Optimal Route Planning for Electric Vehicles in Large Networks

Jochen Eisner and **Stefan Funke** and **Sabine Storandt**
Universität Stuttgart, Institut für Formale Methoden der Informatik, 70569 Stuttgart, Germany
jochen.eisner, stefan.funke, sabine.storandt@fmi.uni-stuttgart.de

Abstract

We consider the problem of routing electric vehicles (EV) in the most energy-efficient way within a road network taking into account both their limited energy supply as well as their ability to recuperate energy. Employing a classical result by Johnson and an observation about Dijkstra under non-constant edge costs we obtain $\mathcal{O}(n \log n + m)$ query time after a $\mathcal{O}(nm)$ preprocessing phase for *any* road network graph whose edge costs represent energy consumption or recuperation. If the energy recuperation is height induced in a very natural way, the preprocessing phase can even be omitted. We then adapt a technique for speeding-up (unconstrained) shortest path queries to our scenario to achieve a speed-up of another factor of around 20. Our results drastically improve upon the recent results in (Artmeier et al. 2010) and allow for route planning of EVs in an instant even on large networks.

Introduction

With the increasing popularity of electric vehicles (EV) as an alternative to fossil fuel driven cars, new routing problems arise that were not as relevant before. Due to large petrol tanks and the very dense network of gas stations, availability of energy is almost no issue for ordinary cars. Current EVs, though, are still hindered by an energy reservoir that limits their cruising range. On the other hand, EVs have the ability to recuperate energy during deceleration phases or when going downhill as long as this does not exceed the capacity of their energy reservoir. Formalized as a graph problem we are faced with the following question:

Given a graph $G(V, E)$ with a possibly negative cost function $cost : E \to \mathbb{R}$ but without any negative cycle, a source node $s \in V$, some $L \in \mathbb{R}^+$ (the initial charge level), some $M \in \mathbb{R}^+, M \geq L$ (maximum charge level), and a target node $t \in V$, determine a path $p = v_0 v_1 \dots v_k$ with $s = v_0, t = v_k$ as well as charge levels b_{v_i} for each of these vertices such that

1. $b_s = L$ (we have not used any energy at the start)
2. $b_{v_i} = \min(b_{v_{i-1}} - cost((v_{i-1}, v_i)), M)$, $i = 1, \dots, k$ (taking edge costs into account but no overcharging of the battery)
3. $b_{v_i} \geq 0, i = 1, \dots, k$ (we must never run out of energy)
4. b_t is maximal over all such possible paths from s to t

Note that without conditions 2 and 3 and minimizing $-b(t)$ we have a regular shortest path problem, which can be solved in time $\mathcal{O}(nm)$ using Bellman-Ford's algorithm. Employing Dijkstra's algorithm might lead to exponential running time due to the edge costs possibly being negative (Johnson 1973).

Our Contribution

The main contributions of our paper are as follows:

- battery capacity constraints – both overcharging as well as running out of energy – can be modelled as cost functions on the edges obeying the FIFO property; hence a regular Bellman-Ford algorithm can be applied to solve the problem in $\mathcal{O}(nm)$; this simplifies and matches the result in (Artmeier et al. 2010)
- by employing a generalization of Johnson's potential shifting technique to the (partly) negative edge cost functions we make Dijkstra applicable; this results in a drastically improved query time of $\mathcal{O}(n \log n + m)$
- the specific construction of our edge cost functions allows for the adaptation of a speed-up strategy for shortest path queries – *contraction hierarchies* (Geisberger et al. 2008) – resulting in ultra fast queries without suffering from the problem of complexity growth of the function descriptions; this improvement is directly reflected in our experimental results on large, real-world problem instances

Modelling Battery Constraints

There are two obstacles that prevent us from using standard Dijkstra (as well as of speed-up techniques like edge reach, transit nodes, or contraction hierarchies): first, the edge costs may be negative (which corresponds to energy recuperation along some road segment) which forbids the use of Dijkstra and only allows for the application of the $\mathcal{O}(nm)$ Bellman-Ford algorithm; secondly, we have the battery constraints which seem to require special algorithmic treatment. In fact, (Artmeier et al. 2010) have shown that a *modified* version of Bellman-Ford solves the problem. But like Bellman-Ford they can only guarantee a running time of $\mathcal{O}(mn)$ and require a dedicated proof of correctness for their algorithm.

In the following we first observe that one can model the battery constraints by cost functions on the edges obeying the FIFO property which immediately implies direct applicability of Bellman-Ford without any further thought; furthermore, a generalization of Johnson's potential shifting technique yields a problem instance with non-negative edge cost functions which ordinary Dijkstra can solve. This will be a prerequisite for the application of an advanced speed-up technique towards later in the paper.

Definition 1 *A function $f : \mathbb{R} \to \mathbb{R}$ satisfies the FIFO (first-in-first-out) property, if $\forall x \leq y$ we have $x + f(x) \leq y + f(y)$.*

In the context of road networks with time-dependent edge costs this means that one cannot arrive earlier at the end of a road segment when starting later, in our context, this means that one cannot end up with more energy at the end of a road segment when starting with a lower battery level.

We now turn to the modelling of our constraints as edge cost functions.

Never Running out of Energy

Let us look at the constraint that the EV never runs out of energy, that is, when being at a node v and considering travelling along edge $e = (v, w)$ and $const_e$ exceeds the current charge level of the EV (i.e. $const_e > b_v$), we are not allowed to take this road segment. Note that this concerns only edges with $const_e > 0$. Consider the function $c_e : \mathbb{R} \to \mathbb{R}$, $c_e(b_v) = \infty$ if $const_e > b_v$ and $c_e(b_v) = const_e$ otherwise. c_e sets the edge cost to infinity if the energy level does not suffice to get across edge e and to $const_e$ otherwise. Clearly c_e satisfies the FIFO property.

No Overcharging

The constraint that the battery charge never exceeds the maximum charge level M can similarly be modeled in a cost function. Note that this only concerns edges $e = (u, v)$ with $const_e < 0$. When arriving with a battery level b_u and such that $b_u - const_e > M$, we want the energy level at w to be M only. Hence define $c_e(b_u) = b_u - M$ for $b_u - const_e > M$ and $c_e(b_u) = const_e$ otherwise. Again, the FIFO property holds.

Having encoded the battery constraints into cost functions on the edges obeying the FIFO property we can use plain Bellman-Ford; the only difference is that when a node v is pulled from the queue, the edge cost functions on its outgoing edges are evaluated at b_v. If all functions on the edge costs were positive, even straightforward Dijkstra could be applied, see (Dreyfus 1968). Unfortunately, this is not the case here (yet). The edges that lead to a gain in energy bear cost functions which are partly negative, which we alleviate later on.

Unified Cost Function Representation

Dealing with two different edge cost functions (depending on the edge types) is somewhat inconvenient. Therefore we unify our cost function representation by defining a new function $c_e : [0, M] \to \mathbb{R}$ to describe the cost of an *arbitrary* edge $e = (v, w)$ (be it a downhill or uphill edge) dependent on the actual battery charge b_v at node v. This can be realized by

$$c_e(b_v) = \begin{cases} \infty & b_v < l_e \\ const_e & b_v \in [l_e, u_e] \\ b_v - u_e + const_e & b_v > u_e \end{cases}$$

with $l_e,\ u_e,\ |const_e| \in [0, M]$ being parameters depending on e. For an uphill edge e with $const_e \geq 0$ we have $l_e = const_e$, $u_e = M$, for an downhill edge e with $const_e < 0$ we have $l_e = 0, u_e = M + const_e$. It is easy to see that this function class subsumes the two previous definitions of c_e and obeys the FIFO property.

Shifting Edge Cost Functions

At this point we are dealing with a graph $G(V, E)$ and for each edge $e \in E$ we have a cost function $c_e : \mathbb{R} \to \mathbb{R}$ obeying the FIFO property. While some of the c_e might have negative function values, there is no negative cycle, i.e. for any battery charge level b_v on a node v we cannot take a roundtrip and end up with a higher charge status than b_v, otherwise we would have constructed a perpetual motion machine of the first kind.

For graphs with constant, possibly negative edge costs, the shifting idea of (Johnson 1977) is to determine a potential function $\phi : V \to \mathbb{R}$ and replace edge costs $const_e$ of an edge $e = (v, w)$ by $const'_e := const_e - \phi(w) + \phi(v)$. If the graph has no negative cycles, a ϕ exists such that $const'_e \geq 0$ $\forall e \in E$. It is not hard to see that the shifting does not affect the structure of shortest paths, since the potential-induced cost of a path from s to v does not depend on the path itself. We can derive such a potential function by computing shortest path distances (using Bellman-Ford) from some arbitrary node x and setting $\phi(v) := d_x(v)$, where $d_x(v)$ denotes the shortest path distance from x to v.

Generalizing this shifting technique to our edge cost *functions* requires the node potentials to become functions, so $\phi : V \to (\mathbb{R} \to \mathbb{R})$. While for constant edge costs standard Bellman-Ford can be used to determine potentials, we now have to employ Bellman-Ford to perform a profile search (Delling and Wagner 2009), which as a result not only yields for any $v \in V$ a *distance value* to x but a *function* describing the distance dependent on the initial charge level at x. This is relatively complicated as well as time- and space-consuming. If the edge cost functions satisfy an additional condition (which is naturally fulfilled in our scenario), we can get away without this inconvenience:
Consider the graph which on each edge e, instead of the function $c_e()$ has constant cost $\underline{const}_e := \min_d c_e(d)$. Assume that the graph with these edge costs does not contain any negative cycle; we can derive a potential function $\phi : V \to \mathbb{R}$ in the same manner as before for constant edge costs. Let us convince ourselves of the use of this potential function for our graph with edge cost functions. We define as new edge cost functions $\widetilde{c_e}(d(v))$ for every $e = (v, w)$:

$$\widetilde{c_e}(d(v)) := c_e(d(v) + \phi(v)) + \phi(v) - \phi(w)$$

Since we have $d(w) \leq d(v) + \underline{const}_e$, obviously $\phi(v) - \phi(w) \geq -\underline{const}_e \geq -c_e(\chi)$ for any χ and hence $\widetilde{c_e}(d(v)) \geq 0$.

It remains to show that this transformation does not change the structure of shortest paths. For that we inductively proof that the cost under the new edge cost function $\widetilde{c}_e()$ equals the cost of the old edge cost function $c_e()$ minus $\phi(v_n)$:

Lemma 2 *Let $p = v_0 v_1 \dots v_k$ be a path with $s = v_0$ and initial distance value $d(s) = d$ under the original edge cost functions $c_e()$ and the same path with initial distance value $\widetilde{d}(s) = d - \phi(s)$ under $\widetilde{c}_e()$.*
Then this yields $\widetilde{d}(v_k) = d(v_k) - \phi(v_k) \quad \forall k \geq 0$.

Proof. This is certainly true for $k = 0$. For $k > 0$ observe, that $\widetilde{d}(v_{k+1}) = \widetilde{d}(v_k) + \widetilde{c}_{e_k}(\widetilde{d}(v_k))$. According to our new edge cost definition we get $\widetilde{c}_{e_k}(\widetilde{d}(v_k)) = c_{e_k}(\widetilde{d}(v_k) + \phi(v_k)) + \phi(v_k) - \phi(v_{k+1})$. By induction hypothesis $\widetilde{d}(v_k) = d(v_k) - \phi(v_k)$, so all in all we get $\widetilde{d}(v_{k+1}) = d(v_k) + c_{e_k}(d(v_k)) - \phi(v_{k+1}) = d(v_{k+1}) - \phi(v_{k+1})$. ∎

Hence our shifting procedure does not affect the structure of shortest paths – paths that are shortest under the old edge cost functions are also shortest under the new ones since the potential-induced cost of a path does not depend on the path itself.

Finally observe that in our case, the Graph (V, E) with edge costs $\underline{const}_e$ has no negative cycle. This is easy to see since we have $\underline{const}_e = const_e$, that is, we have the original edge costs. By assumption this graph did not have any negative cycles. As $\widetilde{c}_e() \geq 0$ for all e and all the $\widetilde{c}_e()$ are trivially shown to fulfil the FIFO property, we can employ Dijkstra's algorithm once we have determined the potential function $\phi()$. Also we can not only calculate the original distances but also the actual battery charge status on $v_n = t$ from the modified edge costs as $b_t = \widetilde{b}_t + \phi(s) - \phi(t)$. So subsequent $s-t$ queries can be answered in time $\mathcal{O}(n \log n + m)$ (Fredman and Tarjan 1987).

A Realistic Cost Model

In the application scenario of energy-efficient routing, the typical case where a road segment $e = (v, w)$ bears negative cost (meaning a *surplus* of energy when travelling along this segment) is that the elevation of the target node w is lower than the one of the source node v, so we are going 'downhill'. It is natural to model the cost of an edge $e = (v, w)$ as $const_e := fixed_e + \alpha(\eta(w) - \eta(v))$, where $fixed_e$ denotes a height-independent part (e.g. depending on the horizontal distance only), $\eta(w),\ \eta(v)$ the elevations of w and v respectively, and α, a weight factor which determines how important the height-dependent part is. Clearly, with this simple model our problem gets considerably easier as the height-dependent part of the cost of a path only depends on the heights of the first and the last node in a path, all other height-dependent parts cancel out, unless overcharging takes place, which can be dealt with quite easily, though.

It gets more interesting (and realistic) when we introduce the natural condition that the energy for going uphill ($\eta(w) - \eta(v) > 0$) is not fully recuperated when going downhill ($\eta(w) - \eta(v) \leq 0$): So we have $const_e = fixed_e + (\eta(w) - \eta(v))$ for $\eta(w) - \eta(v) > 0$ and $const_e = fixed_e + \alpha(\eta(w) - \eta(v))$ for $\eta(w) - \eta(v) \leq 0$ with some $0 < \alpha < 1$. We drop α in the uphill case as this could easily be compensated for by scaling up the $fixed$ part.

Plugging in η for the function ϕ which we had to painfully construct previously, we see that the desired $\widetilde{c}_e()$ can be determined with no effort if the underlying model is known. So in most practical cases we do not need to compute ϕ using Bellman-Ford but simply use the underlying elevation function on the nodes of the network.

Speeding-up Queries with Contraction Hierarchies

The key idea of contraction hierarchies (CH) (Geisberger et al. 2008) is the iterative removal/contraction of nodes in a certain order (in increasing 'importance') while preserving the shortest path distances between the remaining nodes. This is achieved by adding a so called shortcut (u, w) between any pair of neighbors u, w of v, if the shortest path from u to w is uvw. The shortcut is created with cost equal to the sum of the costs of edges (u, v) and (v, w). Having removed all nodes but one, all constructed shortcuts are added to the original graph and the nodes are labelled $1 \dots n$ according to the contraction order. The modified graph has the interesting property that for any pair s, t of nodes, there exists a shortest path from s to t which can be divided into two parts, one part starting at s and only following edges to nodes with larger label followed by a part which only follows edges to nodes with smaller label. Bidirectional Dijkstra can make use of this special property and leads to an extremely efficient computation of s-t-queries, see (Geisberger et al. 2008).

The same approach also works for time-dependent edge costs, see (Batz et al. 2009), but as the authors point out, contracting nodes in time-dependent networks typically leads to cost functions on the shortcut edges with increasing complexity and space consumption due to chaining. In the following we will exhibit some nice characteristics of our edge cost function class that allow for the *space-efficient* employment of contraction hierarchies in our scenario.

Descriptive Complexity

For general edge cost functions the descriptive complexity might increase with every contraction. For example consider two polynomials of degree k, chaining them leads to a polynomial with degree k^2 and so for $k \geq 2$ the description complexity increases dramatically. Similarly, chaining two step functions with different interval boundaries in the worst case leads to a combined function with about twice the number of steps.
In the following we show, that the cost function of a path in our edge cost model has bounded descriptive complexity and hence repeated node contraction does not lead to unbounded growth in function complexity.

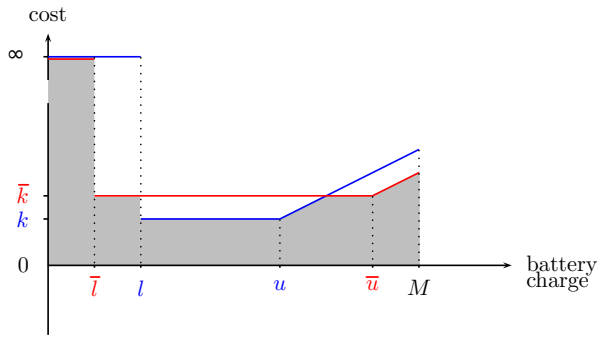

Figure 1: Two examples of the used edge cost function.

Theorem 3 *The descriptive complexity of a cost function c_p of a path p on up- and downhill edges is bounded.*

Proof. Consider an arbitrary path $p = sv_1v_2 \cdots v_{k-1}t$ in the graph $G(V, E)$ with edge costs as defined before. Let l be the minimal battery charge a vehicle has to have at node s to reach t along p, i.e. none of the edge costs $c_{e_i}()$ become ∞ for that starting charge at v_i. Because of the FIFO property we get $\forall b_s < l : c_p(b_s) = \infty$. Amongst all charges at s larger or equal to l let u be the first charge where overcharging along p kicks in (it is possible that $u = l$). Again, due to the FIFO property all initial charges lower than u do not cause overcharging, hence $\forall b_s \in [l, u] : c_p(b_s) = \sum_{i=1}^{k} const_{e_i}$. For battery charges greater than u there is at least one node v_{i^*} where the battery is fully loaded. Therefore $b = b_t$ is the same $\forall b_s > u$ and so the path costs on this interval can be expressed by a linear function, more precisely as the difference of the initial battery power and b. So in summary we need l, u, $const = \sum_{i=1}^{k} const_{e_i}$ and b_s to describe the cost function of an arbitrary path in G and hence the descriptive complexity of chained edge cost functions is bounded. ∎

Calculation of Chained Edge Costs

When creating a shortcut between neighbors v and v'' of a node v' to be contracted, we need to combine the edge cost functions of edges e and e' that form a shortest path $vev'e'v''$ from v to v'' via v'. The newly created shortcut edge edge $\hat{e}$ gets as cost function $c_{\hat{e}}(b_v) := c_e(b_v) + c_{e'}(b_{v'}) = c_e(b_v) + c_{e'}(b_v - c_e(b_v))$. Three cases need to be distinguished as in Table 1.

From this description it is obvious that the cost function for the short cut edge can be found in constant time.

No Space Overhead In practice it might not be enough to have a path cost function with bounded descriptive complexity. Consider for example a cost function linear in b_v with slope larger than one on every edge. When chaining this type of cost function along some path, the coefficient of the (still linear) chained cost function grows exponentially in the length of the path. In the worst case this might lead to coefficients which need $\Omega(n)$ space or to put up with rounding errors. Fortunately in our case parameters only get added or subtracted, as the slope of our linear cost function in the interval $]u, M]$ is always 1, see Table 1. Therefore, irrespectively how many edge cost functions are chained, we never require more space than the original edge costs, even better the parameters also never exceed the range $[-M, M]$, as if they do we can simply set them to the interval boundary without changing the function. So we are able to chain any number of edges and use the same datatypes to store the resulting parameters as for the original edge costs.

Creation of Shortcuts

When contracting a node v and considering edges $e_1 = (w, v)$ and $e_2 = (v, z)$, a shortcut between neighbors w and z with the chained cost function of e_1 and e_2 has to be added, if there exists some battery level at w where the cost along e_1 and e_2 is lower than the cost in the graph with v removed. Naively this requires computing shortest paths from w to z at all possible battery charge levels at w and comparing them with the cost function of the path e_1e_2; in fact it is possible to do exactly this using a so-called *profile search* (Delling and Wagner 2009), which is relatively time- and space-consuming, though. Instead, we discretize the range of possible battery charges and compute shortest paths for these charge levels. For each of these paths (containing at least one node different from w, v, z) we consider its cost function (ranging over all possible charge levels); if for any charge level one of these cost functions does not exceed the cost function of e_1e_2 the shortcut is *not* necessary. This can easily be checked by inspecting the cost functions at all breakpoints. This does not compromise correctness, it might lead to the addition of some unnecessary shortcuts, though. Note that this construction of the contraction hierarchy sometimes leads to multiple edges between a pair of nodes since our allowed function class does not allow for construction of shortcuts bearing the *minimum* of a set of cost functions. Our experimental evaluation shows, though, that this only happens rarely and still allows for a drastic speed-up.

Queries

To answer a query from some node s with battery charge level b_s to some node t we run a breadth first search (BFS) from s identifying all nodes that are reachable from s via edges from smaller to larger labelled nodes, and a BFS from t identifying all nodes that can reach t via edges from larger to smaller labelled nodes (this follows the exposition in (Delling and Wagner 2009)). On the identified edges we run Bellman-Ford (as we still have negative edge weights). But then we can employ the shifting idea from Section to obtain a graph with non-negative edge costs on which Dijkstra is applicable. So we can take full advantage of fast Dijkstra as well as the contraction hierarchy.

	$l_{\hat{e}}$	$u_{\hat{e}}$	$const_{\hat{e}}$
$u_e < l_{e'} + const_e$	M	M	∞
$u_e \in [l_{e'} + const_e, u_{e'} + const_e]$	$\max(l_{e'} + const_e, l_e)$	$\max(l_{\hat{e}}, u_e)$	$const_e + const_{e'}$
$u_e > u_{e'} + const_e$	$\max(l_{e'} + const_e, l_e)$	$\max(l_{\hat{e}}, u_{e'} + const_e)$	$const_e + const_{e'}$

Table 1: Case distinction for chaining edge cost functions.

Experimental results

We present the results for two test graphs, which are based on OpenStreetMap[1] data augemented with SRTM height information[2] and can be characterized as mountainous. The smaller one – the German Taunus – contains 11220 nodes and 24119 edges. Furthermore we had a map of Southern Germany with 5588146 nodes and 11711088 edges. Our implementation is written in C++, timings were taken on a single core of a AMD Phenom 9850 Processor with 2.5 GHz and 16 GB RAM. Preprocessing to get the contraction hierarchy for the larger graph took about 6 CPU hours and added ≈ 1.6 million edges, resulting in an augmented graph with roughly 2.5 times the number of original edges.
The initial battery charge was chosen such that the underlying search space was not restricted by insufficient initial energy. For smaller battery charges, queries are even faster, of course.

Taunus	polls avg (std)	
Bellman-Ford	14080 (10257)	below
Dijkstra	3883 (8084)	measurement
CH	69 (127)	precision
Southern Germany	**polls avg (std)**	**avg runtime in s**
Bellman-Ford	$2.03 \cdot 10^8$ ($6.32 \cdot 10^7$)	66.91
Dijkstra	$5.11 \cdot 10^6$ ($3.28 \cdot 10^6$)	3.43
CH	$5.53 \cdot 10^4$ ($3.86 \cdot 10^4$)	0.18

Table 2: Runtime comparison: Values are averaged over 1000 random queries. "polls" is the number of (priority)queue polls, standard deviation in brackets, runtime given in seconds. Timings on the Taunus graph were omitted as they were too small to be measured.

In Table 2 we see that the number of poll operations as well as the runtime decreases drastically when using Dijkstra instead of Bellman-Ford and another order of magnitude less can be achieved by applying contraction hierarchies. Note, that only the latter allows us to answer queries in a fraction of a second and hence provide instant answers in practice.
Compared to (Artmeier et al. 2010) which only consider a graph of about 1/8th the size, our Dijkstra strategy is superior, whereas our CH strategy beats their results by an order of magnitude. The former can be explained by our potential shifting technique which guarantees $\mathcal{O}(n \log n + m)$ running time in contrast to worst-case exponential running time of Dijkstra in (Artmeier et al. 2010). The latter is simply due to the raw power of contraction hierarchies not being hindered by uncontrolled complexity growth of the edge cost functions.

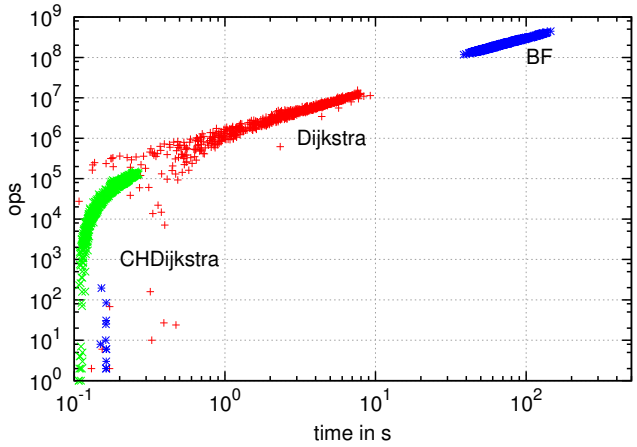

Figure 2: Runtime in s and cost in (priority)queue polls measurements for 1000 random queries on the Southern Germany graph for our approaches. Both scales are logarithmic. The CH Dijkstra does always have to mark the up and down edges, resulting in a minimal static runtime of about 0.1s for running these two BFSs.

In Figure 2 we see that the running times in general are determined by the number of (priority)queue polls, even though there is a constant cost for CH Dijkstra due to the marking of the edges using BFS.
As expected for real data the percentage of edges with negative costs is very small even for graphs with large height differences. Still, taking into account energy consumption and recuperation due to the height differences results in quite different paths, see Figure 3. Note that the paths in Figure 3 do not differ much in length but considerably in terms of energy consumption. Choosing the right path might make the difference between being able to reach the target or to run out of energy in the middle of the trip. The EV chooses a path that avoids driving uphill as far as possible, leading to a higher final battery charge see Figure 4. Energy can also get lost when driving downhill, see Figures 6 and 5. If the initial battery charge is more than 96.5875% we always end up with a final charge level of 98.5%. That means, starting with a full battery leads to a positive total energy consumption due to the overloading constraint, while we could gain energy on the very same path using a smaller initial battery charge.
In general we observe that routing vehicles under battery constraints and height-induced edge cost functions leads to considerably different optimal path structures, still queries can be answered very efficiently with our approach.

[1] www.openstreetmap.org/
[2] www2.jpl.nasa.gov/srtm/

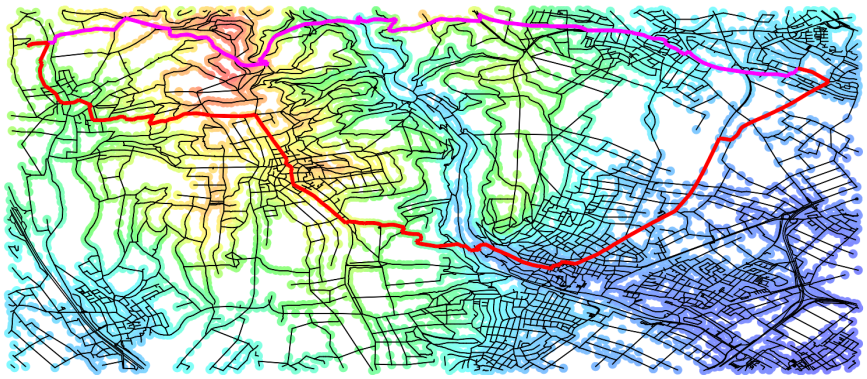

Figure 3: This is a map section of the german Taunus, spanning an area of about 16.1×6.8km. The background colors are indicating the elevation ranging from 99m as deep blue to 412m as red. Two map spanning paths are shown, sharing the same source in the northeastern corner directing westward and also ending in the same target. The upper violet path is the one generated by Dijkstra and is a shortest path in respect to eucledean length. The lower red one is the energy optimal path generated by Johnson.

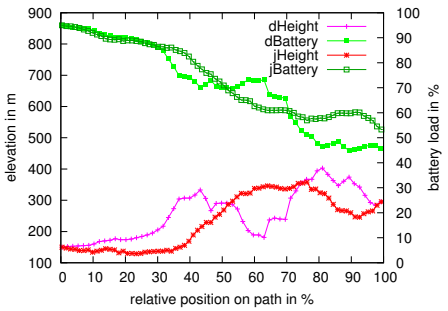

Figure 4: *dHeight* and *jHeight* are the elevation profiles for the Dijkstra and Johnson paths shown in fig. 3. *dBattery* and *jBattery* are the corresponding battery charges. Although the eucledean length of the Dijkstra path is the shorter one, the energy consumption is higher, resulting in a lower final battery charge.

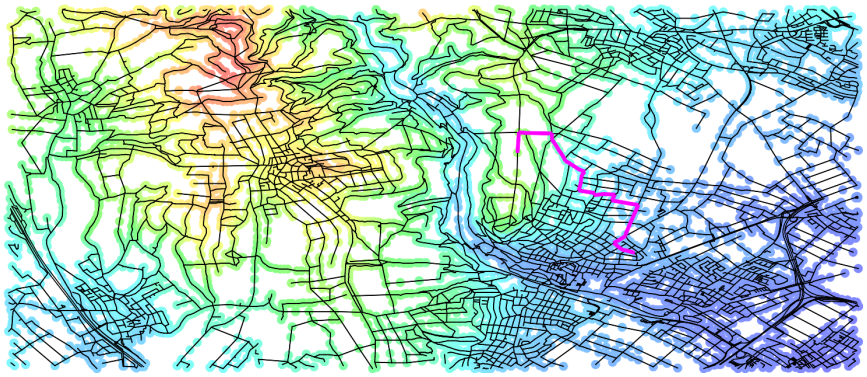

Figure 5: This is the same map section as shown in fig. 3. The violet path on the right starts at 279m and ends at 149m if followed in eastern direction. The downhill nature of the path allows us to study the influence of initial battery charge on battery overloading and net energy cost.

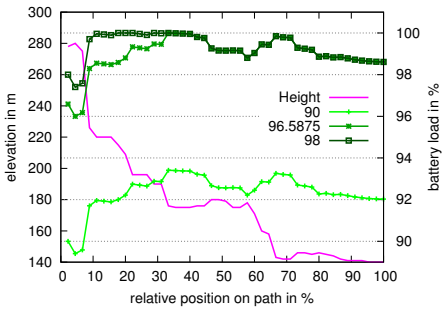

Figure 6: *Height* is the elevation profile of the path, shown in fig. 5, starting at 279m and ending at 149m. 90, 96.5875 and 98 are the battery charge on each path node where the respective number is the initial battery charge (ie. 90%). 96.5875% marks the highest possible battery charge where no overloading occurs.

References

Artmeier, A.; Haselmayr, J.; Leucker, M.; and Sachenbacher, M. 2010. The shortest path problem revisited: Optimal routing for electric vehicles. In *33rd Annual German Conference on Artificial Intelligence (KI-2010)*.

Batz, G. V.; Delling, D.; Sanders, P.; and Vetter, C. 2009. Time-dependent contraction hierarchies. In Finocchi, I., and Hershberger, J., eds., *ALENEX*, 97–105. SIAM.

Delling, D., and Wagner, D. 2009. Time-dependent route planning. In Ahuja, R.; Mhring, R.; and Zaroliagis, C., eds., *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, 207–230. Springer Berlin / Heidelberg.

Dreyfus, S. E. 1968. An appraisal of some shortest path algorithms. In *ORSA/TIMS Joint National Mtg.*, volume 16, 166.

Fredman, M. L., and Tarjan, R. E. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* 34:596–615.

Geisberger, R.; Sanders, P.; Schultes, D.; and Delling, D. 2008. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In *Proceedings of the 7th international conference on Experimental algorithms*, WEA'08, 319–333. Berlin, Heidelberg: Springer-Verlag.

Johnson, D. B. 1973. A note on dijkstra's shortest path algorithm. *J. ACM* 20:385–388.

Johnson, D. B. 1977. Efficient algorithms for shortest paths in sparse networks. *J. ACM* 24:1–13.