Speed-Consumption Tradeoff for Electric Vehicle Route Planning*

Moritz Baum[1], Julian Dibbelt[1], Lorenz Hübschle-Schneider[2], Thomas Pajor[3], and Dorothea Wagner[1]

1 **Department of Informatics, Karlsruhe Institute of Technology (KIT)**
76128 Karlsruhe, Germany
firstname.lastname@kit.edu

2 **Department of Computer Science, University of Leicester**
Leicester LE1 7RH, United Kingdom
lorenz@4z2.de

3 **Microsoft Research, Mountain View, CA 94043, USA**
tpajor@microsoft.com

Abstract

We study the problem of computing routes for electric vehicles (EVs) in road networks. Since their battery capacity is limited, and consumed energy per distance increases with velocity, driving the fastest route is often not desirable and may even be infeasible. On the other hand, the energy-optimal route may be too conservative in that it contains unnecessary detours or simply takes too long. In this work, we propose to use multicriteria optimization to obtain Pareto sets of routes that trade energy consumption for speed. In particular, we exploit the fact that the same road segment can be driven at different speeds within reasonable intervals. As a result, we are able to provide routes with low energy consumption that still follow major roads, such as freeways. Unfortunately, the size of the resulting Pareto sets can be too large to be practical. We therefore also propose several nontrivial techniques that can be applied on-line at query time in order to speed up computation and filter insignificant solutions from the Pareto sets. Our extensive experimental study, which uses a real-world energy consumption model, reveals that we are able to compute diverse sets of alternative routes on continental networks that closely resemble the exact Pareto set in just under a second—several orders of magnitude faster than the exhaustive algorithm.

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

Keywords and phrases electric vehicles, shortest paths, route planning, bicriteria optimization, algorithm engineering

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.138

1 Introduction

Personal electromobility has gained substantial momentum in recent years, which demands for novel route planning algorithms, considering factors such as speed and terrain. Although the past decade has seen a great amount of research conducted in the area of route planning in general, most of it shares one trait, though, and that is a focus on conventional vehicles

* Support by DFG grant WA 654/16-1, by the EU FP7/2007-2013 under grant agreement no. 609026 (project MOVESMART), and by the Federal Ministry of Economics and Technology under grant no. 01ME12013 (project iZeus). This work was done while the third and fourth authors were at Karlsruhe Institute of Technology. It is based on a Bachelor thesis of the third author [18].

14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'14).
Editors: Stefan Funke and Matúš Mihalák; pp. 138–151

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

using internal combustion engines. With electric vehicles, however, new factors become important that must be considered when planning routes: Battery capacity and thus cruising range are severely limited, while driving down-hill and breaking allow for recuperation of energy. Charging is a time-consuming process and therefore not viable en route. It turns out that traditional route planning techniques do not suffice, and new approaches are required.

A recent algorithmic survey for route planning in road networks is given by Bast et al. [2]. While many of the methods optimize a single criterion (typically travel time), some also extend to multiple criteria by utilizing multi-dimensional vertex labels that represent sets of *Pareto-optimal* paths [15, 21]. While theoretically hard [13], in some transportation networks this problem may actually be "feasible in practice" [22]. For general networks, the recent NAMOA* algorithm is an extension of A* search [16] to the multicriteria case [20], where vertex potentials help reducing the number of label scans. This approach was also applied to road networks [19] and later parallelized [24, 10]. For the case that the metric is a linear combination of two or more criteria, practical algorithms are available as well [14, 12].

For electric vehicles, most papers have focused on the integration of battery capacity constraints and negative edge weights (a result of recuperation) into classical single-criterion route planning algorithms optimizing energy consumption [9, 23, 4]. However, such routes may have disproportionate detours: driving slower saves energy at the cost of greatly longer travel time. Storandt [25] therefore optimizes energy consumption, but bounding the amount by which travel time may increase. Instead, we would like to present users a reasonably-sized *set* of routes that differently trade energy consumption and driving time, enabling them to adequately pick the one most suitable to them. Moreover, all known approaches assume a fixed driving speed per road segment, neglecting attractive solutions that still use major roads (such as freeways) but save energy by actually driving below the posted speed limits.

In this work, we compute comprehensive sets of routes that reasonably trade speed and energy consumption. Not only do we consider travel time and energy consumption as criteria, but also explore the possibility of driving the same road segment at different speeds (within reasonable bounds). Even though this extended scenario greatly increases query complexity, we demonstrate that it is practically possible to compute such routes for electric vehicles on large road networks. Applying several nontrivial improvements at query time, we reduce the (empirical) running time of our algorithm by several orders of magnitude (still computing full Pareto sets). Adding heuristic filtering techniques, we further reduce running times to 750 ms for continental road networks and a realistic electric vehicle model.

The paper is structured as follows. Section 2 provides necessary foundations. Section 3 describes our basic approach to compute the full set of Pareto-optimal paths. It also describes extensions that improve the algorithm's running time while retaining correctness. Section 4 introduces heuristic approaches, which aim to reduce the size of the Pareto sets by keeping only the most significant solutions. An experimental evaluation of all presented techniques is given in Section 5, while Section 6 concludes with final remarks.

2 Preliminaries

We consider *directed, weighted* (multi-)graphs $G = (V, E)$ where $E \subseteq V \times V$ is a *multiset* of *edges* (i. e., there is a mapping $m\colon E \to \mathbb{N}$ denoting the multiplicity of each edge). In other words, parallel edges are allowed. We call u the tail and v the head of an edge (u, v), and vertices are neighbors if they are connected by an edge. Moreover, a *weight function* $\omega\colon E \to \mathbb{Z}$ assigns weights to every edge in G. An *s–t-path* in G is a sequence $P_{s,t} = [s = v_1, v_2 \ldots, v_k = t]$ of vertices, such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$. If $s = t$, we call $P_{s,t}$ a *cycle*. A path is

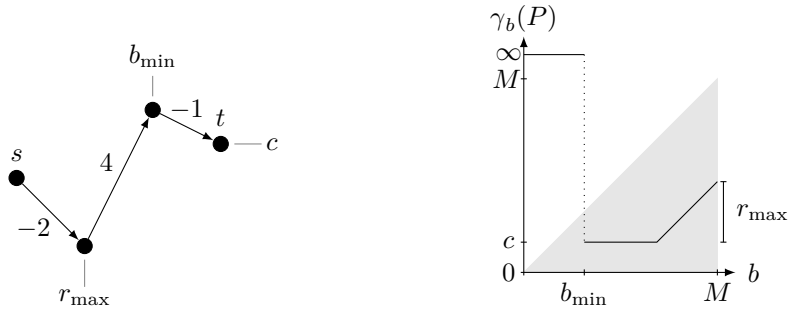

Figure 1 Left: An s–t-path P with edge weights -2, 4, -1. Right: The cost $\gamma_b(P)$ of traversing P subject to the battery's state of charge b at the source vertex s. A state of charge below $b_{\min}$ is not sufficient to reach t, i. e., $\gamma_b(P) = \infty$. In case b exceeds $M - |r_{\max}|$, the overcharging constraint limits recuperation on the first edge, which leads to a higher total cost $\gamma_b(P)$. This typical shape of the cost profile was first observed in Eisner et al. [9]. The cost of any (feasible, nonnegative) path is within the shaded area.

called *simple* if it contains no cycles. The weight (or cost) $\omega(P_{s,t}) = \sum_{i=1}^{k-1} \omega(v_i, v_{i+1}, i)$ of a path $P_{s,t}$ is the sum of its edge weights. A *potential function* $\phi\colon V \to \mathbb{R}$ on the vertices is called *feasible*, if $\omega(u,v) - \phi(u) + \phi(v) \geq 0$ for all $e \in E$. Any feasible potential induces a graph G' of nonnegative *reduced edge weights* by *shifting* the weight of every edge $e = (u,v)$, setting $\omega'(e) = \omega(u,v) - \phi(u) + \phi(v)$. This definition extends to paths canonically.

Dijkstra's algorithm [7] is a well-known approach to solve the shortest path problem on weighted graphs in (almost) linear time. It maintains (scalar) *distance labels* $d(\cdot)$ for each vertex, initially set to 0 for the source vertex s and ∞ otherwise. In each iteration, the algorithm extracts a vertex u with minimum $d(u)$ from a priority queue (initialized with s). It then *scans* all edges (u,v): if $d(u) + \omega(u,v)$ improves $d(v)$, it updates $d(v)$ accordingly and adds (or updates) v in the priority queue. If all edge weights are nonnegative, Dijkstra's algorithm has the *label-setting property*: Once a vertex v has been extracted from the queue, the distance label $d(v)$ is final and corresponds to the shortest path distance to v. The actual path can be retrieved by maintaining parent pointers during the algorithm.

In this work, we consider graphs representing road networks with two associated weight functions on the edges: *travel time* τ and *energy consumption* γ. Specific travel time and energy consumption values are denoted by x and y, respectively. We say that a tuple $d_1 = (x_1, y_1)$ *dominates* a tuple $d_2 = (x_2, y_2)$ if d_1 is smaller in both criteria than d_2 and strictly better in at least one. A set D of tuples is called a *Pareto set* if there are no two tuples $d_1, d_2 \in D$ such that d_1 dominates d_2. Similarly, a path $P_{s,t}$ is called *nondominated*, if no other path exists that dominates $P_{s,t}$ with respect to $\tau(P_{s,t})$ and $\gamma(P_{s,t})$. The *bicriteria shortest-path (BSP)* algorithm [15, 21] is a natural extension of Dijkstra's algorithm to the bicriteria setting. Instead of scalar values, the *label sets* $D(\cdot)$ of a vertex may hold an arbitrary number of *labels* (x,y). The algorithm starts with empty label sets, adding the label $(0,0)$ to $D(s)$. Then, it works along the lines of Dijkstra's algorithm, but propagating labels instead of label sets: In each step, it extracts the label d with smallest associated key from a priority queue, scanning all corresponding outgoing edges (u,v). For each, it generates a new label d' by adding the costs of (u,v) to d. If d' is not dominated by any label in $D(v)$, it adds d' to $D(v)$, removing any dominated labels (by d') in $D(v)$ on the fly. If edge weights of G are nonnegative, and the priority of labels in the queue is a linear combination of their costs, the algorithm is *label-setting*, that is, once a label has been extracted from the queue, it cannot be dominated anymore.

As mentioned before, recuperation of energy may lead to negative γ-values on some edges. (Note that cycles with negative weight are physically ruled out.) While in this setting the BSP algorithm is still correct, it loses its label-setting property. Also, since the battery has a limited capacity M (which cannot be exceeded), we must take additional *battery constraints* into account: for an s–t path $P_{s,t}$ to be *feasible*, the battery's *state of charge*, denoted b, has to remain within the interval $[0, M]$ at every vertex along $P_{s,t}$. Battery constraints can be satisfied by additional checks during the query with negligible overhead; see [9, 4]. Figure 1 shows how consumed energy along a path is influenced by battery constraints. For a study of different strategies to cope with negative edge weights in the context of electric vehicle routing, see Artmeier et al. [1]. They conclude that for metrics representing energy consumption, a label-correcting variant of Dijkstra's algorithm outperforms the Bellman-Ford algorithm [5] in practice (despite its exponential theoretical worst-case running time).

3 Problem Statement and Basic Approach

Traditional route planning algorithms compute routes on a model that assumes a fixed travel speed on each road segment of the network, usually reflected by the posted speed limit including (typical or historic) traffic conditions. Therefore, optimizing energy consumption in this model will likely result in unattractive routes that follow slow roads in order to save energy. On the other hand, energy could also be saved by following fast roads but driving *below* the speed limit: The onboard navigation system could instruct the user about the recommended speed in order to meet a certain total energy consumption goal.

In our model we define with each edge of the graph an *interval* of minimum and maximum speeds, given by the input. Thereby, we only consider a limited number of discrete speed values in the interval (typically in 10 kph steps) in order to make it easy for the driver to comply with them. Hence, given the road network, we create a *multigraph* $G = (V, E)$, in which each road segment of the input is added to E as many times (weighted with appropriate τ and γ values) as there are possible speeds to traverse it. Now, given vertices s and t, our goal is to compute the full Pareto set of all nondominated paths from s to t. Note that besides providing alternative routes, we can also use this Pareto set to derive *constrained* paths, such as the one with minimal travel time subject to energy consumption at most c (for some $c \in [0, M]$). In what follows, we present our basic algorithm for computing full Pareto sets, and then describe several improvements that help reducing the query time.

Basic Approach. To solve the problem, we can immediately use the BSP algorithm (cf. Section 2) on the multigraph G. Recall however, that because the graph may contain negative edge weights (due to recuperation), the algorithm is not label-setting. By these means we cannot use *target pruning*: a label that is dominated by the current (tentative) target label set may still belong to a Pareto-optimal path with a suffix containing negative consumption values. However, as negative cycles are ruled out, we can safely use a technique called *hopping reduction* [8]: after extracting a label (x, y) and before scanning an edge (u, v), we check whether v is the predecessor on the current path to u. If this is the case, traversing this edge provably cannot improve the label set at v. We can thus discard it.

Label-Setting Property. Next, we describe a way to obtain feasible vertex potential functions. These will help to make the algorithm label-setting, which, in turn, enables target pruning. While any feasible (cf. Section 2) potential for the energy consumption function γ would be sufficient to achieve our goal, we present a potential function that addition-

ally helps guide the search toward the target, similarly to A^*-search [16]. Building upon a technique by Tung and Chew [26], we compute the potential by running two Dijkstra queries prior to the BSP algorithm. Both queries work on the reverse graph (i. e., the input graph with all edges reversed), starting from the target vertex t. The first uses the (scalar) edge consumption values and computes labels $d_\gamma(\cdot)$ at all vertices. Note that this query is actually label-correcting. We prune the search whenever a label exceeds the battery capacity, however, for correctness we must ignore the overcharging constraint (thereby obtaining lower bounds on consumption). The second query uses travel times to compute labels $d_\tau(\cdot)$, and is restricted to those vertices in G that have been reached by the first query. Since both queries optimize a single criterion within a limited range around t, their running time is negligible compared to the subsequent BSP query.

Given the labels of both queries, we obtain at every vertex v potentials $\phi_\tau(v) = d_\tau(v)$ for travel time, and $\phi_\gamma(v) = d_\gamma(v)$ for energy consumption. Since the potentials constitute lower bounds on both costs from v to t, feasibility directly follows from the triangle inequality. We now make the BSP query label-setting by adjusting the key of labels (x, y) in the priority queue to be a linear combination of the *reduced costs* (of its corresponding path) according to our potentials ϕ_τ and ϕ_γ. The following Theorem 1 formally proves that this is indeed sufficient to make the algorithm label-setting.

▶ **Theorem 1.** *For a label (x, y) at vertex v, let the priority queue key be defined as $\lambda(x + \phi_\tau(v)) + \mu(y + \phi_\gamma(v))$ (with $\lambda, \mu \in \mathbb{R}^{\geq 0}$). Then the BSP algorithm is label-setting.*

Proof. Assume to the contrary, that a label (x, y) at a vertex v is dominated at some point after being extracted from the queue. Since (reduced) weights are positive, keys of subsequently extracted labels have increasing keys. Hence, the label is dominated after extracting a label (x', y') (at some neighbor u of v) with greater or equal key. Without loss of generality, let $\lambda = 1$, then we have

$$x' + \phi_\tau(u) + \mu(y' + \phi_\gamma(u)) \geq x + \phi_\tau(v) + \mu(y + \phi_\gamma(v)). \tag{1}$$

On the other hand, after scanning an outgoing edge (u, v), the label (x, y) is dominated, i. e., $x' + \tau(u, v) \leq x$ and $y' + \gamma(u, v) \leq y$ (and in at least one case, equality must not hold). However, due to feasibility of the potential, we know that $\tau(u, v) \geq \phi_\tau(u) - \phi_\tau(u)$ holds for travel time. Plugging this bound into the domination condition (and proceeding analogously for consumption) yields

$$x' + \phi_\tau(u) \leq x + \phi_\tau(v) \text{ and} \tag{2}$$
$$y' + \phi_\gamma(u) \leq y + \phi_\gamma(v). \tag{3}$$

However, demanding that inequality holds in Equations (2) or (3) immediately contradicts Equation (1). This completes the proof. ◀

Target Pruning. Potentials enable *target pruning* as follows. Whenever the algorithm is about to add a label (x, y) to a label set at vertex v, it first checks if the label $(x + \phi_\tau(v), y + \phi_\gamma(v))$ is dominated by any label of the target's label set $D(t)$. In this case, it discards (x, y): the distances $\phi_\tau(v) = d_\tau(v)$ and $\phi_\gamma(v) = d_\gamma(v)$ yield lower bounds on the cost of *any* path from v to t, hence, (x, y) cannot be part of the Pareto-optimal solution at t.

We can further exploit the two performed extra queries to strengthen our target pruning. As we already computed the fastest paths from reachable vertices to t (in the second query), we may quickly compute the energy consumption $y_{\max}$ of the fastest s–t path by traversing

its edges, applying battery constraints accordingly. We then add a "virtual" solution $(0, y_{\max})$ to $D(t)$, which is only used for target pruning in the subsequent BSP algorithm. (Note that no label dominated by $(0, y_{\max})$ can be part of a Pareto-optimal solution.) The same cannot be done for the energy-optimal path as easily, since the lower bounds from the first query are not tight; in fact, battery constraints may not only imply a different consumption along the path, but even render it infeasible.

Contracting Vertices with Two Neighbors. Adding parallel edges to the graph may greatly increase the number of Pareto optimal solutions. This becomes particularly evident for long sequences of vertices with parallel edges: Assume a chain of n vertices, each connected to its (at most) two neighbors by k edges. At every vertex u, the BSP algorithm scans k edges (u, v) for each label in the label set $D(u)$, each possibly creating a new label at v. Thus, in the worst case we get $\Theta(k^n)$ nondominated labels at the last vertex of the chain. Indeed, by these means the sizes of the Pareto sets depend on the level of detail present in the model of the road network, rather than its structure. Also, requiring users to frequently adjust their driving speed on long road segments is unreasonable. We therefore *contract* (some) of these vertices. More precisely, we remove v from the graph and for each pair of edges $e_1 = (u, v)$ and $e_2 = (v, w)$, we add (u, w) to E, iff the driving speeds of e_1 and e_2 coincide. By these means, the number of edges can only decrease. Note that we do not contract vertices that represent intersections, or at which the road category or speed limit changes (see Section 5).

Subgraph Extraction. We can improve locality of the BSP algorithm as follows. After running the two initial Dijkstra searches, we extract the (comparatively small) induced subgraph of the reachable vertices. More precisely, we run Dijkstra's algorithm from s, using a linear combination of the reduced weights for every edge (u, v), i. e., $\lambda(\tau(u, v) - \phi_\tau(u) + \phi_\tau(v)) + \mu(\gamma(u, v) - \phi_\gamma(u) + \phi_\gamma(v))$. Thereby, whenever we extract a vertex, it (and its incident edges) are immediately added to a search graph G'. Also, we prune at vertices at which the lower bound on consumption (induced by $\phi_\gamma(\cdot)$) exceeds that of the (previously computed) fastest route. As a result, the graph G' is small and its vertices are arranged in Dijkstra rank order. This greatly improves spatial locality of the subsequent BSP query, which we now run on G' instead of G.

4 Heuristic Improvements

All aforementioned techniques preserve the correctness of the algorithm, i. e., they compute full Pareto sets. However, label set sizes still grow quickly with distance from s (exponentially in the worst case). Therefore, computing the full Pareto set is prohibitive for long-range queries. In what follows, we present several heuristic techniques that aim to reduce label set sizes by discarding labels at certain points during the query. Though we drop exactness, our experimental evaluation (cf. Section 5) shows that they still provide high-quality solutions while greatly improving performance. We present three independent techniques in turn, which can be combined to improve running time further.

Early Aborting. When extracting a label d at some vertex u and scanning parallel edges with head vertex v, we abort the scan at the first (newly generated) label that is dominated by the label set at v, i. e., no more edges with head v are scanned for the label d. The intuition of this *early aborting* technique is that, locally, the tentative labels created by

parallel edges usually differ only slightly, and the effect on solution quality is little. Despite the simplicity of this method, its impact on running time is notable (see Section 5).

Relaxing Dominance. A common technique to heuristically reduce the size of large Pareto sets is to relax dominance. For an overview of common notions of relaxed dominance and an experimental comparison, see Batista et al. [3]. In this work, we consider *ε-dominance*: For given nonnegative values ε_τ and ε_γ, a label (x', y') ε-dominates a label (x, y) iff $(x' - \varepsilon_\tau, y' - \varepsilon_\gamma)$ dominates (x, y). Applying this rule, new labels (x, y) are added to an existing label set only if they are not ε-dominated by any of its labels. In other words, we add the label (x, y) only if it yields a *significant* improvement. The input parameters ε_τ and ε_γ control the amount by which dominance is relaxed.

Label-Discarding Techniques. The next strategy to reduce label set sizes periodically discards insignificant labels from the label sets simultaneously at all vertices of the graph. However, arbitrarily removing labels may lead to infinite loops in the algorithm. Hence, we first establish a sufficient condition that guarantees algorithm termination. We then present two discarding techniques, which obey the termination condition.

To see why removing labels can cause infinite loops, consider the following simple example. Take two adjacent vertices u and v with label sets $D(u) = \{(x, y)\}$ and $D(v) = \emptyset$. Scanning the label (x, y) will generate a new label (x', y') at $D(v)$. If the label (x, y) is cleared before (x', y') is scanned, the algorithm will reinsert a new label (x'', y'') into $D(u)$. Now, clearing $D(v)$ will exactly recreate the initial configuration, potentially causing an infinite loop in the algorithm.

We now show that we can remedy this issue and, more generally, always guarantee algorithm termination, if the lexicographically smallest label is kept in each label set during the discarding process.

▶ **Theorem 2.** *If the graph contains no negative edge weights, all cycles in the graph have positive weights, and, for each label set, the (lexicographically) smallest label is never discarded, the BSP algorithm terminates.*

Proof. We show that after a finite number of extractions, the queue of the BSP algorithm is empty, thus, implying algorithm termination. Recall that the queue operates on single labels, and each label corresponds to a distinct path in the graph. Also, since new labels are created by appending edges to existing paths, every distinct path in the graph (more precisely, the label representing it) is added to the queue at most once. To prove the claim, it suffices to show that the number of distinct inserted paths is finite.

If discarding is not applied, only labels representing *simple* paths are inserted into the queue; labels of paths containing cycles are always dominated by those representing the same path, but excluding all cycles. In this case termination follows immediately, since there is only a finite number of simple paths in the graph. With discarding applied, however, we can no longer guarantee that a label in the queue corresponds to a simple path, e. g., if the label representing its simple subpath has been removed previously.

However, we show that for every label, the (reduced) cost (of both τ and γ) of its path is bounded. Recall that the lexicographically smallest label is never discarded from a label set. Consider an arbitrary (nonempty) label set $D(v)$, and let $\underline{k(v)}$ denote the key of its smallest label. At some point (after a finite number of extractions), the minimum key in the queue exceeds $\underline{k(v)}$. This is because keys of inserted labels increase due to nonnegative reduced edge weights and strictly positive cycles, while $\underline{k(v)}$ can only decrease. Let $\underline{d(v)}$ denote the

smallest label in $D(v)$ at this point. This label can only be removed from the label set $D(v)$ if it is dominated. Hence, after a finite number of extractions, no labels representing s–v-paths dominated by $\underline{d(v)}$ are added to $D(v)$. Therefore, every label must have bounded costs.

It remains to show that the number of paths that are not dominated by $\underline{d(v)}$ is finite. This, however, is easy to see, since every path in the graph is composed of a simple path and an arbitrary numbers of cycles attached to it. As every cycle has strictly positive weight, only a finite number of cycles can be added to a simple path before it is dominated by $\underline{d(v)}$. ◀

Next, we present two approaches for periodically discarding labels. Both are applied every k iterations of the algorithm (where k is a tuning parameter), removing insignificant labels from all label sets that have been modified since the previous discarding procedure.

The first method attempts to identify clusters of labels, from which it deletes all but a small number of "representative" ones. We implement this method by using *DBSCAN* (Density-Based Spatial Clustering of Applications with Noise), a known approach for clustering [11]. In general it takes as input a set of points (of some metric space) and two parameters: a *threshold distance* ε and a *minimum neighborhood size* k. Initially, each point is its own cluster and marked unvisited. While there are unvisited points, the algorithm picks one and checks whether its number of neighbors (i. e., points at distance at most ε) is at least k. In this case, the algorithm joins the clusters of the point and its neighbors, recursing on each newly-added neighbor. In our implementation, we use the Euclidean distance according to (scaled) energy consumption and travel time as metric. For each cluster, we keep every i-th label, including the smallest and largest ones (wrt. lexicographic order; i being a tuning parameter). The running time of the algorithm is in $\mathcal{O}(n \log n)$ [11], and it requires dynamic data structures, such as a queue of unvisited neighbors when growing clusters. Also, labels are discarded from clusters based on lexicographic order, rather than a quality measure.

Next, we propose *delta discarding*, which aims at discarding labels based on relative quality measures. For a given label set it scans labels (x, y) in ascending order, comparing each with its (lexicographic) predecessor $(x_{\text{pre}}, y_{\text{pre}})$. It does so by evaluating the differences $\Delta_x = |x - x_{\text{pre}}|$ and $\Delta_y = |y - y_{\text{pre}}|$. If both are sufficiently small, we discard one label. Assume, without loss of generality, that $y < y_{\text{pre}}$ (hence, $x > x_{\text{pre}}$). To decide which label to discard, we consider the ratio Δ_c/Δ_t. If it is below a predefined threshold, i. e., little overhead in consumption achieves a high gain in travel time, we discard (x, y), as $(x_{\text{pre}}, y_{\text{pre}})$ provides the better tradeoff. Otherwise, we discard $(x_{\text{pre}}, y_{\text{pre}})$. Note that this algorithm sweeps over the label set only once (presuming it is sorted), with little computational overhead per label.

5 Experiments

We implemented all algorithms in C++, using clang++ 3.4.1 (flags `-O3`) as compiler, run on one core of a dual 8-core Intel Xeon E5-2670 processor clocked at 2.6 GHz with 64 GiB of DDR3-1600 RAM, 20 MiB of L3 and 256 KiB of L2 cache.

Input Data. Our experiments are based on road network data which is kindly provided by PTV AG for scientific use. Elevation information stems from the Shuttle Radar Topography Mission (SRTM) data version 4.1, freely available from the CGIAR Consortium for Spatial Information.[1] It covers large parts of the world with a resolution of three arc seconds ($\approx$ 90 meters at the equator). We delete areas from the graph for which elevation

[1] `http://srtm.csi.cgiar.org/`

Table 1 Evaluating our exact BSP algorithm. We use a battery capacity of 4 kWh on 100 random queries. The columns PE (parallel edges), A* (goal-directed search), TP (target pruning), and HR (hopping reduction) indicate whether the respective improvement is enabled (•) or not (○).

	improvements			subgraph extr.		BSP algorithm				
PE	A*	TP	HR	# vert.	time [ms]	# extr.	# comp.	# sol.	time [ms]	spdup
○	○	○	○	9524	206.7	277 k	16 M	18	151	—
•	○	○	○	9524	220.2	13 218 k	152 132 M	1 300	261 641	1.0
•	•	○	○	1921	221.9	2 558 k	5 310 M	1 300	12 648	20.7
•	•	•	○	1921	222.0	197 k	593 M	1 300	710	368.5
•	•	•	•	1921	222.1	197 k	593 M	1 300	700	373.8

information was missing in the data (removing large parts of Scandinavia). Also, we do not consider private roads and ferries, as we have no energy consumption values available for those. For all edges in the input graph, we define an interval of admissible driving speeds depending on the speed limit and the road type. We bound the minimum admissible speed such that traffic flow is still maintained and also by a threshold below which no more energy is saved. For example, we set the minimum speed on motorways to 90 kph. Within these speed intervals, we add parallel edges for every step of 10 kph. We then contract vertices with two neighbors subject to the following conditions. First, their number of incoming and outgoing edges needs to be identical. Second, all edges must share the same road category, with corresponding consumption values having the same sign (note that this is a necessary condition to retain correctness in the presence of battery constraints). From this graph we extract the largest strongly connected component for our experiments. It has 19,046,204 vertices and 66,297,320 edges (44,675,948 unique edges). About 11 % of the edges have a negative consumption value.

The energy consumption data originates from PHEM (Passenger Car and Heavy Duty Emission Model) [17], developed by the Graz University of Technology. PHEM is a microscale emission model based on backwards longitudinal dynamics simulations. Among others, it contains electric vehicle energy consumption values for a large variety of traffic situations, road categories, speed limits, and slopes. We carefully map these values to our network by measuring the similarity of road segments from the PTV data and the parameters of PHEM. The vehicle chosen for our experiments is a Peugeot iOn, for which highly detailed consumption data is available in PHEM. Its battery capacity is 16 kWh.

Evaluating the Exact Algorithm. The first experiment considers the performance of our exact BSP algorithm and its improvements from Section 3. Here, we use a smaller battery capacity of 4 kWh, since running times for 16 kWh are in the order of hours (for plain BSP). For each variant, we evaluate (the same) 100 queries with source and target vertices s, t selected uniformly at random. We pick s and t such that t is always reachable from s (otherwise, the first Dijkstra query during initialization would quickly determine that the target is unreachable). For all queries, we assume that the vehicle battery is fully charged (thereby maximizing range). Based on preliminary experiments, we solely use the energy consumption value of labels as key in the priority queue, which turned out as fastest.

Table 1 reports figures for several variants of BSP, each computing the full Pareto set. We indicate whether the following improvements are active (•) or not (○): goal-directed search (A*), target pruning (TP) and hopping reduction (HR). The table also reports the

average time to extract the subgraph and its size (# vert.), the average number of queue extractions (# extr.), the average number of label comparisons during BSP (# comp.), the average size of the Pareto set at t (# sol.), the running time, and the speedup (spdup). In addition, the first (PE disabled) row shows BSP for the case that no parallel edges are added to the graph, i. e., the driving speed on each road segment is fixed to the speed limit.

We observe that adding parallel edges greatly increases the number of nondominated solutions and, hence, query complexity. This justifies our approach: Many viable solutions are not captured when fixed speeds on the edges are presumed. Subgraph extraction takes around 220 ms on average, resulting in a search graph containing 1921 vertices (9524 vertices without A*, as no pruning is applied during subgraph extraction in this case). Making the algorithm label-setting (A* enabled), improves the average running time by an order of magnitude and greatly decreases the number of extracted vertices and label comparisons. Adding target pruning, further accelerates the algorithm by another order of magnitude. On the other hand, hopping reduction turns out to be of little benefit. Since our implementation quickly detects dominated labels (by maintaining sorted label sets), hopping reduction saves only few label comparisons at the additional cost of checking the label's parent pointer. Summarizing, we observe that already for such short-range queries (the battery charge is only 4 kWh), the exact Pareto sets contain over a thousand solutions on average, justifying the use of our heuristics.

Evaluating the Heuristics. This experiment uses a battery capacity of 16 kWh and evaluates the impact of the heuristics from Section 4. Before discussing performance, we define their parameters (a detailed evaluation of the parameters follows later) and explain how we evaluate the quality of the obtained solutions.

Regarding ε-dominance, we set $\varepsilon_\tau = 1.6\,\text{s}$ and $\varepsilon_\gamma = 4.0\,\text{Wh}$. Recall that this indicates the minimum cost by which two routes have to differ in order to be included in a solution. For DBSCAN, initial experiments showed that requiring two points within a neighborhood of 1000 units works well for most queries. A unit is either 1 mWh (energy consumption) or 0.4 ms (travel time). Recall that we use Euclidean distance according to these units as metric. For each cluster, we keep the (lexicographically) first and last label together with every fifth label of the cluster. Finally, for delta discarding, we set a difference of 1.0 s (time) and 3.0 Wh (energy consumption) as similarity criteria. We set the threshold that determines which labels to keep to 2.5 Wh/s (9 kW). Both discarding techniques are applied every 2^8 queue extractions (set to 2^{10} if discarding is combined with another heuristic).

Regarding solution quality, we use two measures. The first considers how well the solution of a heuristic *covers* the optimal paths (of the exact algorithm). The second measure evaluates the relative *error* in travel time and energy consumption for the Pareto set.

For coverage we compare the set of nondominated paths $\mathcal{P}_{\text{heu}}$ from a heuristic to the exact Pareto set $\mathcal{P}_{\text{opt}}$ from the the exhaustive BSP algorithm at t. We do so by first determining, for each path $P_i \in \mathcal{P}_{\text{heu}}$, its most similar path $P_i' \in \mathcal{P}_{\text{opt}}$ according to the weighted (by *geographical length* len) Sørensen-Dice index [6], which is defined for paths P_1, P_2 as $d(P_1, P_2) = 2\,\text{len}(P_1 \cap P_2)/(\text{len}\,P_1 + \text{len}\,P_2)$. The similarity of $\mathcal{P}_{\text{heu}}$ and $\mathcal{P}_{\text{opt}}$ is then the accumulated similarity of each previously matched pair P_i, P_i', that is, $d(\mathcal{P}_{\text{heu}}, \mathcal{P}_{\text{opt}}) = 2\sum_i \text{len}(P_i \cap P_i')/\sum_i(\text{len}(P_i) + \text{len}(P_i'))$. Note that $d \in [0, 1]$ (larger values are better).

To measure relative errors of Pareto sets, we use the *S-metric* [27]: Given a Pereto set $\mathcal{P}$, consider the rectangles R_i enclosed by each point $P_i \in \mathcal{P}$ and a fixed reference point P^*. We set $P^* = (t_{\max}, M)$, where $t_{\max}$ is the (maximum) travel time of the energy-optimal path and M the battery capacity. (Note that the rectangle enclosed by the points $(0, -M)$

Table 2 Evaluating our heuristics on 100 random queries with a battery capacity of 16 kWh. Columns EA (early aborting), DB (DBSCAN), Δ (delta discarding), εD (ε-dominance) indicate whether a technique is used (•) or not (○). We always enable all improvements from Table 1.

heuristics				query performance				solution quality	
EA	DB	Δ	εD	# extr.	# comp.	# sol.	time [ms]	cov. [%]	err. [%]
○	○	○	○	32 697 k	487 054 M	11 867	808 993	100.0	0.00
•	○	○	○	21 056 k	170 843 M	7 664	295 974	99.6	0.13
○	•	○	○	10 617 k	139 539 M	9 871	413 169	97.4	0.76
•	•	○	○	9 823 k	66 229 M	6 159	220 198	97.9	0.72
○	○	•	○	722 k	2 243 M	1 964	3 106	97.5	0.82
•	○	•	○	618 k	949 M	1 474	1 898	98.0	0.75
○	○	○	•	995 k	315 M	333	1 618	99.2	0.35
•	○	○	•	703 k	158 M	248	1 149	96.3	1.93
○	○	•	•	294 k	60 M	227	750	97.6	0.79
•	○	•	•	215 k	23 M	140	644	95.1	2.45

and P^* bounds the objective space.) Then, the S-metric $S(\mathcal{P})$ is defined as the area of $\bigcup_i R_i$, i. e., the size of the set of points dominated by $\mathcal{P}$, and the relative error of a Pareto set $\mathcal{P}_{\text{heu}}$ is $1 - S(\mathcal{P}_{\text{heu}})/S(\mathcal{P}_{\text{opt}})$. Note that relative errors are in $[0, 1]$, and smaller values are better.

Table 2 reports results on 100 random queries for a vehicle with a battery capacity of 16 kWh. Regarding query performance, the table reports the average number of queue extractions (# extr.), the average number of label comparisons (# comp.), the average number of solutions (# sol.), and the average running time (which includes initialization). Regarding quality, the table reports the average path coverage (cov.) and the relative error of the Pareto set (err.). Extracting the subgraph takes under 500 ms (marking 33 580 vertices on average). We see that early aborting yields a speedup of more than two compared to the basic approach, with only little loss in quality. On the other hand, DBSCAN has similar running times, but with significantly lower quality in both dimensions. Delta discarding and ε-dominance yield even faster queries, achieving speedups by more than two orders of magnitude. However, ε-dominance computes solutions of higher quality with much smaller Pareto sets. Note that the error of suboptimal solutions for ε-dominance is actually bounded by ε. We also evaluate the combination of several heuristics. Early aborting improves running times of the discarding techniques by up to a factor 1.9 with negligible effect on solution quality. The fastest combination is delta discarding with ε-dominance, yielding query times of under 800 ms. Note that in this case, the initialization phase (Dijkstra runs and subgraph extraction) become the major bottleneck. All aforementioned combinations produce solutions of excellent quality, covering more than 97.5 % of all Pareto-optimal paths with an average relative error below 1 %. On the other hand, early aborting increases the error significantly (to up to 2.45 %), if it is combined with ε-dominance. We conclude, that using delta discarding with ε-dominance, we are able to provide solutions of very good quality in well under a second.

Finally, Figure 2 evaluates different parameters for the heuristics from Table 2. It plots the relative error of the solution subject to the running time of the algorithm. Labels at points of discarding techniques depict the (base-2) logarithm of the discarding frequency. Labels at points of ε-dominance depict a factor ε, such that $\varepsilon_\tau = \varepsilon \cdot 0.4\,\text{s}$ and $\varepsilon_\gamma = \varepsilon \cdot 1.0\,\text{kWh}$. The discarding frequency for combinations that include ε-dominance is always 2^{10}, as this

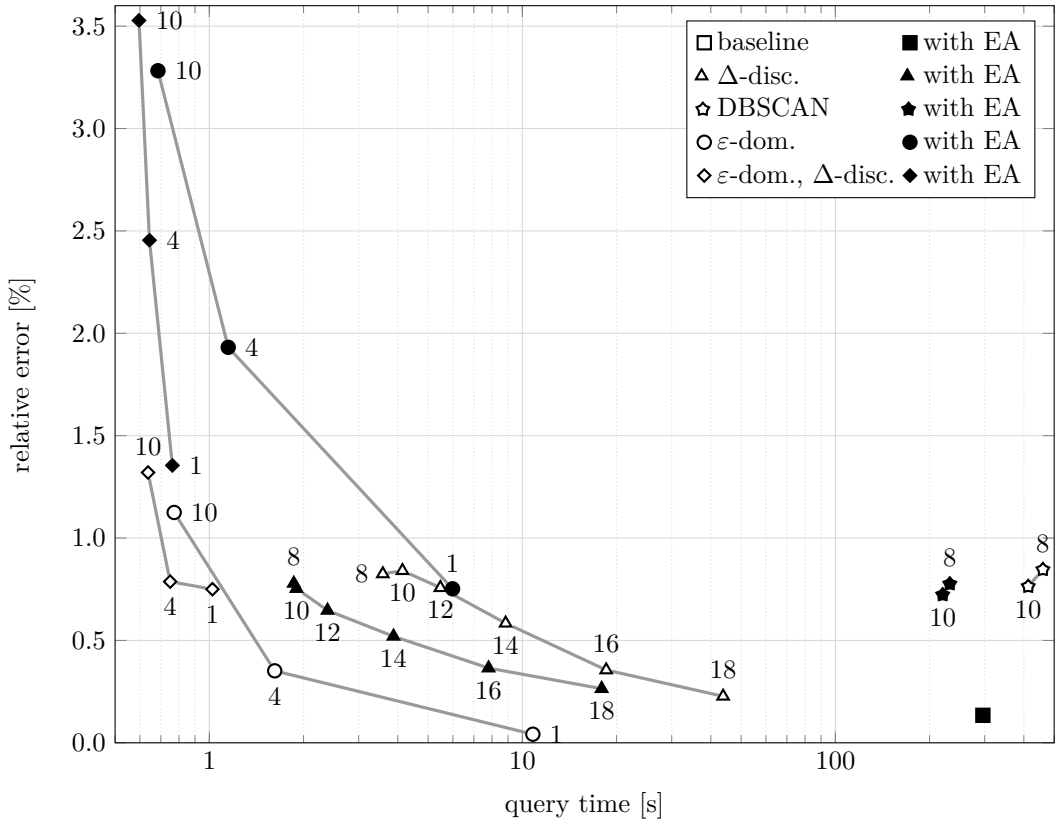

Figure 2 Error subject to query time for the heuristic approaches for different parameter choices, indicated by labels at points of the plot. They represent the (base-2) logarithm of discarding frequency for discarding techniques, and a parameter ε with $\varepsilon_\tau = \varepsilon \cdot 0.4\,\text{s}$ and $\varepsilon_\gamma = \varepsilon \cdot 1.0\,\text{kWh}$ for ε-dominance and combinations of ε-dominance and discarding (discarding frequency was fixed to 2^{10} in this case). Note that the point corresponding to the baseline algorithm is not contained in the plot (as it is beyond the visible region).

slightly outperforms other frequencies. The plot indicates that ε-dominance provides the best quality. Also, by varying the dominance parameters, we can easily trade query time and solution quality. Combining ε-dominance with delta discarding provides even faster queries with smaller error (for similar query times). Considering delta discarding on the other hand, early aborting greatly improves running time, with little impact on solution quality. Moreover, we can reduce the discarding frequency in order to decrease errors (at the cost of additional running time).

6 Conclusion

This paper dealt with computing sets of routes for electric vehicles that trade travel time and energy consumption via Pareto optimization. In the process, we are—to the best of our knowledge—the first to explicitly consider driving road segments at different speeds (below the limit) in order to save energy. Since by that the number of solutions increases significantly, we also proposed several improvements and heuristics, which (in their combination) accelerate query times (for the 16 kWh battery) from hours to just under a second with very little error in solution quality. This makes our approach practical, e. g., for onboard navigation systems.

Future work includes preprocessing for further speedup, enabling real-time applications. We are also interested in incorporating turn costs. This would make the routes more realistic, possibly even eliminating some insignificant Pareto-optimal solutions with tiny detours.

References

1 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*, volume 6359 of *Lecture Notes in Computer Science*, pages 309–316. Springer, 2010.

2 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. Technical Report MSR-TR-2014-4, Microsoft Research, 2014.

3 Lucas S. Batista, Felipe Campelo, Frederico G. Guimarães, and Jaime A. Ramírez. A Comparison of Dominance Criteria in Many-Objective Optimization Problems. In *IEEE Congress on Evolutionary Computation*, pages 2359–2366. IEEE, 2011.

4 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Energy-Optimal Routes for Electric Vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM Press, 2013.

5 Richard Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

6 Lee R. Dice. Measures of the Amount of Ecologic Association between Species. *Ecology*, 26(3):297–302, 1945.

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

8 Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2008.

9 Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal Route Planning for Electric Vehicles in Large Network. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*. AAAI Press, 2011.

10 Stephan Erb, Moritz Kobitzsch, and Peter Sanders. Parallel Bi-objective Shortest Paths Using Weight-Balanced B-trees with Bulk Updates. In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA'14)*, volume 8504 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014.

11 Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231. AAAI Press, 1996.

12 Stefan Funke and Sabine Storandt. Polynomial-Time Construction of Contraction Hierarchies for Multi-criteria Objectives. In *Proceedings of the 15th Meeting on Algorithm Engineering and Experiments (ALENEX'13)*, pages 31–54. SIAM, 2013.

13 Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, 1979.

14 Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route Planning with Flexible Objective Functions. In *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10)*, pages 124–137. SIAM, 2010.

15 Pierre Hansen. Bricriteria Path Problems. In *Multiple Criteria Decision Making – Theory and Application –*, pages 109–127. Springer, 1979.

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

17 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.

18 Lorenz Hübschle-Schneider. Speed-Consumption Trade-Off for Electric Vehicle Routing. Bachelor thesis, Karlsruhe Institute of Technology, 2013.

19 Enrique Machuca and Lawrence Mandow. Multiobjective Heuristic Search in Road Maps. *Expert Systems with Applications*, 39(7):6435–6445, 2012.

20 Lawrence Mandow and José-Luis Pérez-de-la-Cruz. Multiobjective A* Search with Consistent Heuristics. *Journal of the ACM*, 57(5):27:1–27:24, 2010.

21 Ernesto Queiros Martins. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research*, 26(3):236–245, 1984.

22 Matthias Müller–Hannemann and Karsten Weihe. Pareto Shortest Paths is Often Feasible in Practice. In *Proceedings of the 5th International Workshop on Algorithm Engineering (WAE'01)*, volume 2141 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2001.

23 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 Press, 2011.

24 Peter Sanders and Lawrence Mandow. Parallel Label-Setting Multi-Objective Shortest Path Search. In *Proceedings of the 27th International Parallel and Distributed Processing Symposium (IPDPS'13)*, pages 215–224. IEEE Computer Society, 2013.

25 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*, pages 20–25. ACM Press, 2012.

26 Chi Tung Tung and Kim Lin Chew. A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research*, 62(2):203–209, 1992.

27 Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *Evolutionary Computation, IEEE Transactions on*, 3(4):257–271, 1999.