Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Placement of Loading Stations for Electric Vehicles: Allowing Small Detours

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

We consider the problem of covering a street network with loading stations for electric vehicles (EVs) such that EVs can travel along shortest paths and only require small detours (e.g., at most 3 km) to recharge along the route. We show that this problem can be formulated as a Hitting Set problem. Unfortunately, it turns out that even the explicit problem instance construction requires too much time and space to be practical. Therefore, we develop several approximation algorithms and heuristics to solve the problem. Our experiments show that even though small, the allowed detours lead to a considerable reduction in the number of required loading stations. Moreover, we devise an algorithm for planning high-quality EV-routes in a network with loading stations placed by our approach. We empirically show the usability of the routes by evaluating the number of reloading stops and the actually induced detour.

Introduction

Route planning for electric vehicles (EVs) still requires special care compared to route planning for conventional cars. With the latter, one typically decides for the shortest or quickest route from A to B, and relies on a sufficient density of gas stations nearby in case the car runs out of fuel. The situation for battery-powered EVs is quite different, though. Firstly, EVs have a relatively small cruising range due to their limited battery capacity. Hence reloading on longer trips (e.g., more than 150 kilometers) is mandatory for most EVs. Secondly, the network of loading stations is still too sparse to allow for cruising around without a priori taking into account recharging – not only is this inconvenient but also limits the usability of conventional route planning engines and navigation systems for EVs.

In previous work, different objectives were investigated for covering a street network with loading stations: In (Storandt and Funke 2013; Lam, Leung, and Chu 2013) a placement of loading stations was computed such that there exists a path between any two points in the network on which the EV does not run out of energy. So every destination is reachable from every source (assuming a fully charged EV at the source), but reasonability of routes was not considered. The resulting routes might require long detours compared to the shortest path, and also might require numerous reloading stops. Quite differently, the approach described in (Funke, Nusser, and Storandt 2014) assures that every shortest path in the network can be travelled with an EV by placing sufficiently many loading stations in the network. While this allows EV drivers to travel the very same routes as conventional drivers, it also requires a rather large number of loading stations.

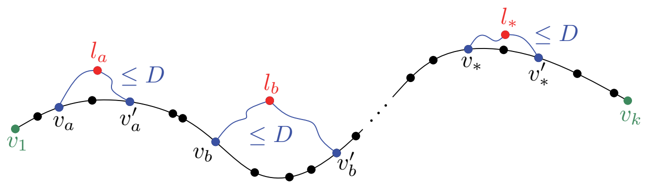

Figure 1: Shortest path (black) along with loading stations (red) that introduce a detour (blue) respecting the bound D.

We propose a new variant of placing loading stations in the network, respecting both, reasonability of routes and sparsity of the loading station set. Our goal is to place loading stations such that EV drivers only need to leave shortest path trips briefly on small detours for recharging. More formally, we are given a street network $G(V, E)$, edge costs $d : E \to \mathbb{R}^+$ (euclidean distances), and $B \in \mathbb{R}^+$ specifying the maximum distance an EV can drive without running out of energy (e.g., $B = 150$km). Additionally, we are given a detour bound D (e.g., $D = 3$km). A loading station $l \in L$ is accessible from a shortest path $\pi = v_1, v_2, \cdots, v_k$, if there are vertices v, v' on π with $d(v, l) + d(l, v') \leq D, v \neq v_k, v' \neq v_1$. We refer to v as the shortest path *exit vertex*, and to v' as the shortest path *(re-)entry vertex* for l. We call π feasible, if there exists a sequence of such loading stations $l_a, l_b, \cdots, l_*$ and respective exit/entry vertices $v_a/v'_a, v_b/v'_b, \cdots, v_*/v'_*$ such that

- $d(v_1, v_a) + d(v_a, l_a) \leq B$, i.e., the first loading station can be reached from the source without running out of energy
- $d(l_a, v'_a) + d(v'_a, v_b) + d(v_b, l_b) \leq B$, i.e., the second loading station l_b can be reached without running out of energy, and so forth for $l_c, l_d \cdots$
- $d(l_*, v'_*) + d(v'_*, v_k) \leq B$, i.e., the target vertex can be reached from the last loading station l_*.

See Figure 1 for an illustration.

We want to compute the smallest set $L \subseteq V$ of loading station positions, such that every shortest path in G becomes feasible wrt B and D – i.e., traversable without running out of energy, assuming the vehicle starts fully charged, and gets fully recharged whenever a loading station is visited. Note that by choosing $D = 0$ we have the same problem as considered in (Funke, Nusser, and Storandt 2014). Choosing a large detour bound D the problem converges to the pure reachability problem as described in (Storandt and Funke 2013; Lam, Leung, and Chu 2013). Hence D can be also interpreted as trade-off parameter between the two extreme models considered in previous work.

Note that previous work often distinguished between the metric determining the shortest/quickest route from A to B, and the edge metric that models energy consumption of the EV. Indeed, the energy consumption does not solely depend on the euclidean distance but also on other factors as the height profile of the underlying terrain (driving uphill consumes more energy than driving downhill). For sake of a cleaner presentation, we decided for a simplified model with both metrics being the same. But one can adapt our approaches to work also in the two metric setting, as we will describe briefly towards the end of the paper.

Related Work

In (Artmeier et al. 2010) the problem of finding energy-optimal routes for EVs was introduced, drawing attention to the fact that EVs might run out of energy on conventionally planned routes. Since then, numerous papers considered more complicated settings, like including reloading decisions and finding paths which are good in both aspects, energy consumption and travel time/distance. An algorithm for computing paths with the minimum number of necessary reloading events was introduced in (Storandt and Funke 2012) but without taking into account travel time or distance. In (Sweda and Klabjan 2012), dynamic programming was used to find an optimal recharging policy for a given route (assuming loading stations are placed directly on that route). In (Storandt 2012), multi-criteria queries, as finding a feasible path with a minimal number of reloading events and bounded distance/travel time were answered. In (Goodrich and Pszona 2014) two-phase routes were suggested, which in the first phase minimize travel time (so conventional route planners can be used), and only switch when necessary to the second phase where energy-consumption is minimized on the remaining path section. Here, also reloading is incorporated, again using two-phase paths between loading stations. Many other routing and reloading policies have been investigated, see e.g. (Baouche et al. 2014; Wang, Cassandras, and Pourazarm 2014; Hayakawa et al. 2014). But in all those papers, loading station positions were assumed to be given.

There is also related work that takes a traffic and city planning point of view on loading station placement. In (Xiong et al. 2015; He et al. 2013) the authors choose a game theoretical approach to take selfishly optimizing EV drivers into account. An approach where recharging is assumed to happen during parking was taken in (Chen et al. 2013; Frade et al. 2011), while (Liu, Wen, and Ledwich 2013; Wang et al. 2010) focus on the nodes in the road network with the highest charging demand. In all those approaches the number of loading stations is either assumed to be given or upper bounded by some given number. In contrast, we try to find an *exhaustive placement* of loading stations, such that EVs can drive around like conventional cars – (almost) deliberated from specialized a priori route planning.

Contribution

We introduce a new model for placing electric vehicle loading stations which is more practical than previously considered models, but also more complex. We show that a careful formulation of our covering problem can be interpreted as an instance of the Hitting Set problem. This, in theory, enables the usage of the standard machinery to solve Hitting Set problems, like the greedy algorithm (which guarantees a $\log(s)$ approximation with s being the number of sets). But we will show, that already the explicit Hitting Set instance construction is challenging as the time and space consumption is huge for large networks. We first investigate ways to reduce the space and time consumption without losing the approximation guarantee provided by the greedy algorithm. Subsequently we introduce heuristics that solve our problem even more efficiently, but at the cost of having no a priori approximation guarantee. We therefore propose new ways of computing lower bounds for the optimum, which then serve as quality indicators for our heuristic solutions. Finally we propose a simple and fast algorithm for computing actual routes for given loading station locations that minimize the number of recharging stops on the route.

Hitting Set Formulation

The classical Hitting Set problem is defined as follows:

Definition 1 (Hitting Set). *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 an element in L, i.e., $\forall S \in \mathcal{S} : L \cap S \neq \emptyset$.*

In (Funke, Nusser, and Storandt 2014) a Hitting Set formulation was used to make all shortest paths feasible without allowing detours. There, the universe is the set of nodes in the network ($U = V$), and the sets are minimal shortest paths where the EV runs out of energy (excluding source and target vertices, as placing loading stations there obviously does not help feasibility).

In our scenario with detours being allowed, a naive Hitting Set formulation would be to also extract all minimal shortest paths where the EV runs out of energy (so paths longer than B), but then 'thicken' these paths by computing all possible loading station positions around the path, that respect the detour bound D. Unfortunately, this would not induce global feasibility of all paths, as shown in Figure 2. We take care of this problem, by not extracting, thickening, and hitting all paths longer than B, but all paths longer than $B - 2D$. Lemma 2 shows that this indeed leads to a feasible route wrt B and D between any two nodes in the network. Note, that for correctness, we assume that the length of a

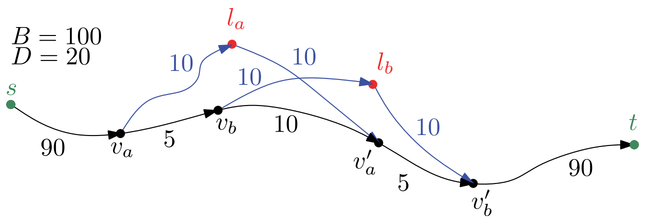

Figure 2: While all subpaths of the shortest path π from s to t on which the EV would run out of energy are hit with a loading station respecting the detour bound D, the overall path π is not feasible. Starting at s with a fully loaded battery and using l_a, we end up with a battery load of 90 at v'_a. This does not suffice to reach t, as the remaining costs are 95. The loading station l_b cannot be reached from s, as the battery load of 5 at v_b is not enough. We cannot use both loading stations as v_b comes before v'_a on π.

single edge is less than $B - 2D$. For reasonable parameter settings, this is naturally fulfilled in real-world instances. Otherwise, too long edges can be subsampled first.

Lemma 2. *If for every shortest path $\pi \in G$ with $d(\pi) > B - 2D$, there is a loading station $l \in V$ which induces a detour of at most D from π, the exit vertex for l is not the target vertex of π, and the re-entry vertex for l is not the source vertex of π, then every shortest path in G is feasible.*

Proof. Let π^* be some shortest path in G, with the source vertex s and the target vertex t. If π^* requires reloading, let π' be the minimal prefix of π^* with a distance exceeding $B - 2D$. Let l be the loading station that hits π', and v' the respective re-entry vertex. At v' the EV has a battery load of at least $B - D$, as the load at l is B (because of the recharging at l) and driving back to the shortest path can cost no more than D. Now consider the subpath π'' of π^* starting at v' that exceeds a distance of $B - 2D$. This path has to be hit as well. The battery load at v' is sufficient to use the next loading station, as the distance to the exit vertex has to be less than $B - 2D$ and the distance from the exit vertex to the loading station is less than D. Therefore, we have a new reachable re-entry vertex v'' later on π^* than v', and we can repeat the whole argumentation until the target is directly reachable from the current re-entry vertex. ■

Alternatively, we could demand that the distance from the exit vertex to the loading station is bounded by $D/2$ and the distance from the loading station to the re-entry vertex as well. Then it would obviously suffice to place loading stations on all thickened paths which exceed a length of $B - D$. Note, that this automatically is true if we consider undirected graphs.

Not also, that for real-world application one would possibly not use the real battery capacity B as bound but some $B' < B$ to accommodate for uncertainties in the energy consumption along the routes and also to allow for driving around safely when the battery is not fully charged in the beginning.

Approximation Algorithms

Based on the Hitting Set formulation, the placement of loading stations can be computed via the standard greedy algorithm. We first describe in detail how to construct the set system in order to apply greedy. Unfortunately, it turns out that the naive construction demands too much memory to be practical. We therefore propose a modified algorithm, which still exhibits the same approximation guarantee as the conventional greedy algorithm but consumes less space.

Explicit Thickening

Let us assume we already constructed the set Π of minimal shortest paths π_i in the network, which exceed a length of $B - 2D$. Efficient methods to compute Π were described in (Funke, Nusser, and Storandt 2014), so we use them as a black box here. Our Hitting Set instance construction demands the computation of 'thickened' paths. In the following, we describe our thickening algorithm in detail. The pseudo-code is given in Algorithm 1.

Algorithm 1 Explicit Thickening

```
1:  explicit_thickening($B$, $D$):
2:  $\Pi \leftarrow$ set of minimal shortest paths $\geq B - 2D$
3:
4:  for all $(\pi_i = s_i \dots t_i) \in \Pi$ do        ▷ construct set system
5:      $S_i \leftarrow \emptyset$
6:      for all $v \in \pi_i \setminus \{t_i\}$ do
7:          forward Dijkstra from $v$ until distance $D$
8:      end for
9:      for all $v \in \pi_i \setminus \{s_i\}$ do
10:         backward Dijkstra from $v$ until distance $D$
11:     end for
12:
13:     for all settled vertices $v$ do
14:         $c_f(v) \leftarrow$ minimal forward distance to v
15:         $c_b(v) \leftarrow$ minimal backward distance to v
16:         if $c_f(v) + c_b(v) \leq D$ then
17:             $S_i \leftarrow S_i \cup \{v\}$
18:         end if
19:     end for
20:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_i\}$
21: end for
22:
23: $\mathcal{H} \leftarrow$ hitting set for $(V, \mathcal{S})$
24: return $\mathcal{H}$
```

For every $\pi_i \in \Pi$ with source s_i and target t_i, we have to compute all loading station positions which make this path feasible to get the respective set $S_i \in \mathcal{S}$ (l. 4-21 in Algorithm 1). To that end, we run from every vertex on π_i a forward Dijkstra (excluding t_i) and a backward Dijkstra (excluding s_i), restricting the search spaces to paths of length D (l. 6-11). For every vertex we remember the minimal distance assigned to it in a forward run and in a backward run, respectively. If these costs sum up to at most D, this vertex belongs to S_i (l. 13-19). Note, that for real-world application, we might like to enforce a no-going-backwards-rule, i.e., the re-entry vertex v' for a loading station l should not be earlier on π_i than the exit vertex v. In that case, we only check for combined distances of source vertices fulfilling this rule.

Once all thickened path sets in the system are known, any algorithm for Hitting Set computation can be applied (l. 23). We use the standard greedy one. So in every round of the algorithm, the vertex that hits most so far unhit sets is determined, usually by a sweep over all elements in all sets, incrementing suitable counters. Then the best vertex (i.e., the one with the highest count) is added to the solution, and all sets that contain this vertex are removed from the set system, triggering another sweep over the sets to find those (during this sweep already the new hit counters for the reduced set system can be computed). As soon as there are no sets left in the system, the algorithm found a feasible solution.

Even for a moderate detour bound D, the sets in our system become much larger than the shortest paths themselves, rendering this approach only viable for extremely small networks as we will see in the experimental section. In (Funke, Nusser, and Storandt 2014), only shortest paths were extracted, but nevertheless for larger networks it was infeasible to store the set system explicitly (even using sophisticated compression tools). Hence in our scenario, the explicit instance construction method will reach its limits for considerably smaller input networks.

Implicit Thickening

To reduce the space consumption of greedy without losing its approximation guarantee, we want to compute faithful hit counters without storing the thickened sets explicitly.

We realize this with an *implicit* thickening approach which is described in the following. The respective pseudocode is provided in Algorithm 2; line numbers will be noted in brackets.

Again, we assume Π to be available (l. 2). Then, for every vertex v, we want to know how many paths in Π are hit by v via a detour $\leq D$ (l. 5-20). Therefore, we first associate with each vertex the IDs of paths it directly hits (l. 6), and a counter. Then for every $v \in V$, we run a forward and backward Dijkstra from v with a distance bound of D (l. 8,9). Subsequently, we search for all paths of which at least one vertex was settled in the forward run, and one in the backward run, with the combined distance being at most D. If the ID of such a path is not associated with v directly, we increment the respective counter for v (l. 11-17). The true number of paths hit by v is then the number of paths v directly hits plus the counter value (l. 19).

So using implicit thickening, the set of thickened paths does no have to be stored anymore. This reduces the space consumption significantly, but at the same time complicates the greedy algorithm (l. 22-36): If the best vertex is identified via comparing the number of hit paths, we have to remove all newly hit paths from the set and update the associated path IDs for every affected vertex as well as the counters. For directly hit paths, this can be done conventionally via a sweep over the affected paths (not the whole set system Π!). To identify the paths π_i indirectly hit (via a detour), we have to repeat the procedure described to determine the counter value (implying a forward and backward

Algorithm 2 Implicit Thickening

```
 1: implicit_thickening($B$, $D$):
 2: $\Pi \leftarrow$ set of minimal shortest paths $\geq B - 2D$
 3: $\mathcal{H} \leftarrow \emptyset$
 4:
 5: for all $v \in V$ do                        ▷ Initial count of hit numbers
 6:     direct_count $\leftarrow |\{\pi \in \Pi | v \in \pi\}|$
 7:
 8:     $c_f(\cdot) \leftarrow$ forward Dijkstra from $v$ until dist. $D$
 9:     $c_b(\cdot) \leftarrow$ backward Dijkstra from $v$ until dist. $D$
10:     indirect_count $\leftarrow 0$
11:     for all $\pi \in \Pi$ do                ▷ Calculate indirect_count
12:         if $\pi$ contains settled vertices $w, z$
13:         and $c_f(w) + c_b(z) \leq D$
14:         and $\pi$ is not directly hit by $v$ then
15:             indirect_count $\leftarrow$ indirect_count $+1$
16:         end if
17:     end for
18:
19:     count($v$) $\leftarrow$ direct_count + indirect_count
20: end for
21:
22: while $\Pi \neq \emptyset$ do                   ▷ Until every set is hit
23:     $v_{max} \leftarrow$ node with highest value for count($v$)
24:     $\mathcal{H} \leftarrow \mathcal{H} \cup \{v_{max}\}$
25:
26:     $c_f(\cdot) \leftarrow$ forward Dijkstra from $v_{max}$ until dist. $D$
27:     $c_b(\cdot) \leftarrow$ backward Dijkstra from $v_{max}$ until dist. $D$
28:     for all $\pi \in \Pi$ do          ▷ Remove sets containing $v_{max}$
29:         if $\pi$ is directly hit by $v_{max}$
30:         or ($\pi$ contains settled vertices $w, z$
31:         and $c_f(w) + c_b(z) \leq D$) then
32:             $\Pi \leftarrow \Pi \setminus \{\pi\}$
33:             Adjust count($v$) for all nodes $v$ hit by $\pi$
34:         end if
35:     end for
36: end while
37:
38: return $\mathcal{H}$
```

Dijkstra run from v and a sweep over Π). Now every path π_i that would increment the counter is removed from Π. To find all vertices with affected counters by those removals, we run the explicit thickening procedure for every such π_i (l. 28-35). For the nodes in the temporarily constructed set S_i, we decrement their counters (l. 33). Hence at the end of the round, we again have the correct hit numbers available for each vertex. Therefore the greedy algorithm picks the same solution when applying implicit thickening as when using explicit thickening.

Heuristic Solutions

Our experiments show that while implicit thickening reduces the space consumption compared to explicit thickening, it is still very time consuming to update the counters during the greedy algorithm. We now introduce heuristics which have the potential to be more efficient than explicit or implicit thickening; but unfortunately they no longer guarantee an a priori approximation factor. To show that the computed solutions are still close-to-optimal for real-world instances, we devise an algorithm to compute good instance based lower bounds for the optimal solution, and later on compare our solutions to these lower bounds.

Batched Computation

The space consumption as well as the runtime of the greedy algorithm depend on the size of the system. One idea to reduce the computational effort is to construct the Hitting Set in batches by partitioning the vertex set V into $V_1, V_2, \dots V_k$. First we compute a Hitting Set L for the Hitting Set instance induced by the $B - 2D$ violating paths originating from nodes of V_1. Then we continue with the Hitting Set instance induced by paths originating from nodes of V_2 but discarding all sets that are already hit by L from the first round. We add the new hitters to L, and repeat the procedure up to V_k. The smaller the partitions are, the faster the individual hitting set computations can be performed and the less space is required. On the other hand, if we use too small partitions, we expect the resulting Hitting Sets to be of worse quality since we lose the global view on the problem. A natural choice for the partitioning is along a space-filling curve such that the individual instances fit well into main memory.

Lazy Greedy

To speed up the instance construction as well as the greedy algorithm, we could just ignore detours in the first place. So we extract the set of minimal shortest paths longer than $B - 2D$ and compute hit counters for the greedy algorithm on that basis. But now, if we have decided for a vertex v to be part of the solution, we do not only remove paths from the set that are directly hit by v, but also the ones indirectly hit. For that purpose, we proceed like in the implicit thickening approach. All directly and indirectly hit paths are removed and the hit counters of contained nodes are updated accordingly. This algorithm does not do justice to vertices which hit very few paths directly and many indirectly. But as in general it can be assumed that these two values are strongly correlated, the solution quality should not be affected too severely.

Pruning

Going one step further, we could compute a preliminary solution without allowing detours at first. For this scenario efficient algorithms (e.g., an incremental method) were described in (Funke, Nusser, and Storandt 2014). Then, in a post-processing step, we try to decrease the solution size by removing hitters that are superfluous when allowing detours. For that purpose, we consider the hitters in the preliminary solution in some order. For every hitter we check whether there exists a so called witness path in the network which is solely hit by this particular hitter, now also considering indirect hitting wrt D. If such a path exists, the hitter has to be part of the final solution. Otherwise we can prune the hitter away, as the remaining set is still a valid loading station cover. To find a witness path, we again use the method described for implicit thickening: We first identify all paths directly and indirectly hit by this node. Then we use explicit thickening for all these paths.

Note, that pruning can be also be used for post-processing a Hitting Set retrieved by batched computation.

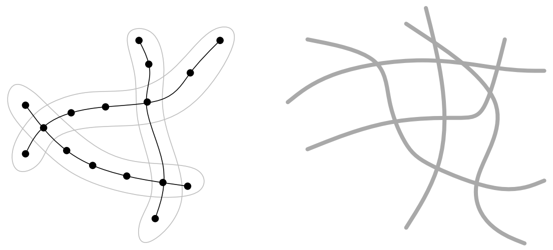

Figure 3: Set of three paths, on the left, and five (stylized) paths on the right. As for both instances all contained paths intersect, the naive lower bound is 1, respectively. But observing that there is no intersection vertex for more than two paths, we get an improved lower bound of $\lceil 3/2 \rceil = 2$ for the left instance and $\lceil 5/2 \rceil = 3$ for the right one.

Lower Bounds

A naive lower bound can be computed by extracting a set of pairwise non-intersecting thickened paths. Obviously, for every path in this set an extra hitter is required. But the larger D is chosen, the looser this lower bound might become.

Another way of constructing a lower bound is to find the hitter that hits most paths, let's say k, and divide the number of paths in the set system by k. But again, this lower bound might be far from tight if some vertex hits many paths, but most of the others only a few. However, this approach can also be applied to a subset of paths. So we aim for extracting a large subset of paths with the best vertex hitting not too many paths at once. Computing a set of disjoint paths can also be seen as an incarnation of that idea. In that case we have a subset, where the best vertex hits only a single path (i.e., $k = 1$). We can generalize the naive lower bound calculation for arbitrary k as follows. We iteratively extract thickened paths. For every vertex $v \in V$ we store a counter, that initially is set to zero, and increments if a path containing v is selected. If for some extracted path a counter would increase above k, we disregard the path. Otherwise we say the path is valid. Let q be the number of paths that are declared valid. Then $\lceil q/k \rceil$ is a lower bound for the optimal solution, as obviously no vertex can hit more than k paths at the same time. In Figure 3, small examples are provided, where $k = 2$ yields a better (larger) lower bound than $k = 1$.

Route Planning in Covered Networks

Having placed loading stations with our or an alternative approach we are still faced with the question on how to actually plan stops at loading stations when travelling with an EV. For a given set of loading stations one is typically interested in minimizing the number of recharging stops as well as the incurred detour. In the following, we briefly sketch the idea of our planning algorithm.

For given $s, t \in V$, we first compute the shortest path π from s to t conventionally by a Dijkstra run. Let R denote the length of the shortest path. If $R \leq B$, reloading is not necessary at all and we are done. Otherwise, we consider the prefix π' of length $B - 2D$ of the shortest path. For each node in π', we start a Dijkstra run up to distance D to identify the set of potential first loading stations L' and their minimal distance $d(l \in L')$ from π'. From each loading station $l \in L'$ we also run Dijkstra, now up to distance $D - d(l)$, to get the possible re-entry vertices v_l on π. We choose the loading station with the latest re-entry vertex v^* (i.e. with the largest distance from s) as the first loading station to visit on the route. Then, we consider the subpath from this re-entry vertex v^* to t as our new π and proceed recursively. The procedure stops when the distance R of the remaining shortest path to t drops below $B - D$.

Note that both the Dijkstras from each node in π' as well as the Dijkstras from all loading stations can be unified and implemented by a single Dijkstra with adjusted initial distance labels to accelerate the computation

We now prove that our planning algorithm always finds a route from s to t such that the EV never runs out of energy.

Lemma 3. *The described heuristic always returns a feasible route.*

Proof. The battery load in a re-entry vertex v^* is always $\geq B - D$ (as the path from the loading station back to π is no longer than D). Our loading station placement guarantees that for every shortest path of length $B - 2D$ there is a usable loading station which induces a detour of at most D. So the set L' is never empty and we can always reach every loading station in L' from v^* as the distance is bounded by $B - 2D + D = B - D$. Therefore, our planning algorithm always returns a feasible route to the target t. ∎

In addition to feasibility, we prove an upper bound on the number of reloading stops on the returned route.

Lemma 4. *The described planning algorithm returns a route with at most $2R/(B - 3D)$ reloading stops.*

Proof. Let $l_1, l_2, \cdots, l_k$ be the sequence of reloading stops computed by our algorithm, and $v_1, v_2, \cdots, v_k$ the respective re-entry vertices on π. By construction, v_{i+1} always appears later on π than v_i. For consecutive re-entry vertices v_i, v_{i+1} with a distance d, we know that v_{i+2} is at least $B - 3D - d$ away from v_{i+1}. Otherwise v_i and v_{i+2} would be less than $B - 3D$ apart from each other, and as the exit vertex v_{ex} for l_{i+2} can be no more than D away from v_{i+2}, v_i and v_{ex} would be within a distance of $B - 2D$. But in that case l_{i+2} would have been in the set of potential loading stations at the moment v_i was considered. And as v_{i+2} comes later on π than v_{i+1}, the loading station l_{i+2} would have been chosen instead of l_{i+1}. Hence, on every path section of length $B - 3D$ that starts in a re-entry vertex, there can be at most one other re-entry vertex. Therefore, we can subdivide π into intersection-free segments of length $\geq B - 3D$, each containing at most 2 re-entry vertices from loading stations. No more than $R/(B - 3D)$ such segments can be on a path of length R, so there at most $2R/(B-3D)$ reloading stops on the route. ∎

	# nodes	# edges	avg. path	B
SA	78,413	151,009	12.5 km	5km
SL	279,268	553,662	30.6 km	15 km
TU	669,875	1,375,845	64.5 km	30 km
ST	1,012,381	2,059,668	64.4 km	30 km
SWG	2,362,948	4,833,341	132.7 km	60 km
SG	6,546,614	13,367,955	193,7 km	100 km
GER	21,721,465	44,108,723	364.0 km	150 km

Table 1: Benchmarks graphs (SA – Saarbrücken, SL – Saarland, TU – Tübingen, ST – Stuttgart, SWG – South-West-Germany, SG – Southern Germany, GER – Germany). The average path length in kilometers was computed by running 10,000 random shortest path queries in the graph.

This directly implies that the maximal detour induced by visiting loading stations is bounded by $2DR/(B-3D)$. Our experimental evaluation will show that in practice the solution quality is even better.

Modifications for the Two Metric Case

If the metric which determines on which paths the EV runs out of energy is different from the euclidean distance, e.g. taking the underlying terrain into account, we have to make certain modifications for our algorithms to work. So now the cruising bound B denotes the maximal allowed energy consumption (e.g., in kWh), while the detour bound D is still a distance. For the correct Hitting Set formulation, we first have to compute the maximal energy consumption E of a path of length D in the network. Then hitting all shortest paths with an energy consumption of $B - 2E$ guarantees global feasibility. To extract these shortest paths we run conventional Dijkstra and check every time we settle a node if the respective path has an energy consumption $\geq B - 2E$. As soon as every node in the Dijkstra priority queue lies on a path which is $B - 2E$ violating, we can abort the Dijkstra run and backtrack the paths. Thickening and all other methods can then be performed as described above.

Experimental Study

In this section, we evaluate our proposed algorithms on real-world instances, providing results on space consumption, runtime and solution quality.

Data and Settings

We implemented all described algorithms in C++ and benchmarked them on an Intel(R) i7-3770K CPU with 3.40GHz and 32GB RAM. For evaluation we used real-world street networks of varying size extracted from OSM[1]. Table 1 lists the characteristics of the networks. Furthermore, we fixed for every network a cruising range bound B for the EV, which is about half of the average path length in the respective network. For GER, the resulting bound B = 150km

[1]http://www.openstreetmap.org

	# sets	∅ set size	space	time
SA	2.43M	1,253	14.8 GB	6 min
SL	[17.94M]	[3,988]	[286.1 GB]	[2 h]
TU	[56.28M]	[9,084]	[2.0 TB]	[21 h]
ST	[114.94M]	[13,716]	[6.3 TB]	[89 h]
SWG	[396.61M]	[18,797]	[29.8 TB]	[538 h]
SG	[2313.78M]	[33,085]	[306.2 TB]	[291 d]
GER	[12420.95M]	[47,176]	[2343.0 TB]	[7 y]

Table 2: Experimental results for the explicit construction of the set system using thickening. M denotes millions, h hours, d days and y years. Values in brackets are extrapolated, as the complete experiment would have required too much space.

	# sets	∅ set size	space	time
SA	2.43M	120	1.4 GB	1 min
SL	17.94M	328	23.5 GB	30 min
TU	[56.28M]	[510]	[114.8 GB]	[2 h]
ST	[114.94M]	[576]	[264.8 GB]	[6 h]
SWG	[396.61M]	[1,222]	[1.9 TB]	[55 h]
SG	[2313.78M]	[2,018]	[18.7 TB]	[695 h]
GER	[12420.95M]	[2,768]	[137.6 TB]	[278 d]

Table 3: Experimental results for constructing the set system without thickening. Values in brackets are extrapolated.

roughly corresponds to the real bound for an average EV. The smaller graphs are considered to illustrate the scalability of our approaches. If not indicated otherwise, the detour bound is fixed to $D = 1$km for SA, $D = 2$km for SL, and $D = 5$km for the larger graphs. We think that a maximum detour of 5km per reloading event reflects best what an EV driver deems acceptable when going on longer tours.

Instance Construction and Greedy Solution

We first evaluated the explicit thickening approach on our benchmark graphs. The results are collected in Table 2.

We observe that already for small graphs (SL, TU) the (extrapolated) space consumption is enormous. For larger graphs this approach is not applicable at all since storage for the set system as well as extraction time explode. Only for the very small SA graph complete construction of the set system is possible.

For implicit thickening we only have to extract and store all minimal $B - 2D$ violating paths. The respective results are shown in Table 3. The average set sizes and therefore the total space consumption are significantly reduced compared to Table 2. Therefore the SL graph can be tackled with implicit thickening while explicit thickening is not applicable. For the larger graphs implicit thickening also demands way too much space to be practical.

For the SA graph, running greedy on the explicit set system took 44 seconds, and results in a solution of size 205. As we have an approximation guarantee of $\ln(|\mathcal{S}|)$, we know that in the SA graph 205 is at most a factor of 14.7 away from the optimal solution. For implicit thickening, the solution size was the same, of course, but the runtime increased to 1.5 hours as updating the hit counters in every round is much more costly. For SL implicit thickening led to a solution of size 248 with a runtime of 62 hours. The approximation guarantee for SL using greedy is 19.0. We will see that this *a priori* approximation guarantee is in fact too pessimistic, as our instance based lower bounds show.

	batched greedy			lower bound		APX
	batch size	runtime	$\|L\|$	k=1	k=10	
SA	55,260	6 min	212	86	94	2.25
SL	8,677	1.9 h	255	64	73	3.49
TU	2,603	22.5 h	308	49	64	4.81
ST	1,214	76.5 h	428	61	81	5.93
SWG	–	–	–	56	73	–
SG	–	–	–	60	89	–
GER	–	–	–	104	165	–

Table 4: Batched greedy Hitting Set computation. APX depicts the ratio of $|L|$ and the largest lower bound, hence APX is an upper bound on the approximation quality.

Heuristic Solutions

Next we study our heuristics which were designed to improve on the approximation algorithms in space or time.

Batched Computation For the graphs in Table 2 with a manageable extraction time but way too high space consumption, we can use batched computation to construct a solution. We used batches such that according to our extrapolations the respective set systems fit into 8GB of RAM, see Table 4. This allowed us to tackle the TU and the ST graphs which were intractable before.

Along with those results, in Table 4 we provide lower bounds for the size of L by applying our improved lower bound construction. We see that for $k = 10$ the lower bounds are clearly larger than for $k = 1$. Comparing the size of L to this lower bound on an instance basis, we can make a statement about the quality of the obtained solution. The resulting approximation ratios (APX in the Table), are all below 6, indicating that our computed sets of loading stations are close-to-optimal.

Lazy Greedy Lazy greedy improves on implicit thickening by the ability to compute hit counters much faster. For SA we computed a solution of size in 232 in 2 minutes, for SL a solution of size 487 in 34 minutes (including set extraction, see Table 3). So the solution quality decreases compared to the other greedy approaches but we require significantly less space and time. We can also combine batched computation and lazy greedy to tackle larger graphs. Using both approaches, we can handle besides TU (552 hitters in 1.8 hours) and ST (762 hitters in 4.8 hours) even the SWG network (642 hitters in 38.1 hours).

Pruning Having an initial solution available which hits all $B - 2D$ violating paths, we can apply pruning. We extracted such initial solutions for all our test graphs using the methods described in (Funke, Nusser, and Storandt 2014). The results for post-processing with pruning are presented in Table 5. We observe that in contrast to the previously discussed approaches, pruning can be applied to all benchmark

	initial solution		pruning			
	time	$\|L\|$	time	$\|L\|$	Δ	APX
SA	24 s	445	1 s	312	-29.9%	3.32
SL	235 s	459	14 s	370	-19.4%	5.07
TU	19 min	573	6 min	412	-28.1%	6.44
ST	45 min	777	19 min	544	-30.0%	6.72
SWG	5.5 h	523	6 min	488	-6.7%	6.68
SG	2.7 h	920	31 min	834	-9.4%	9.37
GER	11.5 h	1809	5 h	1618	-10.6%	9.81

Table 5: Results for applying the pruning approach to precomputed Hitting Sets.

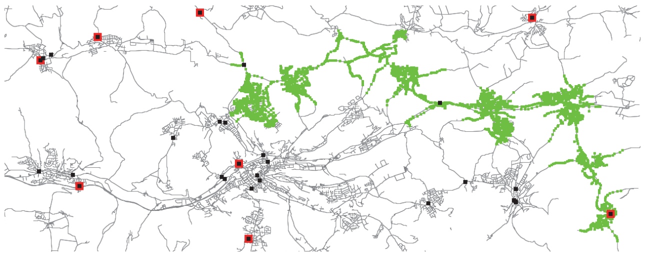

Figure 4: Initial loading stations (black) and remaining loadings stations after pruning (red). The green nodes indicate a thickened path which is only hit by the red loading station on the right, indicating its necessity for the final solution.

instances. On average pruning decreases the initial solution size by about a fifth. An illustration of the pruning approach can be found in Figure 4. When considering the smaller graphs it is evident that the greedy algorithm produces conciser Hitting Sets. But for the larger graphs pruning is the only applicable method. Still, our lower bounds imply that we obtain solutions of good quality using pruning; the APX ratio stays below 10.

If we increase the allowed detour, valid solutions naturally become even smaller. For example, for the SG graph, a detour of $D = 10$ km resulted in a solution of $|L| = 792$, a detour of 20 km led to a solution of size 722. When applying pruning to the solution from the batched computation, we could decrease its size for SWG from 642 to 451, resulting in an improved approximation ratio of 6.17.

Route Planning

Finally, we use our computed loading station placement for ST (with B=30km, D=5km, 428 loading stations) as basis for route planning. We applied the described planning algorithm to compute feasible routes with few reloading stops. Table 6 summarizes the results. We observe that the average number of reloading events per route is never more than 1 above the lower bound (computed by comparing the shortest path length to the bound B) on average, and no more than 3 in the worst case. Furthermore, we see that the detour induced by reloading only increases the route length by 7% to 9% compared to the shortest path. Query times never exceeded 500 milliseconds. The results show that our model and our computed routes are practical.

$\|\pi(s,t)\|$	LB	avg. # LS	max # LS	ratio
0 - 30 km	0	0.00	0	1.0000
31 - 60 km	1	1.22	2	1.0713
61 - 90 km	2	2.41	4	1.0812
91 - 120 km	3	3.61	6	1.0879
121 - 150 km	4	4.71	6	1.0891
151 - 180 km	5	5.91	6	1.0838

Table 6: Experimental results for our route planning routine on the ST network, subdivided by shortest path length $|\pi(s,t)|$. 'LB' indicates the lower bound for the number of reloading events, avg. and max # LS denote the average and maximal number of loading stations in the computed solution. The ratio is defined by the route length (including detours) and the shortest path length. Avg./max # LS and ratio are based on 1,000 random queries.

We also computed the average distance between two reloading stops, which is 24,079.5 meters and hence almost equals $B - D = 25{,}000$ meters. So the natural strategy for driving on the shortest path without planning ahead would be to just look for close-by loading stations when the battery load approaches D plus some safety margin (very similar to the scenario with a conventional car).

Conclusions and Future Work

We introduced a new graph covering problem in the e-mobility context motivated by the desire of EV-drivers to be somewhat oblivious to the location of loading stations when planning their trips with the EV but accepting moderate detours for recharging. Modeling this problem as a Hitting Set problem we were facing various challenges even setting up the problem instance, so specialized algorithms avoiding the explicit instance construction had to be developed. We designed greedy-like algorithms which achieve very good results but are currently limited to smaller networks. For such networks, e.g. the road network of the metropolitan area around Stuttgart, the approach by (Funke, Nusser, and Storandt 2014) which does not allow detours constructs 777 loading stations, our new pruning approach 544, and our new batched approach 428, so in total a more than 40% reduction of required loading stations by allowing detours (for a parameter set aims at short-range EVs like the electric bike ELMOTO HR-2 (Elmoto 2015)). The pruning approach computes solutions even for country-sized networks but unfortunately of worse quality. Our current focus is on tuning the greedy-type algorithms to deal with the largest networks as well. While our algorithms can be quite easily adapted to other energy consumption models or car characteristics, the incorporation of traffic volume estimations and route popularity are interesting further research venues. Our current route planning algorithm requires the execution of several Dijkstra runs, which is acceptable for a single route computation. In a client-server scenario, where thousands of users ask for route suggestions at the same time, some preprocessing scheme must be developed to answer these queries more efficiently.

References

Artmeier, A.; Haselmayr, J.; Leucker, M.; and Sachenbacher, M. 2010. The shortest path problem revisited: Optimal routing for electric vehicles. In *KI 2010: Advances in Artificial Intelligence, 33rd Annual German Conference on AI, Karlsruhe, Germany, September 21-24, 2010. Proceedings*, 309–316.

Baouche, F.; Billot, R.; Trigui, R.; and El Faouzi, N. E. 2014. Electric vehicle green routing with possible en-route recharging. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, 2787–2792.

Chen, T. D.; Kockelman, K. M.; Khan, M.; et al. 2013. The electric vehicle charging station location problem: a parking-based assignment method for seattle. In *Transportation Research Board 92nd Annual Meeting*, volume 340, 13–1254.

Elmoto. 2015. Hr-2 evo. http://www.elmoto.com/en/en-bikes/en-hr2-evo/.

Frade, I.; Ribeiro, A.; Gonçalves, G.; and Antunes, A. 2011. Optimal location of charging stations for electric vehicles in a neighborhood in lisbon, portugal. *Transportation research record: journal of the transportation research board* (2252):91–98.

Funke, S.; Nusser, A.; and Storandt, S. 2014. Placement of loading stations for electric vehicles: No detours necessary! In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 417–423.

Goodrich, M. T., and Pszona, P. 2014. Two-phase bicriterion search for finding fast and efficient electric vehicle routes. *CoRR* abs/1409.3192.

Hayakawa, T.; Ishikawa, K.; Imura, J.-i.; Tanaka, H.; Toyoshima, M.; and Iwai, A. 2014. Incentive based multi-objective optimization in electric vehicle navigation including battery charging. In *Proc. 19th World Congress of the Int. Federation of Automatic Control*.

He, F.; Wu, D.; Yin, Y.; and Guan, Y. 2013. Optimal deployment of public charging stations for plug-in hybrid electric vehicles. *Transportation Research Part B: Methodological* 47:87–101.

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

Liu, Z.; Wen, F.; and Ledwich, G. 2013. Optimal planning of electric-vehicle charging stations in distribution systems. *Power Delivery, IEEE Transactions on* 28(1):102–110.

Storandt, S., and Funke, S. 2012. Cruising with a battery-powered vehicle and not getting stranded. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

Storandt, S., and Funke, S. 2013. Enabling e-mobility: Facility location for battery loading stations. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*

Storandt, S. 2012. Quick and energy-efficient routes: computing constrained shortest paths for electric vehicles. In *5th ACM SIGSPATIAL International Workshop on Computational Transportation Science 2011, CTS'12, November 6, 2012, Redondo Beach, CA, USA*, 20–25.

Sweda, T. M., and Klabjan, D. 2012. Finding minimum-cost paths for electric vehicles. In *Electric Vehicle Conference (IEVC), 2012 IEEE International*, 1–4. IEEE.

Wang, H.; Huang, Q.; Zhang, C.; and Xia, A. 2010. A novel approach for the layout of electric vehicle charging station. In *Apperceiving Computing and Intelligence Analysis (ICACIA), 2010 International Conference on*, 64–70. IEEE.

Wang, T.; Cassandras, C. G.; and Pourazarm, S. 2014. Energy-aware vehicle routing in networks with charging nodes. *arXiv preprint arXiv:1401.6478*.

Xiong, Y.; Gan, J.; An, B.; Miao, C.; and Bazzan, A. L. C. 2015. Optimal electric vehicle charging station placement. In Yang, Q., and Wooldridge, M., eds., *Proc. 24th International Joint Conference on Artificial Intelligence, IJCAI 2015*, 2662–2668. AAAI Press.