Placement of Loading Stations for Electric Vehicles: No Detours Necessary!

Stefan Funke and **André Nusser**
Universität Stuttgart
Institut für Formale Methoden der Informatik
70569 Stuttgart, Germany
{funke,nusser}@fmi.uni-stuttgart.de

Sabine Storandt
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
79110 Freiburg, Germany
storandt@cs.uni-freiburg.de

Abstract

Compared to conventional cars, electric vehicles still suffer from a considerably shorter cruising range. Combined with the sparsity of battery loading stations, the complete transition to E-mobility still seems a long way to go. In this paper, we consider the problem of placing as few loading stations as possible such that on any shortest path there are enough to guarantee sufficient energy supply. This means, that EV owners no longer have to plan their trips ahead incorporating loading station locations, and are no longer forced to accept long detours to reach their destinations. We show how to model this problem and introduce heuristics which provide close-to-optimal solutions even in large road networks.

Introduction

Battery-powered, electric vehicles (EVs) are an important means towards a reduction of carbon dioxide emissions when recharged using renewable energies, e.g. from solar or wind power. Despite their environmental advantages EVs still wait for their breakthrough with the main reason being their limited cruising range (often less than 200km) together with the sparsity of battery loading stations (BLS). Planning a trip from A to B with an EV nowadays is a non-trivial undertaking; the locations of BLS have to be taken into account, and many destinations are not even reachable.

Hence in this early phase of E-mobility an important goal is to establish a network of BLS such that using an EV becomes a worry-free enterprise. As modern BLS require only small space (see Figure 1 for an illustration), they can be placed almost everywhere. But as this generates costs, a natural objective is to minimize the number of installed BLS. In (Storandt and Funke 2013), the authors propose a heuristic to determine BLS locations such that one can get from anywhere to anywhere in the road network without running out of energy (when choosing a suitable route). Unfortunately, this approach only guarantees connectivity but not reasonability of the routes. In fact, even rather close destinations where routes with only one recharging stop are possible, might require long detours with several recharging stops due to the placement of BLS. A related approach by (Lam, Leung, and Chu 2013) suffers from similar drawbacks. In the long run, E-Mobility will only prevail if a road trip with an EV can be undertaken without unreasonable detours being introduced. In this paper we ask for a placement of the BLS such that on *any* shortest path there are enough BLS not to get stranded when starting with a fully loaded battery – just like it is typically the case with gas stations for conventional cars. We call such a set of BLS locations an EV Shortest Path Cover (ESC) and define the respective optimization problem as follows:

Given a (di)graph $G(V, E)$, edge costs $c : E \to \mathbb{R}^+$ and a function η which for a path π decides whether this path can be traveled along without recharging the EV, the problem of determining a minimum subset $L \subseteq V$ of BLS such that every shortest path wrt c can be traveled without running out of energy is called the EV Shortest Path Cover Problem.

Note that for sake of simplicity we will assume unique shortest paths – this can easily be enforced using standard techniques like symbolic perturbation. The function η captures all the energy characteristics of the network and the considered vehicle. Typically, in mountainous areas or on roads with rough surfaces, the minimal paths where energy runs out are considerably shorter than in flat terrain or down-

Figure 1: Inner-City Battery Loading Station

hill. For our experiments we determined the energy consumption of a road segment $(v, w) \in E$ with elevations $h(v)$, $h(w)$ as $|vw| + \alpha \cdot \max(h(w) - h(v), 0)$ for some weighting parameter α (dependent on the EV). That is, the energy consumption is determined by the Euclidean distance and the height differences, similar to the energy model in (Artmeier et al. 2010) but disregarding energy recuperation (negative edge costs). The function η compares for a path π the accumulated energy consumption along its edges with the EV's battery capacity to determine if recharging is necessary. Note that one could employ any kind of monotonous function η here, the algorithms we will introduce in the following are not dependent on this particular choice.

Contribution

In this paper we describe how to model the ESC problem as an instance of the Hitting Set problem with the sets being shortest paths which require at least one battery recharge. This allows us to use algorithms developed for solving Hitting Set problems, e.g. the standard greedy approach which guarantees a solution within a factor of $O(\log n)$ of the optimum (n being the number of nodes in the network). Unfortunately, it turns out that the difficulty of computing an ESC solution is already the instance construction. With $\Theta(n^2)$ shortest paths in the network, extracting and storing them naively requires too much time and space to be practical. We therefore develop new shortest path extraction and representation techniques, which allow to tackle much larger input networks. Moreover, we develop several refinements and heuristics which provide feasible ESC solutions more efficiently. While no a priori approximation guarantee can be provided for them, we can prove a posteriori – using instance-based lower bounds – that for real-world instances the actual approximation ratio is only a small constant.

Preliminaries

To determine suitable BLS positions, we first have to construct the set of shortest paths on which the EV would run out of energy (according to η). Computing the shortest path between two nodes or from one to all other nodes is classically performed using Dijkstra's algorithm. In large street networks Dijkstra is too slow to process a large number of such queries (as it will be necessary for our application), though. Therefore we will instrument speed-up techniques developed for accelerating shortest path queries to achieve better run times for our approaches. In particular, we will employ *Contraction Hierarchies (CH)* (Geisberger et al. 2008) for this purpose. The basic idea behind CH is to augment the graph $G(V, E)$ with a set E' of so called shortcuts, which span (large) sections of shortest paths. Using these shortcuts instead of original edges where possible allows for a dramatic reduction of operations in a Dijkstra run. In the CH-preprocessing phase, each node gets assigned a label $l : V \to \mathbb{N}$. According to these labels, original or shortcut edges (v, w) are referred to as upwards if $l(v) < l(w)$ and downwards otherwise; paths are called up/downwards if they consist exclusively of edges of that type. Shortcuts are inserted such that for each node pair $s, t \in V$ a shortest path exists in $G' = G(V, E \cup E')$ which can be decomposed into an upward path starting at s followed by a downward path ending in t (the highest node of the path wrt l is called the *peak node* in the following). This property allows to restrict a Dijkstra run to $G^{\uparrow}_{out}(s)$ and $G^{\downarrow}_{in}(t)$ which refer to the subgraphs of G' containing only all upwards paths starting in s or all downwards paths ending in t respectively.

For the one-to-all shortest path problem, the PHAST algorithm (Delling et al. 2011) takes advantage of the CH-preprocessing. Here in a first phase all nodes in $G^{\uparrow}_{out}(s)$ for a source $s \in V$ are settled via a Dijkstra run. In the second phase all edges (v, w) are relaxed in the order induced by $l(w)$, thereby computing distances to all nodes in V.

Modelling ESC as a Hitting Set Problem

The classical Hitting Set (HS) problem is defined as follows: *Given a set system $(U, \mathcal{S})$ with U being a universe of elements and $\mathcal{S}$ a collection of subsets of U, the goal is to find a minimum cardinality subset $L \subseteq U$ such that each set $S \in \mathcal{S}$ is hit by at least one element in L, i.e. $\forall S \in \mathcal{S} : L \cap S \neq \emptyset$.* In our case, U consists of all nodes of the road network (the possible BLS locations), $\mathcal{S}$ is composed of the vertex sets of all shortest s-t-paths (excluding s and t themselves) for which a fully charged battery at s does not suffice to reach t. Our function η characterizes these paths whose energy consumption exceed the battery capacity $B \in \mathbb{R}^+$ – we call them *B-violating*. Clearly, we only need to consider set-minimal paths as supersets are hit automatically. At this point, common Hitting Set solving techniques can be applied, e.g., the standard greedy algorithm for HS repeatedly picks the node hitting most so far unhit sets in $\mathcal{S}$ and adds it to the solution. It terminates as soon as all sets are hit. Its running time depends crucially on fast access to the so far unhit sets in $\mathcal{S}$ in each round.

In the remainder of the paper, we will investigate the efficient construction of the set system using different path extraction and representation schemes and study their influence on the greedy algorithm.

Construction of the Set System

The first step towards an ESC solution is to extract the set system $\mathcal{S}$, i.e. computing all minimal shortest paths which are B-violating.

Naive. The simplest approach that comes to mind is computing the shortest path tree (via Dijkstra) for every $s \in V$ and identifying all nodes in the tree with accumulated energy cost values above B. Once all nodes in the priority queue of Dijkstra have settled predecessors that already belong to B-violating paths, we can abort the exploration from that source. The respective paths in the search tree can then be backtracked and stored as vertex sets. For small exploration radii (small bounds B) this might be a practical approach, but for larger exploration radii the time complexity of $\mathcal{O}(n^2 \log n + nm)$ (with n being the number of nodes and m the number of edges in the network), and the space consumption of $\mathcal{O}(n^2\sqrt{n})$ (assuming an average path length of $\sqrt{n}$) limit usability for real-world instances.

In fact, this is also the main difficulty for other Hitting-Set-type problems on street networks. For example, speed-up techniques for shortest path queries like *Transit Nodes (TN)* (Bast, Funke, and Matijevic 2009) or *Hub Labeling (HL)* (Abraham et al. 2012) are based on hitting a certain set of shortest paths as well. Methods for complete instance construction are impractical there. Therefore several custom-tailored heuristics were developed that allow for efficient computation without explicitly constructing $\mathcal{S}$, as e.g. described in (Arz, Luxen, and Sanders 2013) for TN. But their setting differs significantly from ours, as only a single metric is involved (not c and η as in our case). Hence the distance bound employed there leads to a set of equal length paths, while in our scenario due to different energy consumption when driving uphill or downhill the lengths of minimal B-violating paths differ vastly. So unfortunately these TN and HL (and other related) heuristics do not carry over to our setting. Therefore we need to explore new ways of extracting and storing shortest path sets.

PHAST-based Extraction. For large bounds B finding all B-violating paths from a source node resembles the one-to-all shortest path problem. PHAST was explicitly designed to solve this task efficiently. The paths we can backtrack in the respective search tree are in CH-representation, i.e. they consist partly of shortcuts. This is a huge advantage compared to conventional paths in terms of storage, because with shortcuts spanning large portions of the shortest path the number of nodes in the CH-path is significantly smaller (about two orders of magnitude for the street network of Germany). There are some downsides, though: Nodes are processed in the second phase of PHAST in l-order and not increasingly by distance; hence incorporating B as stopping criterion seems difficult. Moreover if B is not that large or leads to paths with vastly differing lengths, the n^2 lower bound for PHAST from every source might already result in a large overhead. Hence we now propose a different strategy which has the potential of being significantly faster.

Peak Node Mapping (PNM). A large number of B-violating paths can originate from a source $s \in V$, and exploring all these paths with Dijkstra or PHAST is very time-consuming. The core idea of PNM is to enumerate B-violating paths completely different by considering the CH-representation of paths. As explained above, shortest CH-paths are unimodal wrt the labeling l and the node with maximal label is called the *peak*. Intuitively, nodes with a high label appear in more shortest paths as peaks. In fact, in real-world graphs, the 5% highest level nodes constitute the peaks of all reasonably long shortest paths. This gives rise to a path enumeration algorithm, which explores paths not from the source but from the peak, resulting in dramatically reduced search spaces for the majority of nodes.

Our PNM algorithm runs as follows: We consider one by one every node $p \in V$ as potential peak. As all shortest paths with peak p can only contain further nodes with a smaller label, we only need to search upwards paths ending in p and downwards paths starting in p for prefix and suffix candidates. The respective subgraphs of the CH-graph G' containing these paths are called $G_1 := G'^{\uparrow}_{in}(p)$ and $G_2 := G'^{\downarrow}_{out}(p)$. A conventional Dijkstra run in each of G_1 and G_2 (which are typically very sparse) reveals the distances between the contained nodes and p. Now we are interested in combinations of shortest upward paths in G_1 and a shortest downward paths in G_2 leading to minimal B-violating paths. Testing them all naively is too expensive. Therefore we construct for each p an interval tree on the nodes in G_2. The interval $[a, b]$ which we associate with such a node t denotes the range of possible energy consumption values of a path prefix $\pi(s, p)$ in G_1 such that $\pi(s, p) \cup \pi(p, t)$ is a B-violating path (and no subpath is). These intervals can easily be computed by a single parse over the Dijkstra search tree in G_2. So for every possible source $s \in G_1$, we query the interval tree for the set of targets T in time $\mathcal{O}(\log(|G_2|)+|T|)$ storing the resulting paths as quadruples $(s, p, t, c(s, p) + c(p, t)), t \in T$.

After all nodes are processed, we have a set of B-violating paths from which unfortunately not all are shortest paths. Obviously, the concatenation of two shortest paths ($\pi(s, p)$ and $\pi(p, t)$) does not need to be a shortest path itself. So it remains to filter this set appropriately. This can be achieved by using distance oracles with quasi constant look-up time as e.g. provided by HL or by another pass over all nodes in the role of the peak, always pruning a quadruple if for s, t a shorter path was found for $p' \neq p$. Note, that pruning can already be employed during the construction phase if the intermediate path set sizes become too large.

The final set of paths is then stored as list of triples (s, p, t) – an even more compact representation than CH-representation. Accessing all nodes in the respective shortest s-t-path in G as required for the greedy Hitting Set algorithm is no longer trivial, though. Therefore, we will provide suitable adaptions of the greedy algorithm to work on the CH-representations and on PNM triples in the next section.

Transformability. Note, that the extraction scheme does not tie us to a certain path representation. In fact, all mentioned representations (vertex sets, CH-paths, triples) can be converted into each other with little effort. Especially the transformation from the naive representation to CH-paths will turn out to be favorable, as CH-paths yield a fair trade-off between space consumption and applicability of the greedy algorithm as explained in more detail in the next section.

Greedy Hitting Set Computation

As explained above, the greedy approach is a natural strategy to solve the Hitting Set problem approximately. Theoretically it yields solutions within a factor of $\mathcal{O}(\log n)$ of the optimum but typically performs much better in practice.

Adaptation to Set System Representation

For all but the simplest set system representation the application of the greedy Hitting Set algorithm is not straightforward and requires some deliberate operations on the set system/path representations as we will see in the following.

Complete Vertex Sets. If the paths are simply given as the set of contained vertices, a single scan over all these sets can determine the 'best' node hitting most paths. Another scan can remove the paths that have been hit by the selected node.

Unfortunately, the space consumption of this approach is enormous, also making a single scan quite expensive.

CH-paths. When representing the minimal B-violating paths as CH-paths, we could convert them into original paths by unpacking the shortcuts. For this purpose, we store with each shortcut the original edges it spans during the CH-construction. But there is a much better strategy than just uncompressing every single CH-path to get to the original node sets: maintaining a usage counter for each edge, we first scan over all edges of all CH-paths in the set system to be hit, incrementing the respective counters. Then we traverse all shortcut edges of the graph in decreasing order of their construction in the CH-preprocessing, incrementing the counters of the spanned edges. The node counters, maintained to identify the maximum node, can then be derived by a final scan over all original (non-shortcut) edges. Keeping reverse information about which edges are spanned by which shortcut also allows the identification of all sets that have been hit by a node. If we update the edge counters when removing CH-paths from the set, picking a node requires only one scan over the edges to push the counts down on the non-shortcut edges, one scan over the usage counters and one scan over the set of CH-paths.

Peak Node Triples. When paths are described as triples of source, target, and peak node, the extraction of the CH-path representation for each path takes some more effort, though. For every peak node p we have to perform a Dijkstra run in $G^{\downarrow}_{out}(p)$ and a second one in reversed $G^{\uparrow}_{in}(p)$. For all source-target pairs associated with this peak, the CH-path can then be generated as the respective concatenation of subpaths. Then we proceed as described above for the CH-path representation. Note, that this CH-path representation is computed on demand for each peak in every round in order to not end up with the total space consumption of storing all paths in CH-representation.

Multi-Stage Construction

For country-sized graphs, even the improved set system extraction methods and representations do not reduce the space and time consumption enough to be practical. In particular for larger battery capacities, the exploration time and space from a single node increase dramatically.

For an instance of our ESC problem determined by the battery capacity B, we make the following important observation: For every capacity $B' \leq B$ a Hitting Set L' for the instance corresponding to B' is also feasible for the original instance (having enough BLS for a smaller battery capacity obviously also suffices for a larger battery capacity). While the construction of L' for some $B' \ll B$ might be considerably faster due to smaller exploration radii, L' is typically also much larger than necessary for the instance defined by B. But there is another advantage of first quickly computing a Hitting Set for small value B': it allows us to construct a new small problem instance for which any feasible Hitting Set L'' is also feasible for our original problem (defined by B) with assumable better quality than L'. This idea can be used to design a multi-stage heuristic to retrieve a good solution for the original problem as we will see in the following.

Nested Hitting Sets. Let us assume we already computed a solution L' for $B' < B$. Now consider the set of shortest, minimal $(B - B')$-violating paths starting with a node in L'. A Hitting Set for those paths is also a feasible ESC solution L for B, as every B-violating s-t-path in the original instance must be hit by a node $v \in L'$ at most B' away from s, and the subpath $v, \cdots, t$ (or a subpath thereof) has to be in the new constructed path set, so this subpath is hit by L.

For very small values of B, we can even compute Hitting Sets without any exploration and evaluation of the η function purely based on the connectivity structure of the graph using a so-called *k-Hop Path Cover* (similar to (Funke, Nusser, and Storandt 2014)) which is a generalization of a *Vertex Cover*. We construct a set of vertices $C \subseteq V$ such that *any* directed (not necessarily shortest) k-hop path in G contains at least one vertex from C (for $k = 1$ this is simply a Vertex Cover). Obviously, C is an ESC solution for B^* where B^* is the maximal energy cost of a k-hop path, which can easily be upper bounded by k times the maximal energy cost of an edge. For values $k \leq 48$ this takes only few minutes even on large graphs using a variant of depth first search, making this step negligible for the overall running time.

In our implementation we combined nested Hitting Sets and k-Hop Path Covers to a multi-stage procedure, constructing a sequence of Hitting Sets $L_r, L_{r-1}, \cdots, L_1 = L$ for a sequence of values $B_r < B_{r-1} < \cdots < B_1 = B$, finally returning L as the Hitting Set for the given instance.

The first B_r results from a k-Hop Cover with small value of k, for all subsequent solutions we apply the nested Hitting Set approach (and choose B_i manually). There might be some loss in terms of quality compared to the greedy algorithm on the full set system due to the nested construction. Our experimental evaluation will show, though, that the loss in terms of quality is not that pronounced, but the running times are drastically improved, and the graph sizes which we can handle with this approach are much larger.

Refinements and Lower Bounds

Multiple Hitters Heuristic. Even with the non-naive representations, there is considerable work involved when picking the next 'best' node in the greedy algorithm. So it might be worthwhile to add several nodes to the Hitting Set in each round. Normally, we would pick only the node which hits most so far unhit sets, and refrain from picking other nodes in the same round as picking the first node influences the hit counters of the others. On the other hand, if we pick nodes that do not interfere with each other, we should do fine. One way to achieve this, is to generate the list of nodes sorted in ascending order of their hit counters, always picking the first one and then going down the list selecting the next nodes which have shortest path distances of at least D to all nodes already picked. Here D is appropriately chosen, e.g. an upper bound on the longest shortest path that is not B-violating. Thereby we make sure no path in our set increased the hit counter of two or more picked nodes.

Simple Instance-based Lower Bounds. To evaluate the quality of our heuristics, we would like to compare the outcome to the optimal solution. But as the optimal value is

typically unknown, we instead compare to a good, but easily computable lower bound. In (Eisner and Funke 2012) a rather involved lower bound was constructed, which takes comparable effort as solving the Hitting Set problem itself. We propose a much simpler alternative which suffices for our purposes: As a by-product of the generation of the set system itself we can obtain a *set of node-disjoint B-violating paths*. Clearly, any feasible solution must contain an extra node per path in this set. Hence the size of a set of node-disjoint paths yields a valid lower bound.

Experimental Evaluation

Our proposed techniques for computing ESC solutions were evaluated in a multi-threaded implementation written in C++ and executed on 2nd generation intel core desktop hardware, an i7-3930 (6 cores 64GB of RAM) for complete set generations and an i7-2700 (4 cores, 32GB RAM) for the multi-stage construction with nested Hitting Sets. We use the following abbreviations to state results: K=10^3, M=10^6, s=seconds, m=minutes, h=hours, d=days, GB=10^9Bytes. Several road networks of Germany derived from Open-

region	abb.	$\|V\|$	$\|E\|$
Pforzheim	(PF)	0.2M	0.4M
Tübingen	(TU)	0.5M	1.0M
Baden-Württemberg South	(BW)	2.2M	4.6M
Southern Germany	(SG)	4.2M	8.6M
Germany	(GE)	17.7M	36.0M

Table 1: Test graph characteristics.

StreetMap data (OSM) were used for evaluation, see Table 1 for an overview. Our used edge cost function c expresses travel time along an edge, so the paths to hit are indeed quickest paths (the term *shortest* paths is conventionally used for subsuming all kinds of metrics). Energy consumption of an EV was modeled as explained in the introduction using distance data from OSM and elevations provided by the Shuttle Radar Topography Mission (SRT). B corresponds to a battery capacity which translates to a certain cruising range (terrain dependent); our α equals 4.

Dealing with Complete Set Systems

Construction and Representation. Let us first examine the time and space complexity of extracting the complete set of minimal B-violating paths. We constructed set systems using the naive strategy (*NAIVE* – representing each path as the complete sequence of its vertices), the PHAST-based exploration (*PHAST* – with paths in CH representation), and peak node mapping (*PNM* – representing each path as *source,peak,target* triple). The respective results can be found in Table 2. Unfortunately, only the two smallest instances were feasible to process using all strategies; already for the BW graph, the time and space consumption of NAIVE exploded (extrapolated more than 500GB and more than 23 CPU days). In comparison, PHAST is about a factor of 3 faster than NAIVE, and the space consumption of CH-paths is an improvement by at least an order of magnitude. PNM can construct the BW instance in 4.4 CPU hours,

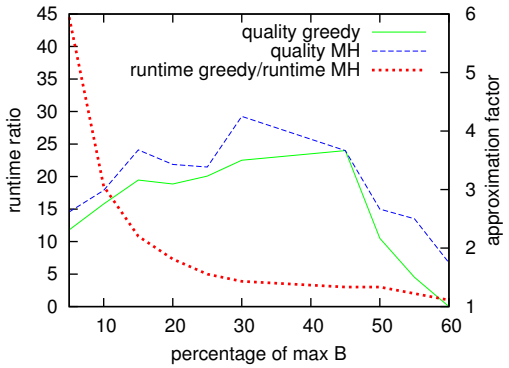

Figure 2: Performance of the greedy algorithm and the multiple hitters variant (MH) averaged over PF, TU and BW.

compared to the week needed by PHAST, and the space consumption using triples decreases by another factor of 2. But for SG and GE, also PHAST and PNM took too much time and space (e.g. extrapolated 557GB/112days for PHAST). So for larger networks, constructing complete set systems seems to be infeasible.

Hitting Set Computation. We evaluated both the standard greedy algorithm as well as the multiple hitters (MH) variation on the set systems for PF, TU, and BW with varying choices for B. Figure 2 shows their performance in terms of quality (standard greedy vs. MH) as well as running time (how much faster MH is compared to standard greedy). The ratios are averaged over all test graphs; the bound B is chosen between almost zero and 60 percent of the maximum energy consumption of some shortest path in the respective network. In all cases, greedy produces results much closer to the optimum as the theoretical $O(\log n)$ guarantee, the maximum deviation from the lower bound was indeed less than 4.5. Employing the MH strategy increases the HS size slightly, but yields significantly decreased running times especially for smaller bounds B (where the optimum is also larger). Still, compared to the construction time of the set systems, the Hitting Set computation times were negligible, so we do not state them explicitly here. This will change when we employ the multi-stage construction, though.

Multi-Stage Construction

As the construction of the complete set system has proven to be infeasible for larger road networks, we will make use of the idea of a multi-stage construction.

k-Hop Cover+PNM. Let us first examine how a compact set system can be constructed using the PNM approach after an initial k-Hop Path Cover. For the BW network we computed a $k = 32$-Hop Cover C ($146,494$ nodes) which corresponds to an ESC solution with $B' = 8832$ (and cruising range of about $9km$ in flat terrain). Then PNM is used to create a final compact set system by only considering $(B - B')$-violating paths that start at nodes in C. Not surprisingly, the number of paths to be hit reduces drastically from 2715M in Table 2 to 24M in Table 3. The running times are still quite high, though, as this approach does not save the exploration from each peak (therefore more stages will not help much

	# paths	computation time						space consumption		
		NAIVE		PHAST		PNM		NAIVE	PHAST	PNM
		CPU	real	CPU	real	CPU	real	vertex sets	CH-paths	triples
PF	38M	1.5h	0.3h	26.7m	5.0m	1.8m	25.1m	5.2GB	0.2GB	0.1GB
TU	168M	24.6h	4.1h	7.5h	1.4h	27.3m	5.4m	24.0GB	2.1GB	0.9GB
BW	2715M	[23.1d]	[3.9d]	7.5d	33.1h	4.4h	47.0m	[526.3GB]	34.6GB	14.3GB

Table 2: Comparison of path extraction/representation schemes. B corresponds to about 40km (for PF and TU) or 125km (for BW) cruising range on flat terrain. Timings include the CH-construction for PHAST/PNM. Values in brackets are extrapolated.

Graph	B_r	B_i's of nested HS	$\|L\|$	LB	APX	CPU	real
TU	–	1k,5k,40k	120	33	3.64	9m	3m
SG	–	2k,10k,125k	106	33	3.21	404m	106m
TU	1.8k	4.8k,40k	116	33	3.52	8m	2m
SG	4.2k	12.2k,125k	110	33	3.33	242m	63m
GE	15.8k	17.8k,33.8k,125k	868	190	4.57	908m	265m
GE	6.2k	8.2k,24.2k,125k	728	190	3.83	1156m	322m
GE	15.8k	17.8k,25.8k,49.8k,125k	1212	190	6.38	645m	209m

Table 4: Multi-stage Hitting Set computation (LB = lower bound, APX = approximation factor).
The last two experiments can be seen in detail in Table 5.

Graph	$\|C\|$	B'	CPU	real	# paths
BW	146,494	8832	2.2h	35.5m	24M
SG	180,455	10048	6.0h	2.8h	75M
GE	769,760	15808	64.8h	27.6h	1085M

Table 3: Instance creation ($B = 40K$) via PNM with initial k-hop solution C for $k = 32$.

here). Since further improvements in terms of running time using PNM in a multi-stage approach cannot be expected, let us now concentrate on the naive approach with the extracted paths being converted into their CH-representation.

Multi-Stage Hitting Sets. We employ the following strategy: first, construct a k-Hop Cover C with e.g. $k = 32$ which yields an initial Hitting Set L_r for some bound B_r. Then we construct a reduced set system consisting of all $(B_{r-1} - B_r)$-violating paths starting at nodes from L_r only and compute a Hitting Set L_{r-1} for that set system. We proceed iteratively until reaching B and the final Hitting Set $L_i = L$. It is intuitive to demand that the gap between B_2 and the original $B_1 = B$ should be large to make sure that the last Hitting Set instance still faithfully characterizes the original Hitting Set instance. Table 4 shows the results for various choices of multi-stage parameters. In Table 5 we give a more detailed account about the intermediate calculations for the large GE graph. The experiments confirm that the larger the gap between B_2 and B_1 is, the better the quality of the final Hitting Set. This comes at the cost of an expensive last stage, though. In contrast to the other experiments, the first two calculations in Table 4 have been conducted without an initial k-Hop Cover. The results obtained on TU and SG suggest that an initial k-Hop Cover accelerates the calculation while maintaining a similar Hitting Set size. Furthermore, all the APX values remain low even though the lower bounds were obtained in a naive way. This proves the excellent quality of the Hitting Sets for the particular instances. Table 5 shows that introducing multiple stages keeps the intermediate set systems rather compact, so efficient computation is actually possible.

i	B_i	t_{SS}	#paths	t_{HS}	$\|L_i\|$	CPU
4	6.2k	–	–	14s	1388k	14s
3	8.2k	363m	34.6M	25m	223k	388m
2	24.2k	249m	36.8M	21m	16k	270m
1	125k	467m	13.5M	31m	728	498m
$\sum$		1079m	–	77m	–	1156m
i	B_i	t_{SS}	#paths	t_{HS}	$\|L_i\|$	CPU
5	15.8k	–	–	36s	770k	36s
4	17.8k	203m	19.7M	25m	208k	228m
3	25.8k	101m	17.1M	21m	38.6k	122m
2	49.8k	81m	9.2M	17m	8445	98m
1	125k	146m	5.6M	50m	1212	196m
$\sum$		531m	–	113m	–	645m

Table 5: Statistics for a 4-stage run starting with a $k = 16$-Hop Cover (above), and a 5-stage construction initialized with a $k = 32$-Hop Cover on GE. #paths is number of sets to hit in the respective stage. t_{SS}/t_{HS} denote CPU time for set system construction/Hitting Set computation.

Conclusions and Future Work

We showed how to model and solve a natural and important facility location problem in the E-mobility context, taking a radically different approach than previous ones avoiding detours to loading stations for EVs.

While a naive strategy only allows for the solution of small instances of few hundred thousand nodes, our compact representation schemes for the underlying set systems and heuristic modifications of the standard greedy approach make the computation of a solution even for country-sized networks like that of Germany possible. Instance-based lower bounds certify that the solution quality is pretty close to optimal and far from the pessimistic theoretically achievable approximation bound. In fact it is remarkable that after all, it was possible to compute a 4-approximate solution to a seemingly intractable Hitting Set problem within few hours on a standard quadcore desktop PC. Our computation determined around 800 locations where placing BLS would establish complete coverage for Germany.

Our framework does not require the metric that decides which shortest paths have to be hit to be identical with the metric that determines which paths are shortest. In fact this

was factored out using our η function, which – depending on the application scenario – can also be used to implement other hitting criteria (e.g. hop distances or risk values).

In future work we intend to examine how the 'exact hitting' requirement can be relaxed. Naturally, it is not necessary that there is always a BLS right on the respective shortest path, but a nearby one suffices. This could be modeled by enlarging the vertex sets of the respective shortest paths by surrounding vertices. Hitting Set sizes for this variant are expected to be considerably smaller than for hitting all shortest paths directly. Another direction of research is to take into account capacity constraints of the BLS (like (Lam, Leung, and Chu 2013)); in particular in urban areas it is certainly necessary to provide recharging stations for a very large number of vehicles.

References

Abraham, I.; Delling, D.; Goldberg, A. V.; and Werneck, R. F. 2012. Hierarchical hub labelings for shortest paths. In *European Symposium on Algorithms (ESA)*, 24–35. Springer.

Artmeier, A.; Haselmayr, J.; Leucker, M.; and Sachenbacher, M. 2010. The shortest path problem revisited: Optimal routing for electric vehicles. In *German Conference on Artificial Intelligence (KI)*, 309–316.

Arz, J.; Luxen, D.; and Sanders, P. 2013. Transit node routing reconsidered. In *International Symposium on Experimental algorithms (SEA)*, 55–66. Springer.

Bast, H.; Funke, S.; and Matijevic, D. 2009. *Ultrafast shortest-path queries via transit nodes*, volume 74 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. Providence, RI: AMS. 175–192.

Delling, D.; Goldberg, A. V.; Nowatzyk, A.; and Werneck, R. F. F. 2011. Phast: Hardware-accelerated shortest path trees. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 921–931.

Eisner, J., and Funke, S. 2012. Transit nodes - lower bounds and refined construction. In *Algorithm Engineering and Experiments (ALENEX)*.

Funke, S.; Nusser, A.; and Storandt, S. 2014. On k-path covers and their applications. In *International Conference on Very Large Databases (VLDB)*.

Geisberger, R.; Sanders, P.; Schultes, D.; and Delling, D. 2008. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In *International Workshop on Experimental Algorithms (WEA)*, 319–333. Springer.

Lam, A.; Leung, Y.-W.; and Chu, X. 2013. Electric vehicle charging station placement. In *International Conference on Smart Grid Communications (SmartGridComm)*, 510–515.

The OpenStreetMap Project
http://www.openstreetmap.org.

Shuttle Radar Topography Mission
http://www2.jpl.nasa.gov/srtm.

Storandt, S., and Funke, S. 2013. Enabling E-Mobility: Facility location for battery loading stations. In *Conference on Artificial Intelligence (AAAI)*.