11] Annette Stümpel, *Spezifikation und Modellierung*, Lecture Notes, University of Lübeck, 2011, retrieved from `http://www.isp.uni-luebeck.de/teaching/lecture/current/V_Modellierung_de.htm` on October 2011.

[UUN+09] Suleyman Uludag, Ziyneti E. Uludag, Klara Nahrstedt, King-Shan Lui, and Fred Baker, *A laplace transform-based method to stochastic path finding*, Proceedings of the 2009 IEEE international conference on Communications (Piscataway, NJ, USA), ICC'09, IEEE Press, 2009, pp. 1319–1323.

[WBP11] Dorothea Wagner, Reinhard Bauer, and Thomas Pajor, *Algorithmen für Routenplanung*, Lecture Notes, Karlsruhe Institute of Technology, 2011, retrieved from `http://i11www.iti.uni-karlsruhe.de/teaching/sommer2011/routenplanung/index` on October 2011.