Consumption Profiles in Route Planning for Electric Vehicles: Theory and Applications*

Moritz Baum[1], Jonas Sauer[2], Dorothea Wagner[3], and Tobias Zündorf[4]

1 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
`moritz.baum@kit.edu`
2 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
`jonas.sauer@student.kit.edu`
3 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
`dorothea.wagner@kit.edu`
4 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
`zuendorf@kit.edu`

Abstract

In route planning for electric vehicles (EVs), *consumption profiles* are a functional representation of optimal energy consumption between two locations, subject to initial *state of charge (SoC)*. Efficient computation of profiles is a relevant problem on its own, but also a fundamental ingredient to many route planning approaches for EVs. In this work, we show that the complexity of a profile is at most linear in the graph size. Based on this insight, we derive a *polynomial-time* algorithm for the problem of finding an energy-optimal path between two locations that allows stops at charging stations. Exploiting efficient *profile search*, our approach also allows *partial recharging* at charging stations to save energy. In a sense, our results close the gap between efficient techniques for energy-optimal routes (based on simpler models) and $\mathcal{NP}$-hard time-constrained problems involving charging stops for EVs. We propose a practical implementation, which we carefully integrate with Contraction Hierarchies (CH) and A* search. Even though the practical variant formally drops correctness, a comprehensive experimental study on a realistic, large-scale road network reveals that it *always* finds the optimal solution in our tests and computes even long-distance routes with charging stops in less than 300 ms.

1998 ACM Subject Classification G.2.2 Graph Theory, G.2.3 Applications

Keywords and phrases electric vehicles, charging station, shortest paths, route planning, profile search, algorithm engineering

Digital Object Identifier 10.4230/LIPIcs.SEA.2017.19

1 Introduction

Route planning services explicitly designed for EVs have to address specific aspects, since EVs usually employ a rather limited cruising range. We study the problem of computing routes that minimize energy consumption, in order to maximize cruising range and for drivers to overcome *range anxiety* (the fear of getting stranded). This imposes nontrivial challenges. *Recharging* en route may become inevitable on long-distance trips. Given that charging stations are scarce, such stops need to be planned in advance [7, 42]. Moreover, EVs can *recuperate* energy (e. g., when going downhill), but the *battery capacity* limits the amount

* Tobias Zündorf's research was supported by DFG Research Grant WA 654/23-1.

of recuperable energy [9, 18, 36]. As a result, the energy-optimal route depends on the initial SoC. This dependency is captured by the notion of *(consumption) profiles*, which map SoC (at the source) to (minimum) energy consumption that is necessary to reach the target [9, 18, 38]. Profiles are relevant in many applications, where the SoC at the start of a journey is either unknown or can be decided by the driver, e. g., when charging overnight. Moreover, they are an important ingredient of *speedup techniques*, where preprocessing is applied to the input network for faster query times [7, 9, 18].

In this work, we examine the complexity of consumption profiles in road networks. Furthermore, we discuss an important application that requires efficient computation of profiles, namely, energy-optimal routes with intermediate charging stops. Even when optimizing for energy consumption only, the integration of charging stations into route planning is a nontrivial task: Recharging to a full battery at a charging station can be wasteful if it prevents the battery from recuperating energy on a downhill ride later on. Hence, profiles help in deciding the optimal amount of energy to be recharged at a station.

Related Work. Classic route planning approaches apply Dijkstra's algorithm [17] to a graph representation of the network, with fixed scalar edge costs representing, e. g., travel time. For faster running times in practice, *speedup techniques* [4] accelerate online shortest-path queries with data *preprocessed* in an offline phase. Examples of such techniques are CH [20], where vertices are contracted iteratively and replaced by *shortcuts* in the graph, and variants of A* search [21, 24]. Combining both techniques, Core-ALT [6] contracts most vertices and runs A* on the remaining *core graph.* Some techniques were also extended to more complex scenarios, such as time-dependent cost functions [5, 10, 12, 14]. In this context, a *profile query* asks for a functional representation of travel time between locations for *any* departure time. Such functions may have superpolynomial complexity [19], but can be computed by an output-sensitive search algorithm [11, 15]. See Bast et al. [4] for a more complete survey.

Regarding route planning for EVs, computing routes that minimize energy consumption requires the integration of battery constraints into Dijkstra's algorithm and adaptation of speedup techniques for fast queries [9, 18, 36]. Eisner et al. [18] observe that consumption profiles have *constant* complexity for a fixed *path.* Subsequent works consider *general* profiles possibly comprising *multiple* paths [9, 38], but their complexity has not been studied. Stops at charging stations are often considered under the simplifying assumptions that the charging always results in a *fully* recharged battery [22, 32, 40, 41, 42, 43]. Routes with a minimum number of intermediate charging stops can then be computed in less than a second on subcountry-scale graphs [41, 42]. More complex models also consider constraints on time spent driving [8, 30] and recharging [7, 27, 31, 33, 44, 45], but this results in much more difficult (typically $\mathcal{NP}$-hard) problems and proposed techniques are inexact or impractical.

Contribution and Outline. In Section 2, we formally introduce our model, describe problem variants, and recap basic algorithmic ingredients needed to solve them. In Section 3, we investigate the complexity of consumption profiles. As our main result, we prove that such profiles have linear complexity—much in contrast to profiles in time-dependent routing, which can have superpolynomial size [19]. This enables us to compute *profile search* efficiently—an algorithm relevant in various query scenarios and a crucial ingredient for many speedup techniques. In Section 4, we consider energy-optimal routes that allow stops at charging stations to recharge the battery. Unlike previous studies [22, 40, 41], we do not assume that using a charging station always results in a fully recharged battery. Instead, we allow the charging process to be *interrupted* beforehand to save energy. Building upon our theoretical findings, we derive a *polynomial-time* algorithm to solve the problem. To make the approach fast in practice, we propose a (heuristic) variant and integrate it with CH [18, 20] and A* search [24]. Section 5 presents our experimental study, in which we demonstrate that our algorithm (empirically) obtains *optimal* results for all queries in well below a second after moderate preprocessing effort. We conclude with final remarks in Section 6.

2 Model, Query Variants, and Basic Algorithms

We model the road network as a directed graph $G = (V, E)$. Energy consumption along edges is given by the cost function $c\colon E \to \mathbb{R}$. Consumption can be negative to model recuperation, but cycles with negative consumption are physically ruled out. An EV is equipped with a battery of limited capacity $M \in \mathbb{R}_{\geq 0}$. Given the current SoC $b_u \in [0, M]$ of a vehicle at a vertex $u \in V$, traversing an edge $(u, v) \in E$ typically results in the SoC $b_v = b_u - c(u, v)$ at v. However, we also take *battery constraints* into account: The SoC b_v must neither exceed the limit M, nor drop below zero [1, 2, 18]. Thus, if the consumption $c(u, v)$ of an edge (u, v) exceeds the SoC b_u at u, the edge cannot be traversed, as the battery would run empty along the way. We indicate this case by setting $b_v := -\infty$. Conversely, if the battery is (almost) fully charged, passing an edge with negative consumption cannot increase the SoC beyond the maximum value M, so we obtain $b_v \leq M$. Given some initial SoC b_s at the source vertex s, we say that an s–t path $P = [s, \dots, t]$ in G is *feasible* if and only if the battery never runs empty, i. e., the SoC b_v obtained at every vertex v of P after iteratively applying above constraints is within the interval $[0, M]$. Note that a path may be infeasible even if its *cost* (i. e., the sum of its consumption values) does not exceed b_s: Due to negative edge weights, there might be a prefix of greater total cost that renders the path infeasible. Given the SoC b_t at the target t of a feasible path, the *energy consumption* on the path is $b_s - b_t$. This value can become negative, due to recuperation.

For the sake of simplicity and without loss of generality, we assume in this work that $c(e) \in [-M, M]$ for all edges of the input graph. Moreover, we assume that shortest paths (wrt. the cost function c) between arbitrary pairs of vertices are unique.

Profiles. Given two vertices $s \in V$ and $t \in V$ of the input graph G, we define the *SoC function* $f\colon [0, M] \cup \{-\infty\} \to [0, M] \cup \{-\infty\}$, also called *SoC profile* or *s–t profile*, which maps SoC at the source s to the maximum SoC at the target t (after traversing any s–t path in G). We define $f(-\infty) := -\infty$. SoC functions are *piecewise linear*, so we use a sequence $F = [(x_1, y_1), \dots, (x_k, y_k)]$ of *breakpoints* to define f, such that $f(b) = -\infty$ for $b < x_1$, $f(b) = y_k$ for $b \geq x_k$, and $f(b)$ is obtained by linear interpolation in all other cases.

For some s–t path P, we denote by f_P the profile of P, i. e., the SoC function that maps initial SoC at s to the resulting SoC at t after traversing P. Given the SoC functions f_P and f_Q of two paths P and Q, we say that f_P *dominates* f_Q (similarly, P *dominates* Q) if $f_P(b) \geq f_Q(b)$ for all $b \in [0, M]$.

To give a simple example, consider the SoC function $f_{[u,v]}$ induced by an edge $(u, v) \in E$. Given the cost $c(u, v)$ of the edge, battery constraints yield

$$f_{[u,v]}(b) := \begin{cases} -\infty & \text{if } b - c(u,v) < 0, \\ M & \text{if } b - c(u,v) > M, \\ b - c(u,v) & \text{otherwise.} \end{cases}$$

The function $f_{[u,v]}$ is represented by the sequence $F_{[u,v]} = [(c(u, v), 0), (M, M - c(u, v))]$ if

$c(u,v) \geq 0$ is nonnegative, and $F_{[u,v]} = [(0, -c(u,v)), (M + c(u,v), M)]$ otherwise. In both cases, the function consists of two breakpoints.

Query Variants. Typically, there are two problem variants of interest. First, an *SoC query* consists of a source $s \in V$, a target $t \in V$, and an initial SoC $b_s \in [0, M]$. It asks for a (single) *energy-optimal* s–t path when departing at s with SoC b_s, i. e., a path that maximizes the SoC b_t at t (and minimizes the consumption $b_s - b_t$). Second, a *profile query* does not take b_s as input, but asks for an s–t *profile*, i. e., the optimal value b_t for *every* initial SoC $b_s \in [0, M]$.

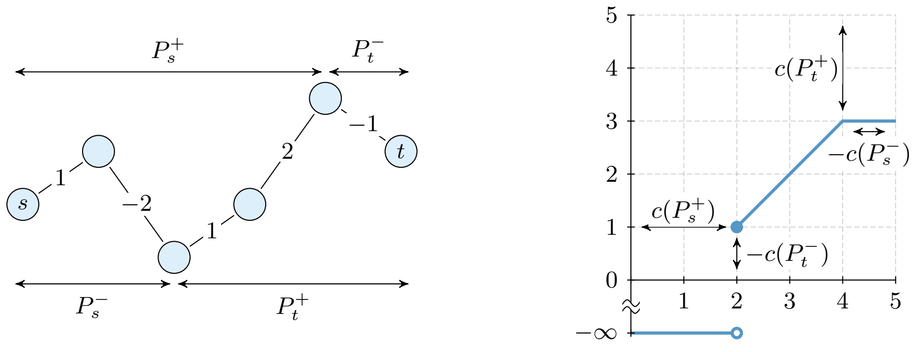
Basic Algorithms. SoC queries can be answered by a variant [1, 2] of Dijkstra's algorithm that handles battery constraints on-the-fly, using explicit checks. However, the algorithm is *label correcting*, i. e., negative edge costs may cause (exponentially many) re-scans of vertices [28]. *Potential shifting* [29] can remedy this, making use of a potential function $\pi\colon V \to \mathbb{R}$ on the vertices with the property that $c(u,v) - \pi(u) + \pi(v) \geq 0$ holds for all $(u,v) \in E$. Then the search becomes *label setting* when using the *reduced* cost function $c(u,v) - \pi(u) + \pi(v)$, i. e., each vertex is scanned at most once. Distances wrt. the original cost function are maintained and battery constraints can still be applied on-the-fly [18, 36]. We refer to this algorithm as *EV Dijkstra (EVD)*.

Profile search [9, 37, 38] is a label-correcting algorithm that answers profile queries. Starting from the source vertex $s \in V$, the algorithm maintains, for each vertex v, a (tentative) s–v profile f_v. It initializes $f_v \equiv -\infty$ for all $v \in V$, except $f_s = \mathrm{id}$. It adds s to a priority queue, which uses the value $\min_{b \in [0,M]}\{b - f_v(b)\}$ as key of a vertex v, i. e., the *minimum energy consumption* of its SoC function. In each step of the main loop, the algorithm extracts a vertex u with minimum key from the queue (thereby *scanning* it) and proceeds along the lines of Dijkstra's algorithm: Incident outgoing edges $e = (u,v)$ are *scanned* by computing the composition $f = f_e \circ f_u$. Afterwards, it sets $f_v = \max(f_v, f)$, i. e., the pointwise maximum of f_v and f, and updates v in the priority queue, if its key changed. After termination, the label f_v of every vertex $v \in V$ is the s–v profile.

3 On the Complexity of Profiles

In this section, we first examine characteristics of SoC functions of single paths. Then, we show that the complexity (i. e., number of breakpoints) of *general* SoC functions is linear in the number $|V|$ of vertices in the graph. An alternative notion that is common in the literature [7, 8, 9, 18] utilizes *consumption profiles* $g(\cdot)$, which map initial SoC to *energy consumption* between two vertices. We obtain the relation $f(b) = b - g(b)$, hence our insights on the complexity of SoC profiles carry over to consumption profiles directly.

Profiles Representing Paths. Given an s–t path P, Eisner et al. [18] show that the number of breakpoints of the SoC function f_P is bounded by a constant. Below, Lemma 1 recaps this fundamental insight, but also provides a full specification of the SoC function of a single path based on the costs of certain *subpaths*. We begin by defining *important* subpaths of an s–t path P. First, let P_s^+ denote the *maximum prefix* of P, i. e., the prefix of P that has maximum cost $c(P_s^+)$ wrt. energy consumption among all its prefixes. (Recall that the cost of a path is defined as the sum of its edge costs, hence, battery constraints do not apply.) If every prefix of P (including P itself) is negative, we obtain $P_s^+ = [s]$ and $c(P_s^+) = 0$. Similarly, the *minimum prefix* P_s^- minimizes the cost $c(P_s^-)$ among all prefixes of P. We obtain $P_s^- = [s]$ and $c(P_s^-) = 0$ in case every prefix of P is positive. The *maximum suffix*



(a) An s–t path with depicted edge costs. **(b)** The corresponding SoC function.

Figure 1 An s–t path together with its SoC function, assuming that the battery capacity is $M = 5$. The cost of the path is 1 and its important subpaths are indicated. Relative vertical positions of vertices correspond to costs of subpaths starting or ending at the respective vertex. The coordinates of breakpoints in the profile are equal to the costs of certain important subpaths.

P_t^+ and *minimum suffix* P_t^- are defined symmetrically. For the sake of simplicity, we assume that P contains no subpath with cost 0 consisting of more than one vertex (this can be enforced by perturbation of edge costs). Thus, the subpaths above are uniquely defined. Moreover, observe that $P = P_s^- \circ P_t^+ = P_s^+ \circ P_t^-$; see Figure 1.

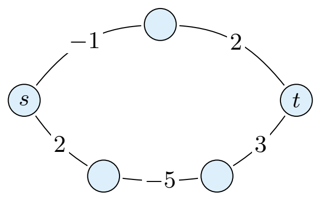
Lemma 1 shows that the SoC function f_P (defined by its breakpoints) of a path P is completely determined by the costs of its important subpaths. At most two breakpoints are necessary to represent the SoC function. It has a characteristic form: It consists of a first part with infinite consumption (the path is infeasible for low SoC), followed by a segment with slope 1 (the consumption is constant, thus SoC at t increases with SoC at s), and a last segment of constant SoC (for high values of initial SoC, the battery is fully charged at some point due to recuperation). Each of these three intervals may collapse to a single point. The segment with slope 1 is also called the *characteristic* segment of the SoC function. An example of a path and its SoC function is depicted in Figure 1.

▶ **Lemma 1.** *Given an s–t path P, its SoC function f_P is a piecewise linear function. It is defined by a sequence F of breakpoints in the following way.*
1. *If there exists a subpath of P with cost greater than M, $F = \emptyset$ and $f_P \equiv -\infty$.*
2. *Otherwise, if there is a subpath of P with cost below $-M$, $F = [(c(P_s^+), M - c(P_t^+))]$.*
3. *If neither such subpath exists, $F = [(c(P_s^+), -c(P_t^-)), (M + c(P_s^-), M - c(P_t^+))]$.*

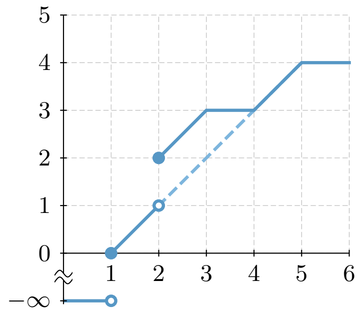
General Profiles. For a pair of vertices s and t, different paths may be the optimal choice for different values of initial SoC; see Figure 2. Therefore, a general SoC function is the upper envelope of a set of SoC functions, each corresponding to a single path. We say that an s–t path *contributes* to the s–t profile if it is optimal for some initial SoC. We bound the number of breakpoints in the SoC function subject to the number of contributing paths. The following Lemma 2 is a direct implication of the observations by Atallah [3]. (In general, the number of breakpoints in the upper envelope of linear functions can be superlinear [46].)

▶ **Lemma 2.** *Given the set $\mathcal{P}$ of all contributing paths of an s–t profile, the number of breakpoints in the SoC function is linear in $|\mathcal{P}|$.*

Since the number of s–t paths can be exponential in the graph size, Lemma 2 does not yield an immediate polynomial bound on the complexity of the s–t profile. Note that in the



(a) Graph with vertices s and t.



(b) The corresponding SoC function.

Figure 2 The SoC profile of given vertices s and t. The battery capacity is $M = 6$. The dashed segment indicates dominated parts of the SoC function of the upper s–t path.

related scenario of time-dependent profiles, it was shown that the number of contributing paths can actually become superpolynomial in the graph size [19]. In contrast, we now show that the number of breakpoints of an SoC function is in fact *linear* in the number of vertices of the input graph in the worst case.

We start with basic properties of paths and their SoC functions. Lemma 3 claims that a path P dominates another path Q if it is shorter (wrt. the cost function c) and both its maximum prefix and maximum suffix are shorter than the respective subpaths of Q. This follows immediately from the structure of SoC functions and is illustrated in Figure 3.

▶ **Lemma 3.** *Let P and Q be two s–t paths, such that $c(P_s^+) \leq c(Q_s^+)$, $c(P_t^+) \leq c(Q_t^+)$, and $c(P) \leq c(Q)$. Then the SoC function f_P of P dominates the SoC function f_Q of Q.*

As argued above, certain subpaths of an s–t path P are relevant to determine the corresponding profile. We add the following definitions. The *bottom vertex* v^- is the last vertex of the minimum prefix (and the first vertex of the maximum suffix) of P. Similarly, the *top vertex* v^+ denotes the last vertex of the maximum prefix (and the first vertex of the minimum suffix) of P. We call v^- and v^+ the *important* vertices of P. We presume that $v^- \neq v^+$, which always holds except in the trivial case $s = t$. The important vertices then separate P into three subpaths. (In case s or t are important vertices, one or two of these subpaths may consist of a single vertex.) Moreover, we distinguish two *types* of paths, depending on the order of appearance of important vertices. A path is called *top-bottom path* if v^+ appears before v^-, otherwise it is a *bottom-top path.* Lemma 4 states that prefixes and suffixes of contributing paths are uniquely defined by their corresponding important vertices.

▶ **Lemma 4.** *Given two vertices $s \in V$ and $t \in V$, let $v \in V$ be an arbitrary fixed vertex. All paths of the same type contributing to the s–t profile with v as their first important vertex share the same s–v subpath. Moreover, all contributing paths of the same type with v as their second important vertex share the same v–t subpath.*

Proof. Assume for contradiction that there are two contributing paths P and Q of the same type, such that the first important vertex of each path is v, but their respective s–v subpaths differ. Without loss of generality, let the s–v subpath of P be shorter. We replace the s–v subpath of Q by the subpath of P, which yields a modified path Q'. Clearly, the total length of Q' is below the length of Q, i. e., $c(Q') < c(Q)$. At the same time, neither the maximum prefix nor the maximum suffix of Q' exceeds the cost of the respective subpath

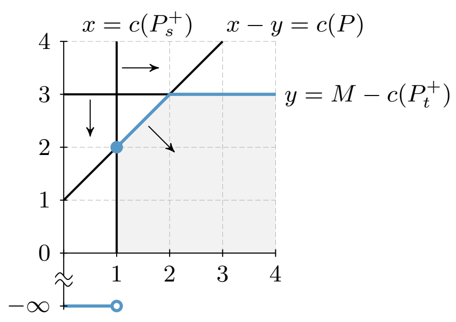


Figure 3 Dominated area of an SoC function, for a path P with $c(P) = -1$. Its maximum prefix and maximum suffix have cost $c(P_s^+) = c(P_t^+) = 1$. The costs induce three lines, each of which subdivides the Euclidean plane into two half planes. The SoC function of a path Q with $c(Q) \geq c(P)$, $c(Q_s^+) \geq c(P_s^+)$, and $c(Q_t^+) \geq c(P_t^+)$ lies in the shaded intersection of three of these half planes.

of Q. By Lemma 3, the modified path Q' dominates Q, contradicting the assumption that Q is a contributing path.

Similarly, we can replace the v–t subpath in one of two paths of the same type that share the second important vertex v by a shorter v–t subpath. Again, we obtain a new path that is shorter, while the lengths of its maximum prefix and suffix do not increase. Hence, at least one of the two paths does not contribute to the profile. ◀

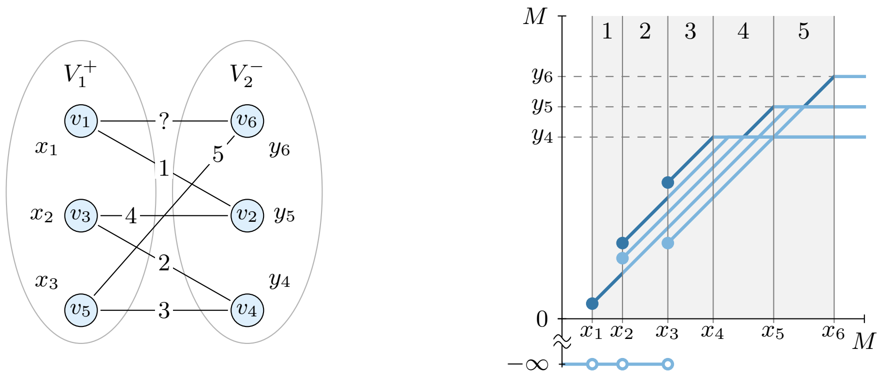
Below, Lemma 5 shows that together with their order in the path, pairs of important vertices uniquely define contributing paths of the same type. Note that this already implies that there are at most $\mathcal{O}(|V|^2)$ paths contributing to an s–t profile. Afterwards, we use a somewhat more sophisticated argument to show that the number of breakpoints is at most linear in the number of vertices.

▶ **Lemma 5.** *Let $s \in V$, $t \in V$, $v^- \in V$, and $v^+ \in V$ be four vertices of the input graph. There is at most one bottom-top path contributing to the s–t profile that has v^- as its bottom vertex and v^+ as its top vertex. Similarly, at most one contributing top-bottom path has v^+ as its top vertex and v^- as its bottom vertex.*

Proof. Assume for contradiction that there exist two distinct contributing s–t paths P and Q, such that both are bottom-top paths, their bottom vertex is v^-, and their top vertex is v^+. By Lemma 4, we know that P and Q share the same s–v^- subpath and the same v^+–t path. Hence, their v^-–v^+ subpaths must differ. Without loss of generality, let the v^-–v^+ subpath of P be shorter. Apparently, the total cost of the path P is lower than the cost of Q, i. e., $c(P) < c(Q)$. Similarly, the cost of the maximum prefix (suffix) of P is at most the cost of the maximum prefix (suffix) of Q. By Lemma 3, this implies that P dominates Q, contradicting the fact that Q contributes to the optimal solution. The other case is symmetric, so the claim follows. ◀

Using a somewhat more sophisticated argument, Theorem 6 shows that the number of breakpoints is at most linear in the number of vertices. It is easy to construct an example where the number of breakpoints in the SoC function is in fact linear in $|V|$, so this bound is tight up to a constant factor. This also enables profile search to run in $\mathcal{O}(|V|^2 \log |V|)$ time.

▶ **Theorem 6.** *Given a source $s \in V$ and a target $t \in V$ in the input graph $G = (V, E)$, the number of contributing paths (and breakpoints) in the s–t profile is in $\mathcal{O}(|V|)$.*



(a) A constructed bipartite graph. **(b)** The SoC functions of the edges.

Figure 4 Illustration of the proof of Theorem 6. The constructed bipartite graph G' contains copies of top and bottom vertices of the input graph. Edges represent paths connecting certain important vertices. Vertices have assigned constants $x_1, x_2, x_3, y_4, y_5, y_6$. Edge labels indicate intervals depicted in Figure 4b, where the corresponding characteristic segments are contained in the upper envelope. Characteristic segments connect vertical lines induced by constants $x_1, \dots, x_6$. Parts of characteristic segments that lie on the upper envelope are highlighted (dark blue). Adding the missing characteristic segment that connects the lines induced by x_1 and x_6 (to form a cycle in the graph) results in at least one dominated SoC function.

Proof. To prove the claim, we construct a graph G' consisting of vertices representing important vertices in G. Edges of G' represent contributing paths between pairs of important vertices. We examine the structure of SoC functions of contributing paths to show that the number of edges in this graph is in $\mathcal{O}(|V|)$.

We construct an undirected graph G' that consists of the union of four sets of vertices $V_1^- = \{v_1^- \mid v \in V\}$, $V_1^+ = \{v_1^+ \mid v \in V\}$, $V_2^- = \{v_2^- \mid v \in V\}$, and $V_2^+ = \{v_2^+ \mid v \in V\}$. Clearly, the number of vertices in G' is linear in the number of vertices in the original graph. We add one undirected edge for every s–t path in the original graph that contributes to the SoC function: For every contributing bottom-top path with first important vertex u and second important vertex w, we add the edge $\{u_1^-, w_2^+\}$. For every contributing top-bottom path with first important vertex u and second important vertex w, we add the edge $\{u_1^+, w_2^-\}$. Lemma 5 implies that there are no multi-edges in the resulting graph. By construction, G' consists of at least two components and each component induces a bipartite subgraph. We claim that G' contains no cycles. This implies that the resulting graph has at most $\mathcal{O}(|V|)$ edges, which proves the theorem.

Assume for contradiction that there is a cycle $C = [v_1, \dots, v_k, v_1]$ in the graph constructed above. There are two possible cases: Either all edges in the cycle correspond to top-bottom paths and it contains only vertices in $V_1^+ \cup V_2^-$, or all edges correspond to bottom-top paths and all its vertices are in the set $V_1^- \cup V_2^+$.

Case 1. All edges represent top-bottom paths, and therefore $\{v_1, \dots, v_k\} \subseteq V_1^+ \cup V_2^-$. Figure 4a shows an example. Consider the profile induced by all paths corresponding to the edges of this cycle. Edges incident to some vertex $v_i \in V_1^+$, with $i \in \{1, \dots, k\}$, correspond to paths with the same top vertex. Lemma 4 implies that these paths also share the same maximum prefix of some length $x \in [0, M]$. Therefore, by Lemma 1, every edge incident to v_i corresponds to some SoC function whose first breakpoint has the x-coordinate x. Thus, the leftmost point of the characteristic segment of each of these SoC functions lies on a vertical line defined by x; see Figure 4b. Similarly, edges incident to a bottom vertex $v_i \in V_1^-$ represent paths with the same maximum suffix of length $y \in [0, M]$. The last breakpoint of each SoC function associated with these paths lies on a horizontal line defined by the y-coordinate y. Hence, each of the k vertices defines either a vertical or a horizontal line. Every edge in the cycle C corresponds to a characteristic segment that starts at a vertical line and ends at a horizontal line, as shown in Figure 4b.

For a constant y inducing a horizontal line, we consider the leftmost x-coordinate of any breakpoint in an SoC function (corresponding to an edge in the cycle C) with the y-coordinate y; see Figure 4b. In total, we defined one x-coordinate for each vertex in C, denoted by x_i for $i \in \{1, \dots, k\}$. Without loss of generality, assume $x_1 < x_2 < \cdots < x_k$. Then, we obtain $k - 1$ intervals $[x_i, x_{i+1}]$, $i \in \{1, \dots, k - 1\}$. By assumption, every edge of C corresponds to a contributing path. Moreover, the characteristic segment of the SoC function of each contributing path is (partially) contained in the upper envelope of the SoC functions of all these paths (otherwise it would not contribute to the s–t profile). Given that all characteristic segments are parallel (with slope 1), this implies that each segment is the unique maximum over all characteristic segments on some interval $[x, x_{i+1}]$, $i \in \{1, \dots, k-1\}$. However, there are only $k - 1$ such intervals for k contributing paths; a contradiction.

Case 2. All edges represent bottom-top paths, and therefore $\{v_1, \dots, v_k\} \subseteq V_1^- \cup V_2^+$. In this case, edges incident to a bottom vertex $v_i \in V_1^-$ for an $i \in \{1, \dots, k\}$ correspond to paths with the same bottom vertex. By Lemma 4, these paths share the same minimum prefix with length $y \in [0, M]$. Moreover, observe that a contributing bottom-top path contains no subpath with cost below $-M$, since the cost of its maximum prefix must not exceed M. It follows that SoC functions of contributing bottom-top paths are of the form as in Case 3 of Lemma 1. Thus, the leftmost point of the characteristic segment of each SoC function represented by an edge incident to a bottom vertex $v_i \in V_1^-$ lies on the horizontal line defined by y. Similarly, edges incident to top vertices $v_i \in V_1^+$ correspond to characteristic segments whose rightmost point lies on the same vertical line defined by a constant $x \in [0, M]$. Along the lines of the first case, this yields a contradiction. ◀

4 Energy-Optimal Routes with Charging Stops

As battery capacities of EVs are typically rather small, *recharging* en route can be inevitable on long-distance trips. Therefore, we extend our model to incorporate charging stops. A subset $S \subseteq V$ of the vertices represents designated charging stations. Every station $v \in S$ has a predefined *SoC range* $R_v = [b_v^{\min}, b_v^{\max}] \subseteq [0, M]$. When arriving at v with *arrival SoC* b, we pick a desired *departure SoC* $b' \in [b_v^{\min}, b_v^{\max}] \cup \{b\}$ with $b \le b'$. SoC ranges are useful to model user preferences or technical features of charging stations. For example, we set $R_v := [M, M]$ for battery swapping stations. It is always allowed to pick the arrival SoC b as departure SoC, to account for the possibility of not charging at v.

Given a source $s \in V$, a target $t \in V$, and the initial SoC $b_s \in [0, M]$ at s, the *Energy-Optimal Route with Charging Stops (EORCS)* problem asks for a feasible path that minimizes *overall* energy consumption, defined as the difference $b_s - b_t$ between SoC at source and target, plus the total amount r_t of energy recharged at charging stations $v \in S$ to reach t. Hence, our objective is to *maximize* $b_t - r_t$ among all feasible solutions.

There are certain challenging aspects to this problem. First, we have to determine the departure SoC when reaching a charging station with some arrival SoC. Note that it may be wasteful to fully recharge the battery, as this may prevent recuperation on subsequent road segments. Second, subpaths of optimal paths are not optimal in general (e. g., detours to a charging station may be necessary). Hence, "greedy" choices can lead to suboptimal results. A natural way to overcome this issue is the use of label *sets* to keep multiple solutions at vertices, as in (exponential-time) multi-criteria search [23, 34]. In fact, this algorithm can be adapted to our problem setting, using labels with *continuous* ranges of SoC and recharged energy to reflect different choices at charging stations and the resulting SoC at a vertex. Then, vertices maintain labels that store an *SoC range* and a *charging range*. As in the multi-criteria scenario, we can apply Pareto dominance to remove suboptimal labels. However, it is not obvious whether such an approach has subexponential running time. Instead, we build upon tools from Section 3 to derive an alternative algorithm that maintains *single* labels on an extended search graph and that is conceptually simpler, can easily be integrated with known speedup techniques, and runs in polynomial time.

Optimal Paths between Charging Stations. When charging at a station $u \in S$, we have to ensure that the SoC is sufficient to reach t or the next charging station $v \in S$. Therefore, we examine an important subproblem, where given a charging station $u \in S$, an (optimal) arrival SoC b_u^{arr}, the amount r_u of energy recharged so far (at *previous* charging stations), and a vertex $v \in S \cup \{t\}$, we want to find a departure SoC $b_u^{\text{dep}} > b_u^{\text{arr}}$ that maximizes the objective at the target vertex t under the assumption that v is the next vertex where energy is recharged (or $v = t$ is the target itself). If we compute the u–v profile $f_{u,v}$, we can greedily optimize the objective on the s–v path by picking an SoC $b_u^{\text{dep}} > b_u^{\text{arr}}$ that maximizes $f(b_u^{\text{dep}}) - (r_u + r)$, where $r := b_u^{\text{dep}} - b_u^{\text{arr}}$ is the amount of energy charged at u. Unfortunately, the s–v path that maximizes this objective does not extend to the best solution at t in general. The reason for this is that charging too much energy might prevent the vehicle from recuperating energy on the following v–t path; see Figure 5 for an example.

Instead, we need a more sophisticated approach. To this end, we identify SoC values b_u^{dep} that may possibly lead to an optimal solution. We know (by the FIFO property [18]) that for an arbitrary departure SoC $b_u^{\text{dep}} \in [0, M]$, the optimal u–v subpath is also *energy-optimal* for b_u^{dep}. By Theorem 6, there can be at most $\mathcal{O}(|V|)$ such u–v paths. For each u–v path P contributing to the u–v profile $f_{u,v}$, we identify a (unique) *canonical* departure SoC b_P^{dep} at u that *always* optimizes the objective at t under the assumption that recharging is necessary at v (or $v = t$). Consider the SoC function f_P of P and let $b_P^{\min} := c(P_u^+)$ denote the minimum SoC that is necessary to traverse P, i. e., $f_P(b) = -\infty$ if and only if $b < b_P^{\min}$. Consequently, we have $b_P^{\text{dep}} \geq b_P^{\min}$. We also know that the objective $f_P(b_P^{\text{dep}}) - b_P^{\text{dep}} + b_u^{\text{arr}} - r_u$ of the s–v path can only *decrease* for $b_P^{\text{dep}} > b_P^{\min}$, since $b_u^{\text{arr}} - r_u$ is constant and the slope of f_P is at most 1 in the interval $[b_P^{\min}, M]$. Assuming that we recharge energy at v anyway, charging more than $b_P^{\min}$ will also never turn out to be essential after visiting v: If necessary, we can recharge missing energy at v. Therefore, given the SoC range $[b_u^{\min}, b_u^{\max}]$ of u, we pick the canonical departure SoC $b_P^{\text{dep}} := \max\{b_P^{\min}, b_u^{\min}\}$ for P, if this value lies in the SoC range of u. Otherwise, we have $b_u^{\max} < b_P^{\min}$ and charging at u never renders P feasible.

Search Graph Construction. Given the original graph G and the target vertex $t \in V$, we augment G with a *charging station (sub-)graph* $G_c = (V_c, E_c)$, which keeps separate vertices for distinct values of departure SoC at charging stations; see Figure 5. For each vertex $u \in S$, we create one *charging vertex* u' per distinct canonical departure SoC b_P^{dep} of *any* contributing path P from u to another charging station or to the target. We explicitly store the corresponding departure SoC b_P^{dep} with the vertex u', i. e., we keep a mapping

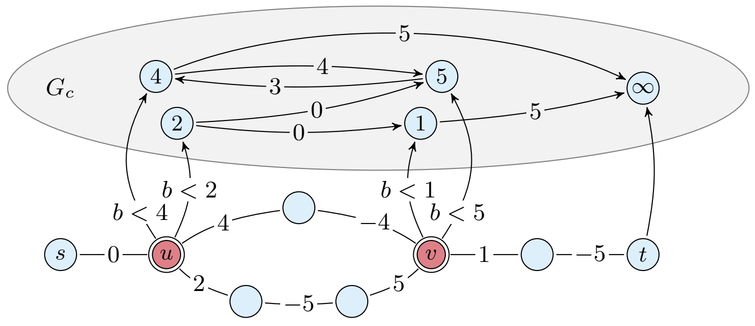


Figure 5 Search graph with charging stations (red, charging range $[0, M]$, $M = 5$). Vertex labels in G_c (shaded) indicate departure SoC. Edge labels indicate costs in G, arrival SoC in G_c, and SoC restrictions for transfer edges. While t is always reached from s with $b_t = 5$, the optimal path (and the objective $b_t - r_t$) depends on b_s: For $b_s < 2$, energy is charged at u ($b_u^{\text{dep}} = 2$) and v ($b_v^{\text{dep}} = 1$), which yields $r_t = 3 - b_s > 1$; for $b_s \in [2, 3]$ it is optimal to charge only at v ($b_v^{\text{dep}} = 1$) to get $r_t = 1$; for $b_s \in [3, 4)$ energy is only charged at v ($b_u^{\text{dep}} = 4$) to get $r_t = 4 - b_s \leq 1$; no charging is necessary at all for $b_s \geq 4$. In all cases the objective at v is maximized for $b_u^{\text{dep}} = 4$, which yields $b_v - r_v = b_u$.

$b^{\text{dep}}\colon V_c \to [0, M]$ and set $b^{\text{dep}}(u') := b_P^{\text{dep}}$. The vertex u' itself is added to V_c. We also add a dummy target vertex t' to V_c with $b^{\text{dep}}(t') := \infty$. For every contributing u–v path between vertices $u \in S$ and $v \in S \cup \{t\}$, we add edges (u', v') from the (unique) charging vertex $u' \in V_c$ of u with $b^{\text{dep}}(u') = b_P^{\text{dep}}$ to every corresponding vertex $v' \in V_c$ of v with $f_P(b_P^{\text{dep}}) < b^{\text{dep}}(v')$ to E_c. Together with the edge (u', v'), we also store the SoC upon arrival at v', i. e., we use a mapping $b^{\text{arr}}\colon E_c \to [0, M]$ and set $b^{\text{arr}}(u', v') := f_P(b_P^{\text{dep}})$. To connect G and G_c, we add (directed) *transfer edges* (v, v') from each charging station $v \in S \cup \{t\}$ to all its corresponding departure vertices $v' \in V_c$. Transfer edges have no cost, but may only be traversed if the current SoC is below the departure SoC $b^{\text{dep}}(v')$ of the respective departure vertex v'. Given the set E' of transfer edges, we obtain the *augmented graph* $G' := (V \cup V_c, E \cup E' \cup E_c)$. Note that its size is polynomial in the size of G.

A Polynomial-Time Algorithm. Using the augmented graph, we adapt EVD to find energy-optimal routes in G'. The modified algorithm maintains a single label $\ell(v)$ per vertex $v \in V \cup V_c$, which stores the values of SoC b_v and recharged energy r_v that maximize the objective $b_v - r_v$ at v. Initially, it sets $b_v = -\infty$ and $r_v = 0$ for all $v \in V$, except for the label $\ell(s) = (b_s, 0)$ of s. In each iteration of the main loop, the label $\ell(u) = (b_u, r_u)$ of some vertex u in G' with maximum key $b_u - r_u$ is extracted from the queue. If u is an original vertex, i. e., $u \in V$, the algorithm proceeds exactly like plain EVD by scanning its outgoing edges. If, additionally, u is a charging station, i. e., $u \in S$, its corresponding charging vertices $u' \in V_c$ are updated (and inserted into the queue) if $b_u < b^{\text{dep}}(u')$ and $\ell(u)$ yields an improvement to the label $\ell(u')$. Vertices $u \in V_c$ in the charging station graph are handled separately by the algorithm. For each outgoing edge (u, v) in G_c, a new label (b, r) is generated as follows. Its SoC is set to the arrival SoC $b := b^{\text{arr}}(u, v)$ at v and the amount of charged energy is set to $r := r_u + b^{\text{dep}}(u) - b_u$. If the resulting label (b, r) improves $\ell(v)$, the latter is updated accordingly. After termination, the label at the dummy target vertex t' (i. e., the unique vertex $t' \in V_c$ with $b^{\text{dep}}(t') = \infty$) contains the optimal pair of SoC and recharged energy. Making use of potential shifting, Theorem 7 follows directly from the polynomial size of G'.

▶ **Theorem 7.** *The problem EORCS can be solved in polynomial time.*

A Practical Variant. The construction of G_c is rather time-consuming on realistic instances. Luckily, we can move most work to preprocessing, since paths between charging stations are independent of source and target. We also propose a much simpler search graph, which can naturally be combined with CH for further speedup. We replace the graph G_c with an overlay graph $G_S = (S \cup \{t\}, S \times S \cup \{(v,t) \mid v \in S\})$. Every edge (u,v) in G_S stores as its cost function the u–v profile (wrt. the original graph). Compared to G_c, this significantly reduces the number of vertices. Moreover, it is straightforward to construct G_S using profile search. We slightly modify the search algorithm to work with G_S instead of G_c: If a scanned vertex $u \in S$ represents a charging station, all shortcuts (u,v) in G_S are scanned. For each, the arrival SoC b that maximizes the objective at v is picked (such that $b \geq b_u$ and $b \in [b_u^{\min}, b_u^{\max}]$). If this yields an improvement to the label of v, it is updated accordingly.

As argued before, picking the SoC at v in this greedy fashion may lead to suboptimal results (recall Figure 5). On real-world networks, however, this is very unlikely to occur, as it requires an optimal route with two charging stops u and v, such that t can be reached from u via v, but *not* directly, and at the same charging too much energy at u (to reach v on an optimal s–v path) prevents recuperation along the v–t path due to a fully-charged battery. Consequently, our approach *always* produced optimal solutions in our tests; see Section 5.

Integration with CH. In its basic variant, CH [20] iteratively *contract* vertices in increasing order of (heuristic) importance during preprocessing, maintaining distances between all remaining vertices by adding *shortcut* edges, if necessary. *Witness searches* determine whether a shortcut is required to preserve distances. The CH query runs bidirectional from source and target on the input graph augmented by all shortcuts added during preprocessing, following only *upward* edges (from less important to more important vertices).

When solving EORCS with CH, we do not contract charging stations during preprocessing [7, 41]. Hence, we stop vertex contraction at some point, leaving an uncontracted *core* of charging stations (and possibly other vertices). We run profile searches on this (relatively small) core graph to quickly construct the overlay G_S. Shortcuts are only added to G_S if their corresponding SoC function is *finite* for some SoC.

In a basic approach, witness search uses profile search to determine whether a shortcut is necessary. For faster preprocessing, an alternative approach replaces profiles by scalar *upper bounds* $\max_{b \in [0,M]}(b - f(b))$ on the energy consumption of an edge with SoC function f. Observe that negative costs are ruled out this way, since consumption must be at least zero for a fully charged battery. This re-enables Dijkstra's algorithm for witness searches, computing upper bounds $a \in [0, M]$ on energy consumption between a given pair of vertices. A shortcut candidate is inserted only if its SoC function f consumes less energy for at least one SoC, i. e., $b - f(b) < a$ for some $b \in [0, M]$. When using upper bounds, we may end up inserting unnecessary shortcuts. This does not affect correctness, but may (slightly) slow down queries. (Similarly, Eisner et al. [18] use a sampling approach to avoid costly profile search during preprocessing in their implementation.)

To obtain the full path description, we enable path unpacking by storing via vertices during contraction, as in plain CH [20]. (Note, however, that we need one via vertex per contributing path of an SoC function.) Additionally, we have to reconstruct paths representing shortcuts between charging stations within the core. This can be done by precomputing and storing the paths explicitly (in the core), or by running a profile search between each consecutive pair of charging stations in the optimal path. Finally, the retrieved paths in the core are unpacked. The optimal amount of energy that must be recharged is easily obtained from the SoC profiles in the overlay G_S (by picking a departure SoC for each profile that maximizes the objective).

The query algorithm consists of two phases. The first runs a *backward* profile search from t, scanning only upward and edges in the core. Shortcuts in G_S are ignored by this search. After its termination, SoC profiles from each charging station to the target are known. We (temporarily) add the target and all corresponding shortcuts to G_S. The second phase runs modified EVD from the source s with initial SoC b_s, on a graph consisting of upward edges and all edges in the core (including G_S).

Adding A* Search. On large instances, scanning shortcuts in the dense subgraph G_S becomes the major bottleneck of the approach. We add A* search [24] to improve performance. The basic idea of A* search is to compute a *consistent potential function* $\pi \colon V \to \mathbb{R}$ on the vertices, which fulfills the condition $c(u, v) - \pi(u) + \pi(v) \geq 0$ for all edges $(v, v) \in E$. The potential of a vertex is added to the key of a label when updating the priority queue. To make the search goal directed, we compute a consistent potential function where vertices that are closer to the target have smaller keys.

Before the CH search, we run a label-correcting backward search from t, scanning upward edges and core edges (except for shortcuts in G_S), with (scalar) *lower bounds* on energy consumption as edge costs. This yields, for each vertex v in the core, a lower bound $\underline{c}(v)$ on energy consumption from v to t. Moreover, $\underline{c}(\cdot)$ induces a consistent potential function on V, as follows immediately from the triangle inequality. The forward search is then split into two phases. The first scans upward edges, but ignores outgoing edges of core vertices. The second phase is initialized with all core vertices scanned during the first phase. Using the potential function, this phase becomes *goal directed*.

An aggressive variant of A* search achieves further speedup at the cost of suboptimal results. As before, when a charging station is visited by the forward search, all outgoing shortcuts (u, v) in G_S are scanned. However, we update the label of at most *one* vertex $v \in S$ and insert it into priority queue, namely, the one with maximum key among all vertices that are improved by the scans.

5 Experiments

We implemented all approaches in C++, using g++ 4.8.5 (-O3) as compiler. Experiments were conducted on a single core of a 4-core Intel Xeon E5-1630v3 clocked at 3.7 GHz, with 128 GiB of DDR4-2133 RAM, 10 MiB of L3, and 256 KiB of L2 cache.

Input and Methodology. We ran experiments on a graph representing the road network of Western Europe, kindly provided by PTV AG (`http://ptvgroup.com`). Energy consumption data stems from PHEM, a detailed micro-scale emission model [25]. We consider two vehicle models. The first is based on a real production vehicle (Peugeot iOn) with a battery capacity of 16 kWh (corresponding to a range of roughly 100–150 km). The second is an artificial model with a capacity of 85 kWh (400–500 km range, similar to recent Tesla models). Data source mentioned above are proprietary, but enable us to test our algorithms on detailed and realistic input data. To accurately assign consumption values to road segments, we retrieved road slopes based on elevation data from the freely available SRTM dataset (`http://www2.jpl.nasa.gov/srtm`). Removing edges without reasonable energy consumption (e. g., due to large areas with missing elevation data), we obtain a graph with 22 198 628 vertices and 51 088 095 edges after extracting the largest strongly connected component from the remaining graph [9]. About 11.8 % and 15.2 % of these edges have negative cost for the Peugeot and the artificial model, respectively. We also conduct

Table 1 Performance of our approaches (Europe). The columns G_S, CH, A*, and agg. (aggressive A*) indicate whether a technique is enabled (●) or not (○). For each approach and model, we report preprocessing time, number of vertex scans during queries (# V. Sc.), and query times.

Techniques				Peugeot, 16 kWh			Artificial, 85 kWh		
G_S	CH	A*	agg.	Prepr. [s]	# V. Sc.	Q. [ms]	Prepr. [s]	# V. sc.	Q. [ms]
○	○	○	○	–	8 895 038	20 160.9	–	11 033 760	32 928.8
●	○	○	○	1 487	759 951	710.0	15 062	7 753 601	6 285.7
●	●	○	○	2 860	8 433	309.6	3 246	19 616	1 281.5
●	●	●	○	2 860	3 563	128.2	3 246	10 418	297.5
●	●	●	●	2 860	1 599	41.0	3 246	9 579	157.8

Table 2 Performance for different distributions and types of charging stations (Germany, Peugeot). Besides timings for preprocessing and queries, we report the number of charging stations ($|S|$), edges in G_S ($|E_S|$), and vertex scans (# V. Sc.) and edge scans (# E. Sc.) during queries.

		Prepr.		Queries		
Scenario	$\|S\|$	T. [s]	$\|E_S\|$	# V. Sc.	# E. Sc.	T. [ms]
reg-cm	1 966	548.5	539 145	4 592	125 535	4.22
mix-cm	1 966	548.1	539 145	4 592	125 381	4.19
reg-r0.01	469	487.2	22 231	2 234	50 070	1.30
reg-r0.1	4 692	582.7	2 263 310	8 904	223 779	7.97
reg-1.0	46 920	965.0	227 514 459	60 527	1 828 581	73.46

experiments on the subnetwork of Germany, which has 4 692 091 vertices and 10 805 429 edges. We located 13 810 charging stations (1 966 of them in Germany) on ChargeMap (`chargemap.com`). Unless mentioned otherwise, all reported query times are average values of 1 000 queries, with source and target vertices picked uniformly at random. Charging stations have the SoC range $[0, M]$ and the initial SoC is set to $b_s = M$.

Evaluation. Table 1 compares different approaches to solve EORCS on our main test instance (Europe) for both vehicle models. Applied techniques are indicated by the four leftmost columns. The first line (no speedup technique enabled) shows our exact baseline approach, which is based on bi-criteria search. It requires no preprocessing, but takes 20–30 seconds to answer queries, which is rather impractical. Simply using the charging station graph already reduces query times greatly. However, scalability of this approach is limited, as increasing the vehicle range affects both preprocessing (longer paths between charging stations must be computed) and queries (the search in the uncontracted network dominates running times). Integrating CH clearly pays off, as it significantly reduces the number of vertex scans and query time after moderate preprocessing effort (below an hour). Query time is dominated by the search in G_S. A* search helps to reduce effort spent searching in G_S and makes our approach rather practical, with running times of less than 300 ms for the artificial model. Moreover, note that even though we use a (formally) inexact implementation, the optimal solution is found in *all* queries. The aggressive variant of A* further reduces query times at the cost of inexact results (even in practice). The average relative error (not reported in the table) is 0.7 % for the Peugeot model and less than 0.01 % for the artificial one. This discrepancy in relative error can be explained by the fact that a larger battery allows the EV

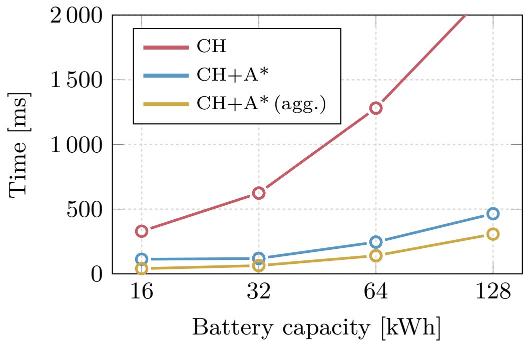


Figure 6 Algorithm performance subject to cruising range (Europe, Peugeot). Each point is the median running time of 1 000 queries for one of the different approaches (CH, CH with A*, CH with aggressive A*) under varying battery capacities.

to stick to energy-optimal paths (fewer detours are necessary), so the quality of the bounds used in A* search increases. Consequently, outliers for the Peugeot scenario exceed 10 % in relative error in about 1 % of the cases, while even the maximum error is below 0.5 % for the artificial model. For all techniques, the artificial model is harder to solve. This is mostly due to the dense charging station graph (more labels per vertex for the baseline approach), since more charging stations are reachable from each station.

In Table 2, we evaluate the performance of our fastest empirically exact approach (CH with A* search) under varying types and distributions of charging stations. The first scenario (reg-cm) uses stations from ChargeMap with (default) SoC range $[0, M]$. The second (mix-cm) uses the same stations, but assigns each a charging range of a regular station ($[0, M]$), a "super charger" that quickly charges to 80 % SoC ($[0, 0.8M]$), or a swapping station ($[M, M]$), with equal probability. The results indicate that SoC ranges have little impact on performance. This is not surprising, since restricting the departure SoC can only reduce the search space (the effect is negligible, though). Finally, we consider random distributions of charging stations with default SoC ranges (reg-r0.01, reg-r0.1, reg-1.0), where we pick 0.01 %, 0.1 %, and 1.0 % of the vertices in V as charging stations uniformly at random, respectively. The number of charging stations has a significant impact on algorithm performance. Given that the number of edges in G_S grows quadratically in the number of charging stations, preprocessing and query slow down for very dense networks of charging stations. This limits scalability, but our approach easily handles realistic distributions of charging stations (for the scenario reg-1.0, the number of charging stations is higher than the current number of gas stations in Germany).

Finally, Figure 6 shows running times of our algorithms for different battery capacities. Without A* search, running time roughly doubles with battery capacity, because G_S becomes more dense (more reachable charging stations). Adding A* search, scalability with available cruising range improves, since potentials quickly guide the search towards the target.

6 Conclusion

We examined consumption profiles for EVs and proved that their complexity is at most linear in the graph size. We also investigated energy-optimal routes with charging stops and showed how profile search can be utilized to solve the problem in polynomial time. In a sense, we closed the gap between (efficiently solvable) energy-optimal routes [18, 36] and $\mathcal{NP}$-hard time-constrained variants with charging stops [7, 35] (which generalize the problem setting considered in this work). In particular, it is indeed the addition of a second optimization criterion (travel time) that makes the latter problems $\mathcal{NP}$-hard, rather than the incorporation of charging stations in combination with battery constraints. We also proposed a practical variant, which computes optimal results in well below a second on realistic, large-scale networks.

Interesting lines of future work include reducing the number of edges in the overlay of charging stations for better performance and scalability [13, 26, 39] or integration of Customizable CH [16] for faster preprocessing. Moreover, one could consider a *profile* variant of EORCS, i. e., ask for a consumption profile instead of a single path.

References

1 Andreas Artmeier, Julian Haselmayr, Martin Leucker, and Martin Sachenbacher. The Optimal Routing Problem in the Context of Battery-Powered Electric Vehicles. In *Proceedings of the 2nd CPAIOR Workshop on Constraint Reasoning and Optimization for Computational Sustainability (CROCS'10)*, 2010.

2 Andreas Artmeier, Julian Haselmayr, Martin Leucker, and Martin Sachenbacher. The Shortest Path Problem Revisited: Optimal Routing for Electric Vehicles. In *Proceedings of the 33rd Annual German Conference on Advances in Artificial Intelligence (KI'10)*, volume 6359 of *Lecture Notes in Computer Science*, pages 309–316. Springer, 2010.

3 Mikhail J. Atallah. Some Dynamic Computational Geometry Problems. *Computers & Mathematics with Applications*, 11(12):1171–1181, 1985.

4 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Route Planning in Transportation Networks*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016.

5 Gernot V. Batz and Peter Sanders. Time-Dependent Route Planning with Generalized Objective Functions. In *Proc. of the 20th Annual European Symp. on Algorithms (ESA'12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2012.

6 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining Hierarchical and Goal-Directed Speed-up Techniques for Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 15:2.3:1–2.3:31, 2010.

7 Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest Feasible Paths with Charging Stops for Battery Electric Vehicles. In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'15)*, pages 44:1–44:10. ACM, 2015.

8 Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner. Speed-Consumption Tradeoff for Electric Vehicle Route Planning. In *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'14)*, volume 42 of *OpenAccess Series in Informatics (OASIcs)*, pages 138–151. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.

9 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Energy-Optimal Routes for Electric Vehicles. In *Proc. of the 21st ACM SIGSPATIAL Int'l Conference on Advances in Geographic Information Systems (GIS'13)*, pages 54–63. ACM, 2013.

10 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Dynamic Time-Dependent Route Planning in Road Networks with User Preferences. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA'16)*, volume 9685 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2016.

11 Brian C. Dean. Shortest Paths in FIFO Time-Dependent Networks: Theory and Algorithms. Technical report, Massachusetts Institute of Technology, 2004.

12 Daniel Delling. Time-Dependent SHARC-Routing. *Algorithmica*, 60(1):60–94, 2011.

13 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable Route Planning in Road Networks. *Transportation Science*, 2015.

14 Daniel Delling and Dorothea Wagner. Landmark-Based Routing in Dynamic Graphs. In *Proceedings of the 6th Workshop on Experimental Algorithms (WEA'07)*, volume 4525 of *Lecture Notes in Computer Science*, pages 52–65. Springer, 2007.

15 Daniel Delling and Dorothea Wagner. *Time-Dependent Route Planning*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer, 2009.

16 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 21:1.5:1–1.5:49, 2016.

17 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

18 Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal Route Planning for Electric Vehicles in Large Networks. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI'11)*, pages 1108–1113. AAAI Press, 2011.

19 Luca Foschini, John Hershberger, and Subhash Suri. On the Complexity of Time-Dependent Shortest Paths. *Algorithmica*, 68(4):1075–1097, 2014.

20 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, 2012.

21 Andrew V. Goldberg and Chris Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 156–165. SIAM, 2005.

22 Michael T. Goodrich and Paweł Pszona. Two-Phase Bicriterion Search for Finding Fast and Efficient Electric Vehicle Routes. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'14)*, pages 193–202. ACM, 2014.

23 Pierre Hansen. Bicriterion Path Problems. In *Multiple Criteria Decision Making – Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer, 1980.

24 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

25 Stefan Hausberger, Martin Rexeis, Michael Zallinger, and Raphael Luz. Emission Factors from the Model PHEM for the HBEFA Version 3. Technical report I-20/2009, University of Technology, Graz, 2009.

26 Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering Multilevel Overlay Graphs for Shortest-Path Queries. *ACM Journal of Experimental Algorithmics*, 13:2.5:1–2.5:26, 2009.

27 Gerhard Huber and Klaus Bogenberger. Long-Trip Optimization of Charging Strategies for Battery Electric Vehicles. *Transportation Research Record: Journal of the Transportation Research Board*, 2497:45–53, 2015.

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

29 Donald B. Johnson. Efficient Algorithms for Shortest Paths in Sparse Networks. *Journal of the ACM*, 24(1):1–13, 1977.

30 Sebastian Kluge, Claudia Sánta, Stefan Dangl, Stefan M. Wild, Martin Brokate, Konrad Reif, and Fritz Busch. On the Computation of the Energy-Optimal Route Dependent on the Traffic Load in Ingolstadt. *Transportation Research Part C: Emerging Technologies*, 36:97–115, 2013.

31 Yuichi Kobayashi, Noboru Kiyama, Hirokazu Aoshima, and Masamori Kashiyama. A Route Search Method for Electric Vehicles in Consideration of Range and Locations of Charging Stations. In *Proceedings of the 7th IEEE Intelligent Vehicles Symposium (IV'11)*, pages 920–925. IEEE, 2011.

32 Chung-Shou Liao, Shang-Hung Lu, and Zuo-Jun Max Shen. The Electric Vehicle Touring Problem. *Transportation Research Part B: Methodological*, 86:163–180, 2016.

33 Chensheng Liu, Jing Wu, and Chengnian Long. Joint Charging and Routing Optimization for Electric Vehicle Navigation Systems. In *Proceedings of the 19th International Federation of Automatic Control World Congress (IFAC'14)*, volume 47 of *IFAC Proceedings Volumes*, pages 9611–9616. Elsevier, 2014.

34 Ernesto Q. V. Martins. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

35 Sören Merting, Christian Schwan, and Martin Strehler. Routing of Electric Vehicles: Constrained Shortest Path Problems with Resource Recovering Nodes. In *Proceedings of the 15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15)*, volume 48 of *OpenAccess Series in Informatics (OASIcs)*, pages 29–41. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015.

36 Martin Sachenbacher, Martin Leucker, Andreas Artmeier, and Julian Haselmayr. Efficient Energy-Optimal Routing for Electric Vehicles. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI'11)*, pages 1402–1407. AAAI Press, 2011.

37 René Schönfelder and Martin Leucker. Abstract Routing Models and Abstractions in the Context of Vehicle Routing. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 2639–2645. AAAI Press, 2015.

38 René Schönfelder, Martin Leucker, and Sebastian Walther. Efficient Profile Routing for Electric Vehicles. In *Proceedings of the 1st International Conference on Internet of Vehicles (IOV'14)*, volume 8662 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2014.

39 Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Using Multi-Level Graphs for Timetable Information in Railway Systems. In *Proceedings of the 4th Workshop on Algorithm Engineering & Experiments (ALENEX'02)*, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer, 2002.

40 Olivia J. Smith, Natashia Boland, and Hamish Waterer. Solving Shortest Path Problems with a Weight Constraint and Replenishment Arcs. *Computers & Operations Research*, 39(5):964–984, 2012.

41 Sabine Storandt. Quick and Energy-Efficient Routes: Computing Constrained Shortest Paths for Electric Vehicles. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science (IWCTS'12)*, pages 20–25. ACM, 2012.

42 Sabine Storandt and Stefan Funke. Cruising with a Battery-Powered Vehicle and Not Getting Stranded. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, pages 1628–1634. AAAI Press, 2012.

43 Zhonghao Sun and Xingshe Zhou. To Save Money or to Save Time: Intelligent Routing Design for Plug-In Hybrid Electric Vehicle. *Transportation Research Part D: Transport and Environment*, 43:238–250, 2016.

44 Timothy M. Sweda, Irina S. Dolinskaya, and Diego Klabjan. Adaptive Routing and Recharging Policies for Electric Vehicles. Working paper no. 14-02, Northwestern University, Illinois, 2014.

45 Yan Wang, Jianmin Jiang, and Tingting Mu. Context-Aware and Energy-Driven Route Optimization for Fully Electric Vehicles via Crowdsourcing. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1331–1345, 2013.

46 Ady Wiernik and Micha Sharir. Planar Realizations of Nonlinear Davenport-Schinzel Sequences by Segments. *Discrete & Computational Geometry*, 3(1):15–47, 1988.