The optimal routing problem in the context of battery-powered electric vehicles

Andreas Artmeier, Julian Haselmayr, Martin Leucker, Martin Sachenbacher

Technische Universität München, Department of Informatics
Boltzmannstraße 3, 85748 Garching, Germany
{artmeier,haselmay,sachenba,leucker}@in.tum.de

May 6, 2010

Abstract

Electric vehicles (EV) powered by batteries will play a significant role in the road traffic of the future. The unique characteristics of such EVs – limited cruising range, long recharge times, and the ability to regain energy during deceleration – require novel routing algorithms, since the task is now to determine the most economical route rather than just the shortest one. This paper proposes extensions to general shortest-path algorithms that address the problem of energy-optimal routing. Specifically, we (i) formalize energy-efficient routing in the presence of rechargeable batteries as a special case of the constrained shortest path problem (CSP) with hard and soft constraints, (ii) present an adaption of a general shortest path algorithm (using an energy graph, i.e., a graph with a weight function representing the energy consumption) that respects the given constraints and has a worst case complexity of $O(n^3)$, and (iii) show that acceleration and deceleration costs for segments with different cruising speed can be taken into account by an unfolding of a weighted routing graph. The presented algorithms have been implemented and evaluated within a prototypic navigation system for energy-efficient routing.

1 Introduction

Dwindling fossil fuel reserves and the severe consequences of climate change are major challenges of the new millennium. Electric vehicles (EVs) offer a potential contribution: they can be powered by regenerative energy sources such as wind and solar power, and they can recover some of their kinetic and/or potential energy during deceleration phases. This so-called recuperation or regenerative braking increases the cruising range of current EVs by about 20 percent in typical urban settings, and often more in hilly areas. However, a more wide-spread use of EVs is still hindered by limited battery capacity, which currently allows cruising ranges of only 150 to 200 kilometers. Thus, accurate prediction of remaining cruising range and energy-optimized driving are important issues for EVs in the foreseeable future.

The unique characteristics of EVs have an impact on search algorithms used in navigation systems and route planners. With the limited cruising range, long recharge times, and energy recuperation ability of battery-powered EVs, the task is now to find energy-efficient routes, rather than just fast or short routes. This modification might appear insignificant at first glance, as it seems enough to simply exchange time and distance values with energy consumption in the underlying routing problem. But on a second glance, several new challenges surface, which require novel algorithms that go beyond existing solutions for route search in street networks.

For example, consider the simple vehicle routing problem shown in Figure 2. The nodes correspond to locations and the edges correspond to road segments. The positive (resp. negative) values at the edges denote consumption (resp. gain) of energy, which we assume is taken from (resp. stored in) a battery with maximum capacity C_{max}. There are two possible paths from source s to destination t: $(s \xrightarrow{2} x \xrightarrow{-1} t)$ and $(s \xrightarrow{-1} y \xrightarrow{2} t)$, both of which are optimal from a classical shortest-path perspective as they have the same overall cost of 1 energy unit. However, if at node s there is only $C_s = 1$ unit of energy left in the battery, the path via y becomes the only feasible path. If instead we start with a battery charged to $C_s = C_{max} \geq 2$ at node s, both paths to t are feasible, but the path via x is now preferable since it has a lower overall energy consumption (the other path via y will require 2 energy units, since recuperation is no longer possible in the segment $s \xrightarrow{-1} y$).

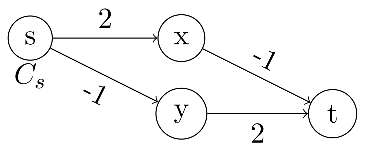


Figure 1: A simple electric vehicle routing problem

In this work, we address the problem of finding the most energy-efficient path for battery-powered electric cars with recuperation in a graph-theoretical context. This problem is similar to the *shortest path problem* (SP), which consists of finding a path P in a graph from a source vertex s to a destination vertex t such that $c(P) = \min_{Q \in U}(c(Q))$, where U is the set of all paths from s to t, and $c : E \to \mathbb{Z}$ is a weight function on the edges E of the graph. SP is polynomial; the best known algorithm for the case of non-negative edges is Dijkstra [Dij59] with time complexity $O(n^2)$, while in the general case, Bellman-Ford [Bel58], [LRF56] has $O(n^3)$. However, most commonly used SP algorithms like contraction hierarchies [GSSD08], highway hierarchies [SS05] and transit vertex routing [BFSS07] can't be applied to our problem because of the negative weights due to recuperation. In addition, SP does not consider the constraints that result from the discharge and recharge characteristics of the EV's battery pack, namely that it neither can be discharged below zero, nor charged above its maximum capacity. An extension of the SP, the *constrained shortest path problem* (CSP) [Jok66], is to find a shortest path P from s to t among all feasible paths in a graph, where a path P is called feasible if $b(P) \leq T \in \mathbb{N}$ for an additional weight function $b : E \to \mathbb{N}$. Our problem of energy-efficient routing with recuperation can be framed as such a CSP, but CSP is known to be NP-complete [MD79]. In a second step, we extend our method to also take into account dynamic energy costs originating from acceleration and deceleration due to changes of the road's speed limit.

In the following, we (i) show how to model energy-efficient routing in the presence of rechargeable batteries as a special case of a CSP, extending the shortest path problem with problem-specific hard and soft constraints, (ii) present a novel variant of a general shortest path algorithm that respects these constraints, but still has a polynomial worst case time complexity of $O(n^3)$, and (iii) show that acceleration and deceleration costs for segments with different cruising speed can be taken into account by an unfolding of a weighted routing graph.

2 Preliminaries

We assume that a road network can be modeled as a directed graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. A *path* P of *length* $|P| = k$ is then a sequence of $k+1$ distinct vertices $(v_1, v_2, \ldots, v_{k+1})$ with $(v_i, v_{i+1}) \in E$ for all $i \in \{1, 2, \ldots, k\}$. In addition, we assume that a weight function $c : E \to \mathbb{Z}$ models the amount of energy required or gained when traveling along the edges in the network. The *weight* $c(P)$ of a path P of length k is then defined by $c(P) = \sum_{i=1}^{k} c(v_i, v_{i+1})$. A cycle $C = (v_1, v_2, \ldots, v_k, v_1)$ is called *positive* if $c(C) \geq 0$. For a road network, the absence of negative

Algorithm 2.1: ConstrainedGenericShortestPath

```
input: A directed graph $G = (V, E)$, weight function $c : E \to \mathbb{Z}$, source vertex $s$, strategy
       $S$$\underline{\text{, maximum capacity } C_{max} \text{ and initial } U_s}$
output: A $\underline{\text{prefix bounded}}$ shortest path tree from $s$ $\underline{\text{with respect to absorp}}$
begin
    foreach vertex $v$ in $V$ do
        $d(v) \leftarrow \infty$;
        $p(v) \leftarrow$ null;
    $d(s) \leftarrow 0\underline{+U_s}$;
    $Q \leftarrow \{s\}$;
    while $Q \neq \emptyset$ do
        choose $u$ from $Q$ with strategy $S$;
        $Q \leftarrow Q \setminus \{u\}$;
        foreach successor $v$ of $u$ do
            $d' \leftarrow d(u) + c(u, v)$;
            $\underline{d' \leftarrow \max(d', 0)}$;
            if $d' < d(v)$ $\underline{\text{and } d' \leq C_{max}}$ then
                $d(v) \leftarrow d'$;
                $p(v) \leftarrow u$;
                $Q \leftarrow Q \cup \{v\}$;
end
```

cycles corresponds to the law of conservation of energy.

Algorithm 2.1 ([GP86, BCR93]) without the underlined statements – these show our changes to be explained in the next sections – computes a shortest path tree for a given source vertex and expansion strategy, where the shortest path tree is a directed tree that represents the solution of all shortest path problems with this source vertex, and the strategy determines the order in which vertices are processed. The algorithm extends and improves a tree defined by function p, where $p(v)$ is the predecessor of v in the tree. The distance $d(v)$ is the weight $c(P)$ of the path P from s to v in this tree. The algorithm improves these distance values $d(v)$ for $v \in V$ until the queue Q is empty, and the best path (according to the distance value) to every reachable vertex is found. Note that the algorithm does not terminate if a cycle with negative weight is reachable from the source vertex.

The time complexity of the algorithm with Dijkstra's strategy – choose the vertex with smallest distance value from Q – is known to be $O(n^2)$ for positive weights but exponential for the general case [Joh73], while using the Bellman-Ford strategy – choose the vertices in a FIFO manner from Q – results in time complexity $O(n^3)$ for arbitrary weights.

3 Prefix-bounded Shortest Paths

Our goal is to find an energy efficient route for a battery-powered EV in a road network with given energy values, modeled as a directed graph $G = (E, V)$ with weight function $c : E \to \mathbb{Z}$.

Clearly, route segments are only feasible if the required energy does not exceed the charging level of the battery. In addition, due to elevation differences or changes in the cruising speed, on each route there can be road sections where the energy consumption is negative, such that the EV's battery can be recharged. However, we also have to consider that the battery has a certain maximum capacity, beyond which recharging is no longer possible. The two conditions on the charge level of the battery—it cannot be discharged below zero, it cannot store more energy than its maximum capacity—can be viewed as *hard* and *soft constraints*, respectively, on possible routes: a route is infeasible if there is a point where the required energy exceeds the charge level, and a route is less preferred if there is a point where energy could be recuperated but the battery's maximum capacity is exceeded.

For modeling these constraints, some more definitions are needed. Let C_{max} be the maximum battery capacity, C_{v_i} the charge level of the battery at vertex v_i, and $U_{v_i} = C_{max} - C_{v_i}$ the remaining storage capacity of the battery at vertex v_i. Then for a path $P = (v_1 \to v_2 \to \cdots \to v_k)$ with $v_1 = s$ in G, we define the *absorption function* absorp recursively as

$$\text{absorp}(P^k) = \begin{cases} U_{v_1} & \text{if } k = 1, \\ \max\left(\text{absorp}(P^{k-1}) + c(v_{k-1}, v_k)\right., 0) & \text{if } k > 1 \end{cases}$$

where $P^i = (v_1 \to v_2 \to \cdots \to v_i)$ is the subpath of P ending in vertex v_i (with $i \leq k$). Moreover, we call path P *prefix-bounded by C_{max}*, if $\text{absorp}(P^i) \leq C_{max}$ holds for every $i = 1, 2, \ldots, k$.

The problem we want to solve can then be described as follows: given a start point and battery charge level, find a route to a target point that respects the constraints and where the remaining charge is highest. We call this problem the *prefix-bounded shortest path problem*, or for short PBSP. A prefix-bounded path P corresponds exactly to a route which is feasible to use ($0 \leq C_{v_i}, U_{v_i} \leq C_{max}$) and where $\text{absorp}(P)$ is the remaining storage capacity at the end of the route. Formally, a solution to a PBSP is a path $P \in W$ such that

$$\text{absorp}(P) = \min_{Q \in W}(\text{absorp}(Q))$$

where W is the set of all paths w from s to t for which the constraint $w^i \leq C_{max} - U_s$ holds for all $i \in \{1, 2, \ldots, (|w|+1)\}$. Since maximizing the remaining battery charge is equivalent to minimizing the remaining battery capacity, we can search for a prefix-bounded shortest path tree with respect to function absorp in order to get a solution to the PBSP.

If we only consider graphs with non-negative edges, the PBSP is a special case of a CSP [Jok66], where the two weight functions b and c are equivalent. However, algorithms for CSP are exponential in the worst case, and they do not allow for negative weights. Therefore, we instead propose an algorithm that extends the generic shortest path algorithm by taking the prefix-bound constraints into account. As we will see, the additional prefix-bound constraints don't change the time complexity of the shortest path problem, and thus this algorithm solves the problem in polynomial time ($O(n^3)$ with the Bellman-Ford strategy).

4 Computing Prefix-Bounded Shortest Path Trees

In this section, we modify the generic shortest path algorithm (see Section 2) to compute a prefix-bounded shortest path tree with respect to function absorp, that is, to solve a PBSP. The modifications are shown as underlined statements in Algorithm 2.1. One modification concerns the initialization of the source vertex with the difference between the maximum capacity of the battery and its current charge (line 5). The other modifications implement the soft constraint that the battery can't be overcharged (line 12) and the hard constraint prohibiting the use of road sections with insufficient energy (line 13).

Theorem 4.1. *Algorithm 2.1 computes for every strategy S a prefix-bounded shortest path tree with respect to function* absorp*.*

A detailed proof of correctness for the algorithm can be found in the appendix. In [ZN00], several expansion strategies are presented for solving the general shortest path problem. However, in the context of electromobility, novel ones are needed that take into account the specific energy graph properties, e.g., far fewer negative than positive edge weights. As a first variant, we propose the following expand strategy. For each vertex, a counter is initialized with 0 and incremented each time this vertex is chosen for expansion in line 8; the strategy then chooses a vertex from Q with the smallest counter. This strategy leads to the same time complexity as the FIFO strategy does for the Bellman-Ford algorithm in the general shortest path problem (see [BCR93] for instance).

Theorem 4.2. *Algorithm 2.1 using strategy expand has time complexity $O(n^3)$.*

Lemma 4.3. *Algorithm 2.1 using strategy dijkstra has time complexity $O(n^2)$ if c is non-negative.*

Now we consider the strategy expand-distance: choose the vertex u from Q which has smallest $d(u)$ among vertices in Q which are expanded least of all. This new strategy, combining both previously presented strategies, guarantees time complexity $O(n^3)$ in any case. In addition, if c is non-negative, we get $O(n^2)$ since at the beginning of the algorithm expand$(u) = 0$ and no vertex is inserted twice into Q. Therefore the following key statement is proven.

Corollary 4.4. *Algorithm 2.1 using strategy expand-distance has time complexity $O(n^3)$ for arbitrary weight function c and $O(n^2)$ if c is non-negative.* □

Since in our graph modeling the energy values there are typically only few edges with negative weight, the time complexity of the expand-distance strategy can be expected to be near $O(n^2)$, and in contrast to the conventional Dijkstra algorithm we are sure to be far away from exponential time. These two advantages are the reason why we suggest Algorithm 2.1 using strategy expand-distance in the context of routing for electromobility.

5 The Energy Graph

The routing algorithm of the last section requires a graph representing the energy consumption or recuperation of the road network. For a given graph $G = (V, E)$, energy function F and velocity function $g : E \to \mathbb{N}$ this extended graph, called energy graph, is calculated. Note that the value $F(e)$ of an edge e can only be evaluated in the context of a path because the second argument of F is the velocity of its preceding edge. This characteristic distinguishes the energy efficient routing problem from the task of determining the fastest or shortest path which can be calculated locally with only the source and destination vertices for each edge. While it is easy to determine the distance of a road section and the average time to pass it, the calculation of the energy costs depends on many parameters. They consist of the street surface, the inclination of the road, parameters of the car and the weather conditions. Therefore, the collection of these parameters poses a practical challenge beside the required mathematical theory.

The graph in Figure 2a represents a theoretical routing problem. The labels over the edges represent its velocity values in km/h. Consider the problem of determining the energy consumption of the edge $v \xrightarrow{50} w$. The driver has to decelerate in this section if he uses the path $u \xrightarrow{100} v \xrightarrow{50} w$ and thus the accumulator recharges or he has to accelerate on the path $x \xrightarrow{30} v \xrightarrow{50} w$ and therefore energy is consumed. This problem is solved by introducing a vertex for every distinct velocity value of the predecessors of v. The rest of the section explains and analyzes this idea in depth.

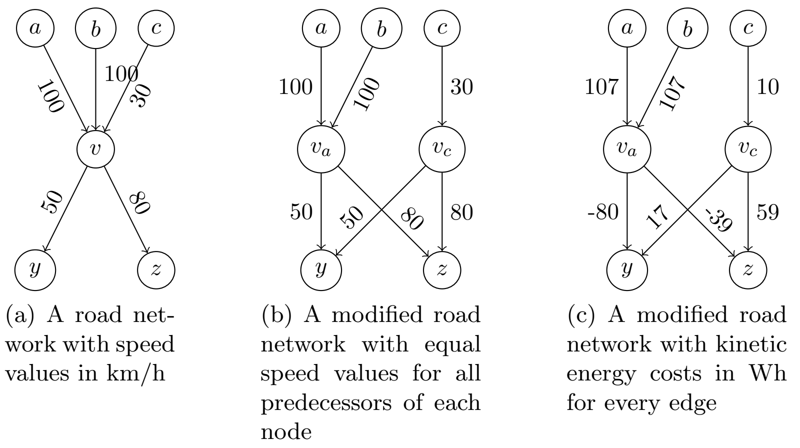

a
b
c
100
100
30
v
50
80
y
z
(a) A road network with speed values in km/h
a
b
c
100
100
30
v_a
v_c
50
50
80
80
y
z
(b) A modified road network with equal speed values for all predecessors of each node
a
b
c
107
107
10
v_a
v_c
-80
17
-39
59
y
z
(c) A modified road network with kinetic energy costs in Wh for every edge

Figure 2: The calculation process of algorithm 5.1

The input of Algorithm 5.1 is a directed graph $G = (V, E)$, $g : E \to \mathbb{N}$ and $F : E \times \mathbb{N} \to \mathbb{Z}$ which determine every edge's velocity and energy consumption. The algorithm generates a modified directed graph $G' = (V', E')$ with a weight function $c : E \to \mathbb{Z}$ providing the energy consumption for every edge independently from its predecessor. The function $M : V' \to V, \quad v_u \mapsto v \quad \forall v, u \in V$ enables us to determine the path in G corresponding to the algorithm's output path in G'.

We analyze the central idea with respect to our example. The vertex v is replaced by v_a and v_c. The node v_a compromises the incoming edges from a and b with velocity 100 while v_c governs the edge from c with velocity 30. No additional vertices are required because the remaining vertices have no incoming edges with different velocities.

Theorem 5.1. *Algorithm 5.1 has time complexity $O(n^3)$ and generates a directed graph with weights representing the energy consumption with respect to the function absorp.*

Note that road networks have an average degree of approximately 3 and therefore the algorithm needs only $O(n)$ steps in real settings.

Theorem 5.2. *Let $\Delta(v)$ for $v \in V$ be the number of outgoing edges in graph G and $\omega(v) = |\{c(u,v)|\exists(u,v) \in E\}|$ be the number of different weights of the incoming edges of $v \in V$. Graph $G' = (V', E')$ generated by algorithm 5.1 has then order $|V'| = n' = \sum_{v \in V} \min(\omega(v), 1)$ and size $|E'| = m' = \sum_{v \in V}(\min(\omega(v), 1)\Delta(v))$.*

Corollary 5.3. *The order n' and size m' of the graph $G' = (V', E')$ generated by algorithm 5.1 have complexity $n' = O(n^2)$ and $m' = O(n^3)$.*

Corollary 5.3 follows immediately from $\Delta(v) \leq n$ and $\omega(v) \leq n \; \forall v \in V$. Note that in real road networks $\Delta(v)$ and $\omega(v)$ are bounded by the number of streets connected at a junction which usually won't exceed 4. Hence, $n' = O(n)$ and $m' = O(n)$.

6 Prototypic Implementation and Experimental Results

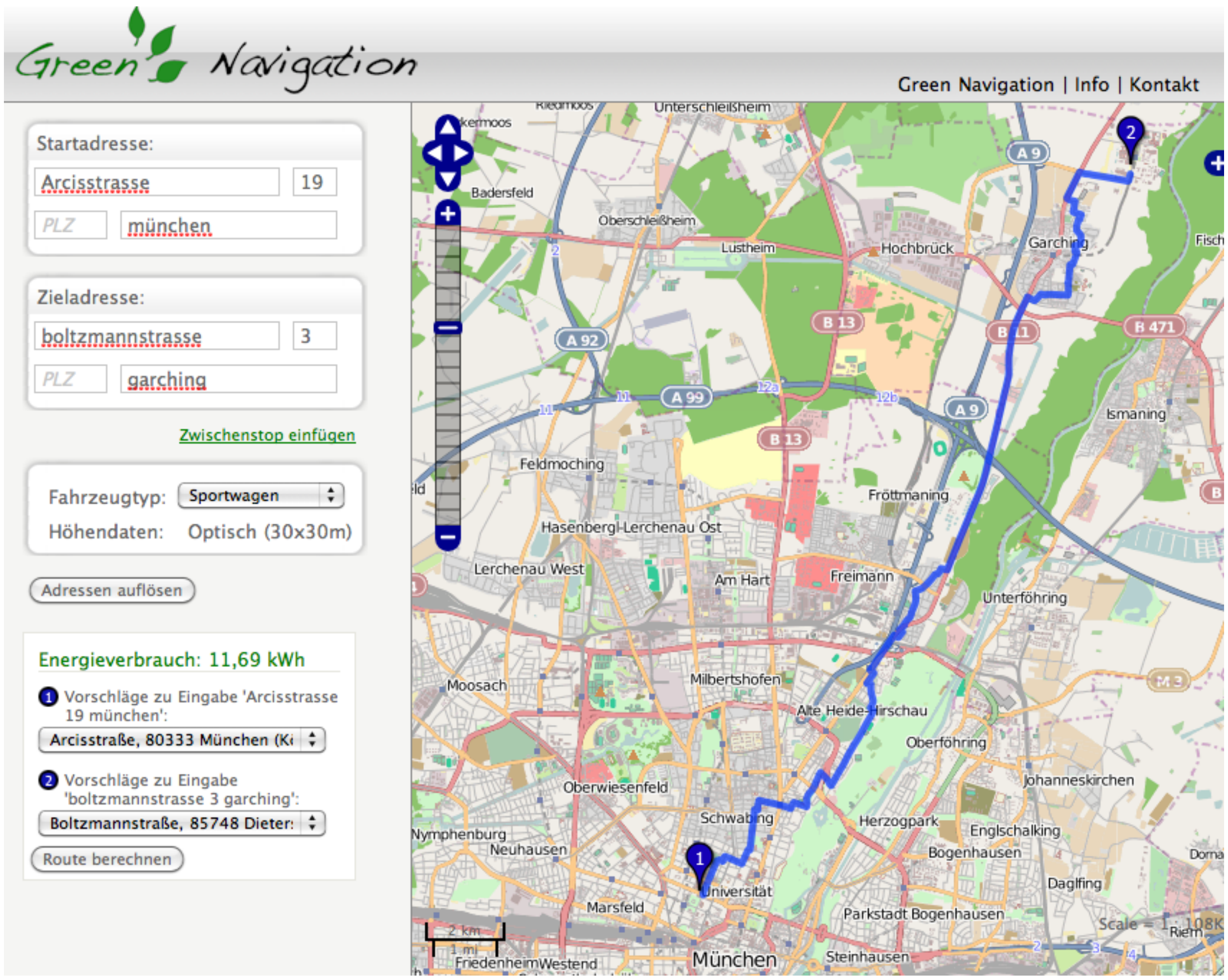
We developed a prototypic software system for energy efficient routing, based on opensource libraries and freely available data. It is possible to access this system online (www.greennav.org). For a given car type, source address and destination address, the system computes a route with minimum

Algorithm 5.1: EnergyGraph

```
   input: Directed graph $G = (V, E)$, velocity $g : E \to \mathbb{Z}^+$ and energy function $F : E \times \mathbb{N} \to \mathbb{Z}$
   output: Directed graph $G' = (V', E')$ with weight function $c : E' \to \mathbb{Z}$
1  begin
2      $G' = (V', E') \leftarrow (V, E)$;
3      foreach $v$ in $V$ do
4          $L \leftarrow \emptyset$;
5          foreach predecessor $u$ of $v$ in $G'$ do
6              if there is a vertex $u' \in L$ with $g(u, v) = g(u', v)$ then
7                  $E' \leftarrow E' \cup \{(u, v_{u'})\}$;
8              else
9                  $L \leftarrow L \cup \{u\}$;
10                 $V' \leftarrow V' \cup \{v_u\}$;
11                 $F\big((u, v_u), \cdot\big) \leftarrow F\big((u, v), \cdot\big)$;
12                 foreach successor $w$ of $v$ in $G'$ do
13                     $E' \leftarrow E' \cup \{(v_u, w)\}$;
14                     $g(v_u, w) \leftarrow g(v, w)$;
15                     $c(v_u, w) \leftarrow F\big((v, w), g(u, v)\big)$;
16         if $v$ has predecessors then
17             $V' \leftarrow V' \setminus \{v\}$;
18 end
```

energy costs. The data basis consists of the collaborative project OpenStreetMap (OSM), which aims to create and distribute freely available geospatial data, and the altitude map of the NASA Shuttle Radar Topographic Mission (SRTM), which provides digital elevation data with a resolution of about 90m. By combining these two sources, we created a road map with elevation and cruising speed information for every point in this network. The first step was then to derive a graph with weights corresponding to the energy consumption of road sections. For this purpose, we used a simplistic physical model of an EV. The consideration of recuperation induces negative weights for a small percentage of edges.

Figure 3 shows the web interface to our prototype. It is possible to choose source, destination and car type on the left side. The blue path displays the energy efficient shortest path according to the available data and vehicle model. While some of the proposed deviations from a straight (shortest) route are indeed due to energy savings, others originate from an overly simplistic vehicle model and some missing speed tags in the OSM data; future development will address these problems. Within this prototype, we evaluated Algorithm 2.1 using the four different strategies dijkstra, expand expand-distance, and FIFO. The evaluation was carried out on a section of the OSM map that covers the Allgäu region southwest of Munich, and contains 776,419 vertices and 1,713,900 edges. We set $U_s = 0$ and take $k = 10$ randomly selected source vertices s as input. The individual runtimes x_i were used to calculate the average time $\bar{x} = \frac{1}{k}\sum_{i=1}^{k} x_i$ (first value in columns 2-5 in Table 1) and the variance $\frac{1}{k}\sum_{i=1}^{k}(x_i - \bar{x})^2$ (second value). These calculations were done for different values of C_{max} with steps of 10 kWh. Additionally, the average size of the corresponding prefix-bounded shortest path tree (number of vertices) is presented. Our experiments were conducted on a Intel Core2 Duo CPU with 2.20 GHz and 2 GB RAM.



Startadresse: Arcisstrasse 19
PLZ münchen

Zieladresse: boltzmannstrasse 3
PLZ garching

Fahrzeugtyp: Sportwagen
Höhendaten: Optisch (30x30m)

Energieverbrauch: 11,69 kWh

1 Vorschläge zu Eingabe 'Arcisstrasse 19 münchen':
Arcisstraße, 80333 München (K

2 Vorschläge zu Eingabe 'boltzmannstrasse 3 garching':
Boltzmannstraße, 85748 Dieter:

Figure 3: Screenshot of our route planner prototype, showing the energy optimal route for an EV from TUM's Munich campus to TUM's Garching campus.

In Table 1 it can be seen that the strategies FIFO and expand are far away from practical usability, since we have aborted the individual executions for $C_{max} > 20$ after one minute without any result. Strategy expand-distance is less than two times slower than dijkstra in our experiments (average of 10 computations), and moreover, the former strategy has the advantage that the worst case time complexity is polynomial.

7 Conclusion

Optimal routing for electrical vehicles with rechargeable batteries will become increasingly important in the future. In this paper, we studied this problem within a graph-theoretic context. We modeled energy-optimal routing as a shortest path problem with constraints, and proposed a family of search algorithms that respect these constraints with a worst case time complexity of $O(n^3)$; in fact, our problem constitutes a tractable variant of the more general constrained shortest path problem. Furthermore, in order to also take into account energy costs or gains that result from velocity changes, we presented a method that preprocesses the routing graph, with a modest price in terms of size increase. Further research will study the impact of the negative/positive edge ratio in

C_{max} [kWh]	dijkstra	expand	FIFO	expand-distance	average tree size
10	0.28 / 0.00	1.20 / 1.07	0.80 / 0.19	0.29 / 0.00	19695
20	0.33 / 0.01	16.23 / 169.21	23.80 / 963.10	0.36 / 0.01	67211
30	0.44 / 0.02	> 60 / -	> 60 / -	0.54 / 0.03	136884
40	0.64 / 0.07	> 60 / -	> 60 / -	0.85 / 0.15	231475
50	0.86 / 0.13	> 60 / -	> 60 / -	1.22 / 0.35	330229
60	1.00 / 0.18	> 60 / -	> 60 / -	1.54 / 0.55	414227
70	1.13 / 0.19	> 60 / -	> 60 / -	1.84 / 0.68	486764
80	1.22 / 0.19	> 60 / -	> 60 / -	2.11 / 0.78	551194
90	1.32 / 0.16	> 60 / -	> 60 / -	2.37 / 0.68	612294
100	1.41 / 0.11	> 60 / -	> 60 / -	2.59 / 0.59	664905
infinity	1.57 / 0.03	> 60 / -	> 60 / -	3.14 / 0.17	776419

Table 1: Runtime in seconds (and variance) for computing energy-optimal paths with algorithm ConstrainedGenericShortestPath, using four different strategies.

our routing graphs and the development of special tailored heuristics using the law of conservation of energy. In addition, we plan to extend our approach by modeling the energy consumption with stochastic instead of constant values for assessing the risk of running out of energy before arriving at the destination. Finally, it is interesting to extend the framework towards energy-efficient management of a fleet of EVs, for instance in car-sharing scenarios.

References

[BCR93] B.V. Cherkassky, A.V. G. ; Radzik, T.: Shortest paths algorithms: Theory and experimental evaluation. In: *Mathematical Programming* 73 (1993), Nr. 2

[Bel58] Bellman, R.: On a routing problem. In: *Quart. of Appl. Math.* 16 (1958), Nr. 1, S. 87–90

[BFSS07] Bast, Holger ; Funke, Stefan ; Sanders, Peter ; Schultes, Dominik: Fast routing in road networks with transit nodes. In: *Science* 316 (2007), Nr. 5824, S. 566

[Dij59] Dijkstra, E.: A note on two problems in connexion with graphs. In: *Numerische Mathematik* 1 (1959), S. 269–271

[GP86] Gallo, G. ; Pallottino, S.: *Mathematical Programming Studies.* Bd. 26: *Shortest path methods: A unifying approach.* Springer Berlin Heidelberg, 1986

[GSSD08] Geisberger, Robert ; Sanders, Peter ; Schultes, Dominik ; Delling, Daniel: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In: *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)* (2008)

[Joh73] Johnson, Donald B.: A Note on Dijkstra's Shortest Path Algorithm. In: *Journal of the ACM* 20 (1973), Nr. 3, S. 385–388

[Jok66] Joksch, H. C.: The Shortest Route Problem with Constraints. In: *Journal of Mathematical Analysis and Applications* 14 (1966), S. 191–197

[LRF56] L. R. Ford, Jr: Network flow theory / RAND. 1956. – Forschungsbericht

[MD79] M., Garey ; D., Johnson: *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979

[SS05] In: Sanders, Peter ; Schultes, Dominik: *Lecture Notes in Computer Science*. Bd. 3669: *Highway Hierarchies Hasten Exact Shortest Path Queries*. Springer, 2005

[ZN00] Zhan, F. B. ; Noon, C. E.: A Comparison Between Label-Setting and Label-Correcting Algorithms for Computing One-to-One Shortest Paths. In: *Journal of Geographic Information and Decision Analysis* 4 (2000), Nr. 2, S. 1–11

A Appendix: Proofs of Section 4

Theorem 4.1. *Algorithm 2.1 computes for every strategy S a prefix-bounded shortest path tree with respect to function* absorp*.*

Proof. Every time a vertex v with $d(v)$ is inserted into Q (i.e. the operation in line 16), a path P from s to v in G (defined over the predecessor function p) with $d(v) = \text{absorp}(P)$ is found. Due to the inequality in line 13 it additionally holds that every calculated path is prefix-bounded by C_{max}. Moreover the current path has always a smaller value $d(v)$ than any previously seen path with target v. Since there is only a finite number of paths from s to v in G, v is only finitely often inserted into Q. Therefore Q will finally become empty (due to deletion in 9) and the algorithm terminates.

For the claim of the theorem it is sufficient to show that every vertex $v \in V$ is sometime inserted with $d(v) = d(s, v)$ into Q (where $d(s, v)$ denotes the distance of the shortest path from s to v with respect to absorb). Let

$$P = (s = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k = v)$$

be a shortest prefix-bounded path with respect to absorp in G. Therefore we prove by induction over $i = 1, \ldots, k$ that v_i is inserted with $d(v_i) \leq \text{absorp}(P^i)$ into Q. The base case $i = 1$ is obvious. For the induction step $i \rightarrow i + 1$ we notice that for every strategy S, v_i will be eventually removed from Q and Q will finally become empty. Due to the check in line 13, v_{i+1} is inserted with

$$d(v_{i+1}) = \max\left(d(v_i) + c(v_i, v_{i+1})\,, 0\right)$$

into Q, or this vertex was previously added with a smaller value. Hence using the induction hypotheses we have

$$\begin{aligned} d(v_{i+1}) &\leq \max\left(d(v_i) + c(v_i, v_{i+1}), 0\right) \\ &\leq \max\left(\text{absorp}(P^i) + c(v_i, v_{i+1}), 0\right) = \text{absorp}(P^{i+1}). \end{aligned}$$

Therefore theorem 4.1 is proven. □

Theorem 4.2. *Algorithm 2.1 using strategy expand has time complexity $O(n^3)$.*

Proof. For every vertex u in Q, there is a corresponding path $P(u)$ from s to u, which is defined by predecessor function p. We want to show $\text{expand}(v) \leq |P(u)|$, where $\text{expand}(\cdot)$ denotes the number of expansions of a vertex and $|\cdot|$ the graph-theoretical length of a path.

We prove by induction over $i \geq 1$ that this assertion is true at any time i the algorithms is in line 7 and for every vertex in Q. The base case $i = 1$ is obvious, since there is only s in Q: $\text{expand}(s) = 0 = |P(s)|$. For the induction step $i \rightarrow i + 1$ we assume u chosen to be expanded at time i. It is sufficient to show that every successor v of u, which is updated in 15 at time i, fulfills $\text{expand}_{i+1}(v) \leq |P^{i+1}(v)|$. (The subscript index denotes the time which is considered). Now, let us consider the last time $j < i + 1$, where v was chosen in line 7 (if v was never chosen, we have $\text{expand}_i(v) = 0$ and there is nothing to prove). Considering our strategy and using induction hypotheses it is

$$\text{expand}_j(v) \leq \text{expand}_j(x) \leq |P^j(x)|$$

for every vertex x in Q at time j. Since exactly the vertices in Q are expanded, every path, which is found at time $k > j$, contains at least one vertex x. Therefore the graph-theoretical lengths are increasing and especially it is $|P^j(x)| \leq |P^i(u)|$. Since $P^{i+1}(v)$ consists of the path $P^i(u)$ and the edge (u, v), we have

$$\text{expand}_{i+1}(v) = \text{expand}_j(v) + 1 \leq |P^i(u)| + 1 = |P^{i+1}(v)|.$$

as required.

Every graph-theoretical length of a path is bounded by $n - 1$. Hence no vertex is expanded more than n times. Since every expansion takes at most $O(n)$ and the minimal element can surely be found with $O(n)$, this leads to the desired time complexity $O(n^3)$. □

Lemma 4.3. *Algorithm 2.1 using strategy dijkstra has time complexity $O(n^2)$ if c is non-negative.*

Proof. Let u be the vertex which is chosen at time i in line 7. Since c is non-negative and $d_i(u)$ is minimal (subscript index denotes the time which is considered), we get $d_j(v) \geq d_i(u)$ for all vertices v chosen in 7 at every time $j \geq i$. Hence value $d(u)$ cannot be improved anymore. Therefore every vertex is inserted only once into Q. Since every expansion (line 10 to 16) takes $O(n)$ and the minimal element can surely be found with $O(n)$, we get an overall time complexity of $O(n^2)$. □

B Proofs of Section 5

Theorem 5.1. *Algorithm 5.1 has time complexity $O(n^3)$ and generates a directed graph with weights representing the energy consumption with respect to the function absorp.*

Proof. The time complexity of the algorithm originates from its three nested loops. For every vertex in the original graph V all its predecessors are traversed. The third loop connects every of these predecessors with the successors of v. The number of predecessors and successors is bounded by $n - 1$. Therefore the algorithm needs $O(n^3)$ steps to terminate.

The second part of the theorem is divided in two parts. First, we proof that a path $p = (v_1, v_2, \dots, v_n)$ with $v_i \in V$ exists between two vertices in graph G iff a path $p' = (v'_1, v'_2, \dots, v'_n)$ with $v'_i \in V'$ exists in graph G' with $M(v'_i) = v_i \ \forall i \in \{1, 2, \dots, n\}$. This follows directly from the central idea of the algorithm: each vertex is replaced by different vertices, each compromising incoming edges with the same velocity in the original graph. Obviously no new paths are created and every path in G has a projection $M(p') = p$ from a path p' in G'.

Finally we show that the correct weights are used. The energy consumption is calculated in line 15. Because all incoming edges have the same velocity in the modified graph G' we can use a constant value $c(v_u, w) \leftarrow F((v, w), g(u, v))$ instead of a function. □

Theorem 5.2. *Let $\Delta(v)$ for $v \in V$ be the number of outgoing edges in graph G and $\omega(v) = |\{c(u, v) | \exists (u, v) \in E\}|$ be the number of different weights of the incoming edges of $v \in V$. Graph $G' = (V', E')$ generated by algorithm 5.1 has then order $|V'| = n' = \sum_{v \in V} \min(\omega(v), 1)$ and size $|E'| = m' = \sum_{v \in V} (\min(\omega(v), 1)\Delta(v))$.*

Proof. We first determine the number of vertices a vertex v in V in graph G is replaced with in order to gain graph G'. In line 2 every vertex from V is inserted in V' which will be finally removed again in line 17 if it has at least one predecessor. In the second loop a new vertex is added to V' for every predecessor u of v in G if no other with the same velocity value has been processed before. Therefore, $\omega(v)$ vertices are inserted. Note that $\omega(v)$ is the cardinality of the weights of the predecessors of v. Consequently, v is replaced by $\omega(v)$ vertices if it has predecessors or left in the graph otherwise which yields a total number of $|V'| = n' = \sum_{v \in V} \min(\omega(v), 1)$ vertices in the generated graph G'.

The size of the graph G' is the sum of the outgoing edges of its vertices. We determine this number for all nodes replacing one vertex v in G. The edges inserted in line 2 are removed with its adjacent vertices in line 17 if the source vertex has at least one predecessor. Line 13 creates an edge from every of the $\omega(v)$ vertices replacing v to every of the $\Delta(v)$ successors of v. We see that every vertex v in V leads to exactly $\min(\omega(v), 1) * \Delta(v)$ outgoing edges in G'. $\square$