Scalable Exact Visualization of Isocontours in Road Networks via Minimum-Link Paths*

Moritz Baum[1], Thomas Bläsius[2], Andreas Gemsa[3], Ignaz Rutter[4], and Franziska Wegner[5]

1 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
`moritz.baum@kit.edu`
2 **Hasso Plattner Institute, Potsdam, Germany**
`thomas.blaesius@hpi.de`
3 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
`andreas.gemsa@kit.edu`
4 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
`ignaz.rutter@kit.edu`
5 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
`franziska.wegner@kit.edu`

Abstract

Isocontours in road networks represent the area that is reachable from a source within a given resource limit. We study the problem of computing accurate isocontours in realistic, large-scale networks. We propose isocontours represented by polygons with minimum number of segments that separate reachable and unreachable components of the network. Since the resulting problem is not known to be solvable in polynomial time, we introduce several heuristics that run in (almost) linear time and are simple enough to be implemented in practice. A key ingredient is a new practical linear-time algorithm for minimum-link paths in simple polygons. Experiments in a challenging realistic setting show excellent performance of our algorithms in practice, computing near-optimal solutions in a few milliseconds on average, even for long ranges.

1998 ACM Subject Classification G.2.2 Graph Theory, G.2.3 Applications, I.3.5 Computational Geometry and Object Modeling

Keywords and phrases isocontours, separating polygons, minimum-link paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.7

1 Introduction

How far can I drive my battery electric vehicle (EV), given my position and the current state of charge? – This question can be answered by a map visualizing the reachable region. This region is bounded by *isocontours* representing points that require the same amount of energy to be reached. Isocontours are typically considered in the context of functions $f\colon \mathbb{R}^2 \to \mathbb{R}$, in our case describing the energy necessary to reach a point in the plane. However, f is defined only at certain points, namely vertices of the graph representing the road network. We have to fill the gaps by deciding how an isocontour should pass through regions between roads. The fact that the quality of the resulting visualization heavily depends on these decisions makes

* This work was partially supported by the EU FP7/2007-2013 under grant agreement no. 609026 (project MOVESMART) and the Helmholtz Program Storage and Cross-linked Infrastructures, Topic 6 Superconductivity, Networks and System Integration.

© Moritz Baum, Thomas Bläsius, Andreas Gemsa, Ignaz Rutter, and Franziska Wegner;
licensed under Creative Commons License CC-BY

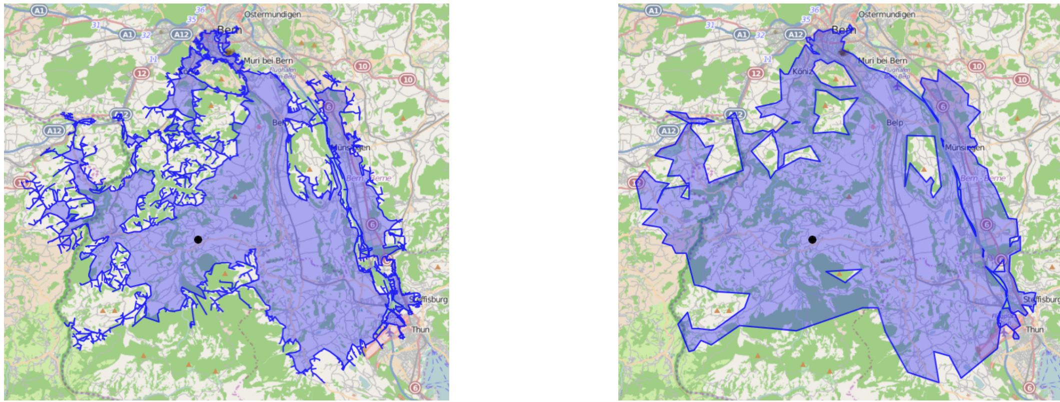

Figure 1 Isocontours in a mountainous area (near Bern, Switzerland), showing the range of an EV positioned at the black disk with a state of charge of 2 kWh. Note that the polygons contain holes, due to unreachable high-ground areas. Left: state-of-the-art approach (over 10 000 segments, computed by `RP-RC` from Section 4); right: our approach (416 segments, computed by `RP-CU`).

computing isocontours in road networks an interesting algorithmic problem. Besides range visualization for EVs, isocontour visualization is relevant in a wide range of applications, including reachability analyses in urban planning [2, 20, 22, 25], geomarketing [10], and environmental and social sciences [20].

Several techniques consider the problem of computing the subnetwork that is reachable within a given timespan (but not the actual isocontour), enabling query times in the order of milliseconds [4, 11, 12]. O'Sullivan et al. [25] introduce basic approaches for isocontour visualization based on merging shapes covering the reachable area. Marciuska and Gamper [22] propose isocontours induced by reachable points in the network, but their approaches are too slow for interactive applications (several seconds for small and medium ranges). In contrast, our work is motivated by more challenging scenarios, e. g., visualizing the range of high-end Tesla models or the area reachable by a truck driver within a day of work. Our algorithms are guided by three major objectives: Isocontours must be *exact*, i. e., correctly separate the reachable subgraph from the remaining unreachable part; they should be polygons of low *complexity*, i. e., consist of few segments (enabling fast rendering and a clear, uncluttered visualization); algorithms should be fast enough for interactive applications, even on large inputs. Figure 1 compares an example resembling state-of-the-art techniques [9, 10, 12, 22] to one of our approaches. The original works also consider inexact variants of the approach we refer to as state-of-the-art (e. g., by omitting holes or degeneracies from the polygon). We resort to an exact variant, in accordance with our objectives.

Contribution and Outline. We propose several new algorithms for computing polygons that represent isocontours in road networks. All approaches compute exact isocontours, while having low complexity. Their efficient performance is both proven in theory and demonstrated in practice on large, realistic instances. Section 2 states the precise problem and outlines our algorithmic approach. Section 3 attacks the important subproblem of separating the boundaries of a hole-free region by a polygon with minimum number of segments. While it can be solved in $O(n \log n)$ time [28], we propose a simpler algorithm that uses at most two additional segments, runs in linear time, and requires computation of only a single minimum-link path. We also propose a minimum-link path algorithm that is simpler than previous approaches [26]. Section 4 extends these results to the general case, where

unreachable parts of the network can induce holes between the boundaries to be separated. As the complexity of this problem is unknown, we focus on efficient heuristic approaches that work well in practice, but do not give (nontrivial) guarantees on the complexity of the resulting range polygons. Section 5 contains our extensive experimental evaluation. It demonstrates that all approaches are fast enough even for use in interactive applications. Section 6 concludes with final remarks. See the full version for omitted details and proofs [3].

2 Problem Statement and General Approach

Let $G = (V, E)$ be a road network, which we consider as a geometric graph where vertices have a fixed position in the plane and edges are straight-line segments between their endpoints. The function cons: $E \to \mathbb{R}$ assigns resource consumption to all edges. A source $s \in V$ and range $r \in \mathbb{R}_{\geq 0}$ partition the graph into two parts, one that is within range r from s, and the part that is not. A vertex v is *reachable* if the resource consumption $\text{cons}(\pi_v)$ on the shortest (wrt. a nonnegative length function on the edges) s–v-path π_v is at most r. An edge (u, v) is *passable* if it can be traversed in at least one direction, i. e., $\text{cons}(\pi_u) + \text{cons}((u, v)) \leq r$ or $\text{cons}(\pi_v) + \text{cons}((v, u)) \leq r$. Let V_r be the set of reachable vertices and E_r the set of passable edges. The *reachable subgraph* is $G_r = (V_r, E_r)$. Let $V_u = V \setminus V_r$ be the set of *unreachable vertices* and E_u the set of *unreachable edges* for which both endpoints are unreachable. The *unreachable subgraph* is $G_u = (V_u, E_u)$. Edges in $E \setminus (E_r \cup E_u)$ are called *boundary edges*. A *range polygon* is a plane (not necessarily simple) polygon P separating G_r and G_u, such that its interior contains G_r and has empty intersection with G_u. Note that if G is not planar, a range polygon may not even exist: If a passable edge intersects an unreachable edge, the requirements of including the passable and excluding the unreachable edge obviously contradict. To resolve this issue, we consider the planarization G_p of G, which is obtained from G by considering each intersection point p as a *dummy vertex* that subdivides all edges of G that contain p. A dummy vertex is reachable if and only if it subdivides a passable edge of the original graph. An edge in G_p is passable if and only if the edge in G containing it is passable. As before, an edge of G_p is unreachable if both endpoints are unreachable. Finally, let the graph G' consist of the union of the reachable and unreachable subgraph of G_p. A face of G' incident to both the reachable and unreachable subgraph is a *border region*.

Given a source $s \in V$ and a range $r \in \mathbb{R}_{\geq 0}$, we seek to compute a range polygon wrt. G_p that has the minimum number of holes, and among these we seek to minimize the complexity of the range polygon. This can be achieved as follows.

1. Compute the reachable and unreachable subgraph of G.
2. Planarize G, compute the reachable and unreachable subgraph of its planarization G_p.
3. Compute the border regions.
4. For each border region B, compute a simple polygon of minimum complexity contained in B that separates the unreachable components incident to B from the reachable component.

Step 1 is solved by a variant of Dijkstra's algorithm [8]. Tailored preprocessing-based algorithms for road networks exist [4, 11]. For Step 2, we planarize G during preprocessing in a single run of the well-known sweep line algorithm [6] to obtain G_p. In a query (i. e., for given $s \in V$ and $r \in \mathbb{R}_{\geq 0}$), reachability of dummy vertices is then determined in a linear scan of all original edges containing a dummy vertex. This produces limited overhead in practice, since the number of dummy vertices in graphs representing road networks is typically small (as large parts of the input are already planar). Border regions are extracted in Step 3 by traversing faces of G_p that contain at least one boundary edge. In the remainder of this work, we focus on Step 4. Each connected component of the boundary of a border region

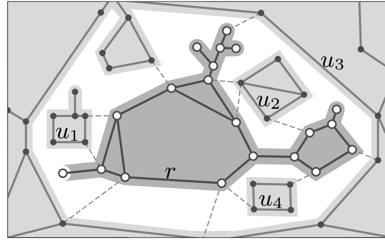

Figure 2 Border region (white), with the reachable boundary $R = \{r\}$ and the unreachable boundary with components $U = \{u_1, u_2, u_3, u_4\}$. Reachable and unreachable parts are shaded.

is a hole-free non-crossing polygon. Note that these polygons are not necessarily simple in the sense that they may contain the same segment twice in different directions; see Figure 2. Each border region is defined by two sets R and U of such polygons, where R contains the reachable components and U contains the unreachable components. We seek a simple polygon with minimum number of segments that separates U from R. Guibas et al. [17] showed that this problem is NP-complete in general. In our case, however, it is $|R| = 1$ since the reachable subgraph is, by definition, connected. Guibas et al. left this case as an open problem and, to the best of our knowledge, it has not been resolved.

3 Range Polygons in Border Regions Without Holes

In this section, we consider the special case of a border region B with $|R| = |U| = 1$. A polygon of minimum complexity that separates the two polygons can be found in $O(n \log n)$ time [28]. However, the algorithm is rather involved and requires computation of several minimum-link paths. We propose a simpler algorithm that uses at most two additional segments, runs in linear time, and computes a single minimum-link path. It adds an edge e to B that connects both boundaries. In the resulting polygon B', it computes a minimum-link path π that connects the two sides of e. The algorithm of Suri [26] finds such a path π in linear time. We obtain a separating polygon by connecting the endpoints of π along e.

We address the subproblem of computing a *minimum-link path* between two edges a and b of a simple polygon P, i. e., a polygonal path with minimum number of segments that connects a and b and lies in the interior of P. The algorithm of Suri [26] starts by triangulating the input polygon. We preprocess this step by triangulating all faces of the planarized input graph only once. Afterwards, in each step of Suri's algorithm a *window* (which we define in a moment) is computed. To this end, several visibilty polygons are constructed. This suffices to prove linear running time, but seems wasteful from a practical point of view. Below, we present a simpler linear-time algorithm for computing the windows, called FMLP (*fast minimum-link path*). It can be seen as a generalization of an algorithm for approximating piecewise linear functions [19].

Windows and Visibility. Let T be the graph obtained by arbitrarily triangulating P. Let t_a and t_b be the triangles incident to a and b, respectively. As T is an outerplanar graph, its (weak) dual graph has a unique path $t_a = t_1, t_2, \ldots, t_{k-1}, t_k = t_b$ from t_a to t_b; see Figure 3a. We call the triangles on this path *important* and their position in the path their *index*. The *visibility polygon* $V(a)$ of the edge a in P is the polygon that contains a point p in its interior if and only if there is a point q on a such that the line segment pq lies inside P. Let i be the highest index such that the intersection of the triangle t_i with the visibility polygon $V(a)$ is

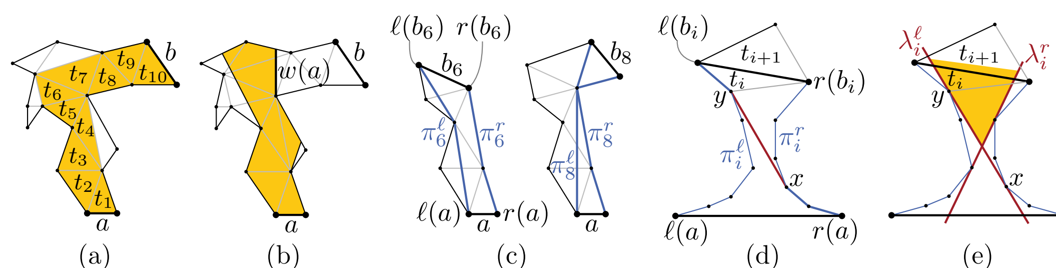

Figure 3 (a) Important triangles wrt. a and b. (b) The window $w(a)$ is an edge of the (shaded) visibility polygon. (c) The left and right shortest paths (blue) intersect for $i = 8$ but not for $i = 6$. (d) The shortest path from $r(a)$ to $\ell(b_i)$ contains the bold prefix of π_i^r, the red segment, and the bold suffix of π_i^ℓ. (e) Visibility lines spanning the (shaded) visibility cone.

not empty. The *window* $w(a)$ is the edge of $V(a)$ that intersects t_i closest (wrt. minimum Euclidean distance) to the edge between t_i and t_{i+1}; see Figure 3b. Note that $w(a)$ separates the polygon P into two parts. Let P' be the part containing the edge b that we want to reach. A minimum-link path from a to b in P can then be obtained by adding an edge from a to $w(a)$ to a minimum-link path from $w(a)$ to b in P'. Thus, the next window is computed in P' starting with the previous window $w(a)$. Below, we first describe how to compute the first window and then discuss what has to be changed to compute the subsequent windows.

Let T_i be the subgraph of T induced by the triangles $t_1, \dots, t_i$ and let P_i be the polygon bounding the outer face of T_i. The polygon P_i has two special edges, namely a and the edge shared by t_i and t_{i+1}, which we call b_i. Let $\ell(a)$ and $r(a)$, and $\ell(b_i)$ and $r(b_i)$ be the endpoints of a and b_i, respectively, such that their clockwise order is $r(a)$, $\ell(a)$, $\ell(b_i)$, $r(b_i)$ (think of $\ell(\cdot)$ and $r(\cdot)$ being the left and right endpoints, respectively); see Figure 3c. We define the *left shortest path* π_i^ℓ to be the shortest polygonal path (shortest in terms of Euclidean length) that connects $\ell(a)$ with $\ell(b_i)$ and lies inside or on the boundary of P_i. The *right shortest path* π_i^r is defined analogously for $r(a)$ and $r(b_i)$; see Figure 3c.

Assume that the edge b_i is visible from a, i.e., there exists a line segment in the interior of P_i that starts at a and ends at b_i. Such a visibility line separates the polygon into a left and a right part. Observe that it follows from the triangle inequality that the left shortest path π_i^ℓ and the right shortest path π_i^r lie inside the left and right part, respectively. Thus, these two paths do not intersect. Moreover, the two shortest paths are *outward convex* in the sense that the left shortest path π_i^ℓ has only left bends when traversing it from $\ell(a)$ to $\ell(b_i)$ (the symmetric property holds for π_i^r); see the case $i = 6$ in Figure 3c. We note that the outward convex paths are sometimes also called "inward convex" and the polygon consisting of the two outward convex paths together with the edges a and b_i is also called *hourglass* [15]. The following lemma, which is similar to a statement shown by Guibas et al. [16, Lemma 3.1], summarizes the above observation.

▶ **Lemma 1.** *If the triangle t_i is visible from a, then the left and right shortest path in P_{i-1} have empty intersection. Moreover, if these paths do not intersect, they are outward convex.*

Guibas et al. [16] argue that the converse of the first statement is also true, i.e., if the two paths have empty intersection, then the triangle t_{i+1} is visible from a. Their main arguments go as follows. The shortest path (wrt. Euclidean length) in the hourglass that connects $r(a)$ with $\ell(b_i)$ is the concatenation of a prefix of π_i^r, a line segment from a vertex x of π_i^r to a vertex y of π_i^ℓ, and a suffix of π_i^ℓ; see Figure 3d. We call the straight line through x and y the *left visibility line* and denote it by λ_i^ℓ. We assume λ_i^ℓ to be oriented from x to y and call

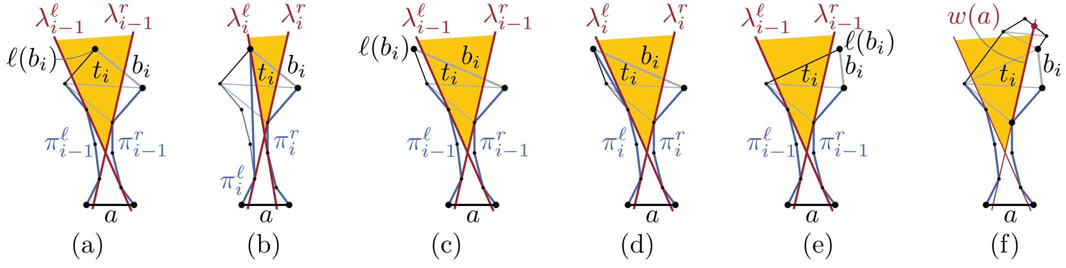

Figure 4 (a) The new vertex $\ell(b_i)$ lies in the visibility cone. (b) The updated left shortest path π_i^ℓ and left visibility line λ_i^ℓ. (c) The vertex $\ell(b_i)$ lies to the left of λ_{i-1}^ℓ. (d) The left shortest path has to be updated, the left visibility line remains unchanged. (e) The vertex $\ell(b_i)$ lies to the right of λ_{i-1}^r, i.e., t_{i+1} is not visible from a. (f) The window $w(a)$ is a segment of λ_{i-1}^r.

x and y the *source* and *target* of λ_i^ℓ. Analogously, one can define the *right visibility line* λ_i^r; see Figure 3e. We call the intersection of the half-plane to the right of λ_i^ℓ with the half-plane to the left of λ_i^r the *visibility cone*. It follows that the intersection of the visibility cone with the edge b_i is not empty and a point on the edge b_i is visible from a if and only if it lies in this intersection [16]. This directly extends to the following lemma.

▶ **Lemma 2.** *If the left and right shortest path in P_{i-1} have empty intersection, t_i is visible from a. Moreover, a point in t_i is visible from a if and only if it lies in the visibility cone.*

The above observations then justify the following approach for computing the window. We iteratively increase i until the left and the right shortest path of the polygon P_i intersect. We then know that the triangle t_{i+1} is no longer visible; see Lemma 1. Moreover, as the left and the right shortest path did not intersect in P_{i-1}, the triangle t_i is visible from a; see Lemma 2. To find the window, it remains to find the edge of the visibility polygon $V(a)$ that intersects t_i closest to the edge between t_i and t_{i+1}. Thus, by the second statement of Lemma 2, the window must be a segment of one of the two visibility lines. It remains to fill out the details of this algorithm, argue that it runs in overall linear time, and describe what has to be done in later steps, when we start at a window instead of an edge.

Computing the First Window. We start with the details of the algorithm starting from an edge. Assume the triangle t_i is still visible from a, i.e., π_{i-1}^ℓ and π_{i-1}^r do not intersect. Assume further that we have computed the left and right shortest path π_{i-1}^ℓ and π_{i-1}^r as well as the corresponding visibility lines λ_{i-1}^ℓ and λ_{i-1}^r in a previous step. Assume without loss of generality that the three corners of the triangle t_i are $\ell(b_{i-1})$, $\ell(b_i)$, and $r(b_i) = r(b_{i-1})$. There are three possibilities shown in Figure 4, i.e., the new vertex $\ell(b_i)$ lies either in the visibility cone spanned by λ_{i-1}^ℓ and λ_{i-1}^r (Figure 4a), to the left of the left visibility line λ_{i-1}^ℓ (Figure 4c), or to the right of the right visibility line λ_{i-1}^r (Figure 4e).

By Lemma 2, a point in t_i is visible from a if and only if it lies inside the visibility cone. Thus, the edge b_i between t_i and t_{i+1} is no longer visible if and only if the new vertex $\ell(b_i)$ lies to the right of λ_{i-1}^r; see Figure 4e. In this case, we can stop and the desired window $w(a)$ is the segment of λ_{i-1}^r starting at its touching point with π_{i-1}^r and ending at its first intersection with an edge of P; see Figure 4f.

In the other two cases (Figure 4a and Figure 4c), we have to compute the new left and right shortest path π_i^ℓ and π_i^r and the new visibility lines λ_i^ℓ and λ_i^r (Figure 4b and Figure 4d). Note that the old and new right shortest path π_{i-1}^r and π_i^r connect the same endpoints $r(a)$ and $r(b_{i-1}) = r(b_i)$. As the path cannot become shorter by going through the new triangle t_i,

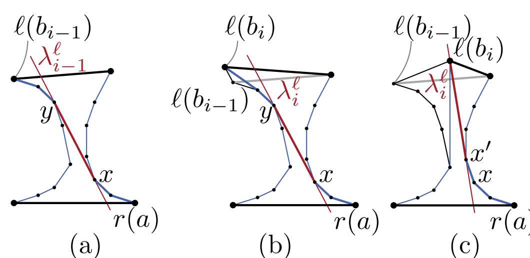

Figure 5 (a) The shortest path from $r(a)$ to $\ell(b_{i-1})$ (bold) defining the left visibility line λ^ℓ_{i-1}. (b) The visibility line does not change if $\ell(b_i)$ lies to the left of λ^ℓ_{i-1}. (c) Illustration how the visibility line changes when $\ell(b_i)$ lies to the right of λ^ℓ_{i-1}.

we have $\pi^r_i = \pi^r_{i-1}$. The same argument shows that $\lambda^r_i = \lambda^r_{i-1}$ (recall that the visibility lines were defined using a shortest path from $\ell(a)$ to $r(b_{i-1}) = r(b_i)$).

We compute the new left shortest path π^ℓ_i as follows. Let x be the latest vertex on π^ℓ_{i-1} such that the prefix of π^ℓ_{i-1} ending at x concatenated with the segment from x to $\ell(b_i)$ is outward convex. We claim that π^ℓ_i is the path obtained by this concatenation, i.e., this path lies inside P_i and there is no shorter path lying inside P_i. It follows by the outward convexity, that there cannot be a shorter path inside P_i from $\ell(a)$ to $\ell(b_i)$. Moreover, by the assumption that π^ℓ_{i-1} was the correct left shortest path in P_{i-1}, the subpath from $\ell(a)$ to x lies inside P_i. Assume for contradiction that the new segment from x to $\ell(b_i)$ does not lie entirely inside P_i. Then it has to intersect the right shortest path and it follows that the right shortest path and the correct left shortest path have non-empty intersection, which is not true by Lemma 1.

To get the new left visibility line λ^ℓ_i, we have to consider the shortest path in P_i that connects $r(a)$ with $\ell(b_i)$. Let x and y be the source and target of λ^ℓ_{i-1}, respectively, i.e., the shortest path from $r(a)$ to $\ell(b_{i-1})$ is as shown in Figure 5a. If the new vertex $\ell(b_i)$ lies to the left of λ^ℓ_{i-1} (Figure 5b), then the shortest path from $r(a)$ to $\ell(b_i)$ also includes the segment from x to y. Thus, $\lambda^\ell_i = \lambda^\ell_{i-1}$ holds in this case. Assume the new vertex $\ell(b_i)$ lies to the right of λ^ℓ_{i-1} (Figure 5c). Let x' be the latest vertex on the path π^r_i such that the concatenation of the subpath from $r(a)$ to x' with the segment from x' to the new vertex $\ell(b_i)$ is outward convex in the sense that it has only right bends; see Figure 5c. We claim that this path lies inside P_i and that there is no shorter path inside P_i. Moreover, we claim that x' is either a successor of x in π^r_{i-1} or $x' = x$. Clearly, the concatenation of the path from $r(a)$ to x with the segment from x to $\ell(b_i)$ is outward convex, thus the latter claim follows. It follows that the segment from x' to $\ell(b_i)$ lies to the right of the old visibility line λ^ℓ_{i-1}. Thus, it cannot intersect the path π^ℓ_i (except in its endpoint $\ell(b_i)$), as π^ℓ_{i-1} lies to the left of λ^ℓ_{i-1}. Moreover, as we chose x' to be the last vertex on π^r_{i-1} with the above property, this new segment does not intersect π^r_i (except in x'). Hence, the segment from x' to $\ell(b_i)$ lies inside P_i. As before, it follows from the convexity that there is no shorter path inside P_i. Thus, λ^ℓ_i is the line through x' and $\ell(b_i)$ (x' is the new source and $\ell(b_i)$ is the new target).

▶ **Lemma 3.** *Let t_h be the triangle with the highest index that is visible from a. Then, the algorithm FMLP computes the first window $w(a)$ in $O(h)$ time.*

Computation of Subsequent Windows. As mentioned before, the first window $w(a)$ we compute separates P into two smaller polygons. Let P' be the part including the edge b (and not a). In the following, we denote $w(a)$ by a'. To get the next window $w(a')$, we have to apply the above procedure to P' starting with a'. However, this would require to partially

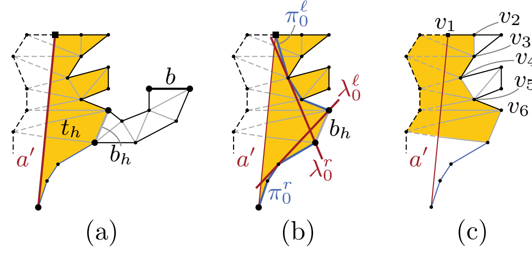

Figure 6 (a) Polygon P' after computing the window a'; P_0' is shaded. (b) Shortest paths π_0^ℓ and π_0^r (blue) and visibility lines (red). (c) Sequence $v_1, \ldots, v_6$, triangles intersected by a' are shaded.

retriangulate the polygon P'. More precisely, let t_h be the triangle with the highest index that is visible from a and let b_h be the edge between t_h and t_{h+1}; see Figure 6a. Then b_h separates P' into an initial part P_0' (the shaded part in Figure 6a) and the rest (having b on its boundary). The latter part is properly triangulated, however, the initial part P_0' is not. The conceptually simplest solution is to retriangulate P_0'. However, this would require an efficient subroutine for triangulation (and dynamic data structures that allow us to update P and T, which produces overhead in practice). Instead, we propose a much simpler method for computing the next window. The general idea is to compute the shortest paths in P_0' from $\ell(a')$ to $\ell(b_h)$ and from $r(a')$ to $r(b_h)$; see Figure 6b. We denote these paths by π_0^ℓ and π_0^r, respectively. Moreover, we want to compute the corresponding visibility lines λ_0^ℓ and λ_0^r. Afterwards, we can continue with the correctly triangulated part as before.

Concerning the shortest paths, note that the right shortest path π_0^r is a suffix of the previous right shortest path, which we already know. For the left shortest path π_0^ℓ, first consider the polygon induced by the triangles intersected by a'; see Figure 6c. Let $v_1, \ldots, v_g$ be the path on the outer face of this polygon (in clockwise direction) from $\ell(a') = v_1$ to $\ell(b_h) = v_g$. We obtain π_0^ℓ using *Graham's scan* [14] on the sequence $v_1, \ldots, v_g$, i. e., starting with an empty path, we iteratively append the next vertex of the sequence $v_1, \ldots, v_g$ while maintaining the path's outward convexity by successively removing the second to last vertex, if necessary. It remains to compute the visibility lines λ_0^ℓ and λ_0^r in the hourglass consisting of a', b_h, and the paths π_0^ℓ and π_0^r. Note that the whole edge b_h is visible from a', since a' intersects the triangle t_h. Thus, the visibility lines go through the endpoints of b_h. It follows that λ_0^ℓ is the line that goes through $\ell(b_h)$ and the unique vertex on π_0^r such that it is tangent to π_0^r; see Figure 6b. This can clearly be found in linear time in the length of π_0^r. The same holds for the right visibility line.

▶ **Lemma 4.** *The algorithm FMLP computes the initial left and right shortest paths π_0^ℓ and π_0^r as well as the corresponding visibility lines in $O(|P_0'|)$ time.*

We compute subsequent windows until we find the last edge b. A minimum-link path π is obtained by connecting each window $w(a)$ to its corresponding first edge a with a straight line [26]. In our implementation, we do not construct P and its triangulation T explicitly, but work directly on the triangulated input graph. The next important triangle is then computed on-the-fly as follows. Consider an important triangle $t_i = uvw$, and let uv be the edge shared by the current and the previous important triangle. Clearly, exactly one endpoint of uv is part of the reachable boundary, so without loss of generality let u be this endpoint. Then the next important triangle is the triangle sharing vw with t_i if w is reachable, and the triangle sharing uw with t_i otherwise. In other words, the next triangle is determined by the unique edge that has exactly one reachable endpoint. Linear running time of the algorithm follows immediately from Lemma 3 and Lemma 4. Theorem 5 summarizes our findings.

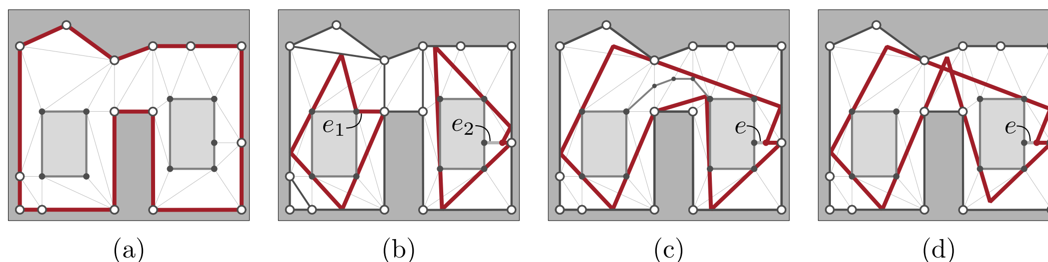

Figure 7 Results of RP-RC (a), RP-TS (b), RP-CU (c), and RP-SI (d), starting at indicated edges.

▶ **Theorem 5.** *Given two edges a and b of a simple polygon P, the algorithm FMLP computes a minimum-link path from a to b contained in P in linear time.*

4 Heuristic Approaches for General Border Regions

A border region B may consist of several unreachable components, i. e., $|U| > 1$. In this general case, it is not clear whether one can compute a (non-intersecting) range polygon of minimum complexity in polynomial time [17]. Even for the simpler subproblem of computing a minimum-link path in a polygon with holes (without assigning the holes to the reachable or unreachable part), the fastest known algorithm has quadratic running time [21, 24]. This is impractical for large instances. We propose heuristics with (almost) linear running time (in the size of B) that are simple and fast in practice. Figure 7 shows examples.

The first approach, RP-RC (*range polygon*, extracted *reachable components*), simply extracts and returns the reachable boundary R; see Figure 7a. The result resembles previous approaches [12, 22], so it can be seen as an efficient implementation of the state-of-the-art.

Separating Border Regions Along the Triangulation. The second approach, denoted RP-TS (*triangular separators*), works as follows. For each border region B, we consider its triangulation. We add all edges of the triangulation that either connect two reachable vertices or two unreachable vertices of G_p to the boundary of B, possibly splitting B into several regions $B' = R' \cup U'$ (see bold edges separating the border region in Figure 7b). For each region B', we obtain $|U'| \leq 1$, since two components of U must be connected by an edge of the triangulation or separated by an edge with two endpoints in R. Then, we run the algorithm presented in Section 3 on each instance B' with $|U'| = 1$ to get the range polygon. Linear running time follows, as FMLP is run on disjoint subregions of B.

Clearly, the set of edges added to B is not minimal (we could possibly omit some and still obtain $|U'| \leq 1$). However, it allows us to implement RP-TS without explicitly constructing B': Starting from an arbitrary (unvisited) boundary edge in B with one endpoint in each R and U, we run FMLP and determine the important triangles on-the-fly (as described in Section 3). We mark encountered boundary edges and repeat this procedure until all boundary edges in B (with endpoints in both R and U) were visited. Note that FMLP becomes even simpler in this variant, because regions B' contain only important triangles. However, the number of modified regions B' can become quite large. Below, we propose a more sophisticated approach to obtain regions with a single unreachable component.

Connecting Unreachable Components. Third, RP-CU (*connecting unreachable* components) adds new edges to border regions B with $|U| > 1$ to connect all components in U without intersecting the reachable boundary; see Figure 7c.

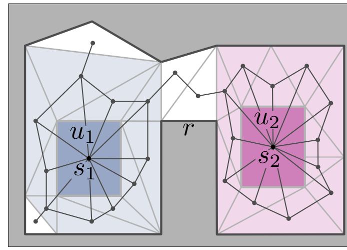

Figure 8 Dual graph with super sources s_1 and s_2. Shaded triangles are assigned to u_1 and u_2, respectively.

Given a border region B with $|U| > 1$, the heuristic starts by checking for each pair of components in U whether it can be connected directly by an edge of the triangulation. This requires traversal of the vertices in unreachable components, scanning for each vertex its incident edges in the triangulation. Edges that connect two unreachable components are added to the boundary of B and the adjacent components are merged (i. e., considered equal in the further course of the algorithm). To connect all remaining unreachable components after this first step, we consider the (weak) dual graph of the triangulation of B; see Figure 8. Since no pair of remaining unreachable components can be connected by a single edge in the primal graph, each triangle intersects at most one unreachable component. We assign a component to each dual vertex, namely, the reachable component if the corresponding triangle contains only reachable vertices, and the unique unreachable component it intersects, otherwise. For each unreachable component, we add a *super source* to the dual graph that is connected to all vertices assigned to this component. Since we want to add as few edges to the primal graph as possible, our goal is to find a tree of minimum total length in the modified dual graph that connects all super sources. Finding such a minimum Steiner tree is NP-hard [13], so we run a heuristic search. It iteratively adds shortest paths between two sources that are not yet connected in a greedy fashion. This is achieved by a multi-source variant of a breadth-first search (BFS) starting from all super sources. Whenever a path connecting two super sources is found, the corresponding components are merged. The algorithm stops when all super sources are connected. Finally, we add new vertices and edges to B along the obtained paths to connect all components in U, as illustrated in Figure 7c. We add further edges to maintain the triangulation, if necessary. The resulting border region B' is solved by the algorithm presented in Section 3. Making use of a union-find data structure, the BFS runs in $O(n\alpha(n))$ time [27], where n is the size of B and α the inverse Ackermann function. All remaining steps run in linear time, so the overall running time is almost linear.

A crucial observation is that realistic instances of border regions often consist of one major unreachable component and many tiny components, as illustrated in Figure 2. To significantly reduce the (empirical) running time, we start the BFS from all but the largest component. This requires little overhead (traversing unreachable components to identify the largest one), but searches from small components are likely to quickly converge to the large component. Moreover, after extracting the next vertex from the queue, we first check whether its source was connected to the largest component in the meantime. If this is the case, we prune the search at this vertex, because it now represents the search from the largest component. Similarly, before running the BFS we also omit traversal of the vertices of the largest component when checking for edges in the triangulation that connect two components. For further (practical) speedup, we modify the BFS to always expand the search from the component that is currently the smallest. This can be done by using a priority queue whose

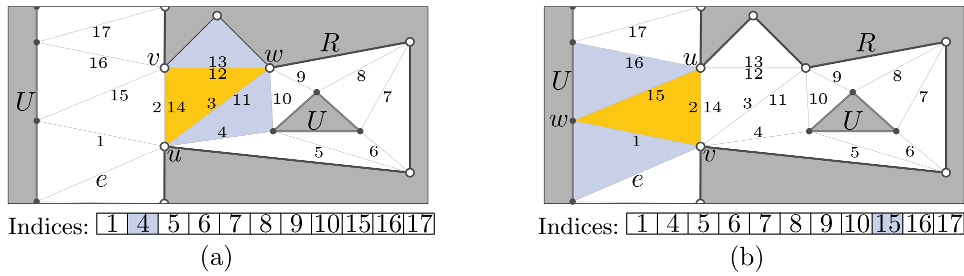

Figure 9 Edge indices starting at e for each direction of traversal (∞ if not specified). Separator edge indices are given in the list (the next index is shaded). (a) The second triangle (shaded yellow) has two possible next triangles t_{uw} and t_{vw} (shaded blue). The next one is t_{uw}, since the index of the edge vw with higher index (12) is greater than the next separator edge index (4). Observe that the yellow triangle is visited a second time during the course of the algorithm. (b) The next triangle is t_{uw}. The index of the next separator edge is updated from 15 to 16.

elements are components. Additionally, we maintain a queue for each component that stores the vertices visited by the BFS (extracting them in first-in-first-out order). In each step of the BFS, we check for the smallest component in the priority queue, and extract the next vertex from the queue of this component. Note that the use of a priority queue actually increases the asymptotic running time of the BFS, however, we observe a significant speedup in practice.

Self-Intersecting Minimum-Link Paths. Our last approach is denoted `RP-SI` (*self intersecting* polygons). It computes a minimum-link path that separates reachable and unreachable boundaries of B. This path has at most $\text{OPT} + 1$ segments (inducing a lower bound for B), but may intersect itself; see Figure 7d. To obtain a range polygon, we add more segments to resolve intersections. Below, we generalize FMLP to border regions with several unreachable components. Note that the (weak) dual graph of the triangulation of B is not outerplanar if $|U| > 1$. Thus, paths between dual vertices are no longer unique and we have to compute a *shortest* path in the dual graph that separates the boundaries. Vertices may even occur multiple times in such a path; see the triangles crossed by the polygon in Figure 7d.

Given a boundary edge e with endpoints in R and U, we compute a sequence $t_1, \ldots, t_k$ of important triangles that must be passed in this order by a minimum-link path (between the two sides of e) that separates R and U. Our approach runs in two phases. Exploiting that the reachable boundary is connected, the first phase traverses it starting from e. It assigns *indices* to all edges in the triangulation incident to the reachable boundary, according to the order in which they are traversed. In doing so, we distinguish both sides of edges; see Figure 9. For consistency, sides of edges that are not traversed get the index ∞. During this traversal, we also collect an ordered list of indices corresponding to *separator edges* in the triangulation, i. e., edges with one endpoint in each R and U. Every separator edge in B is traversed exactly once. Moreover, the minimum-link path must intersect these edges in the same order (lest having unreachable components on both sides of the path).

The second phase uses this information to compute the actual sequence of important triangles. This sequence must pass all separator edges exactly once and in increasing order of their indices. Therefore, we obtain the sequence of important triangles by computing shortest paths in the dual graph between pairs of consecutive separator edges. We maintain the index of the next separator edge that was not traversed yet, initialized to the first element of the list. Starting at the triangle t_1 containing the first edge e, we add triangles to the sequence

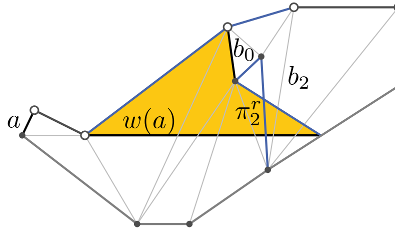

Figure 10 The path π_2^r (blue) between the right endpoints of $w(a)$ and b_2 intersects itself; P_0' is shaded. Note that two unreachable vertices (black) are not connected to the remaining unreachable boundary.

of important triangles until e is reached again. Let $t_i = uvw$ denote the previous triangle appended to this sequence, and uv the edge shared by t_i and t_{i-1} (if $i = 1$, let $uv = e$). We determine the next important triangle t_{i+1}; see Figure 9. We consider the possible next triangles t_{uw} containing the edge uw and t_{vw} containing vw. Without loss of generality, let the index of uw be lower than the index of vw (and thus, finite). This implies that uw is not contained in the boundary of B. If both u and w are part of the reachable boundary, uw separates B into two subregions; see Figure 9a. Thus, t_{uw} is the next triangle if and only if the subregion containing t_{uw} contains a separator edge that was not passed yet. Therefore, we continue with t_{uw} if and only if the index of the other edge vw is higher than the index of the next separator edge. If either u or w is part of the unreachable boundary, uw is the next separator edge; see Figure 9b. We update the index of the next separator edge to the next element in the according list. We continue until the first edge e is reached again. Note that this second phase can be performed on-the-fly within FMLP.

To preserve correctness of FMLP, further modifications are necessary, as a path in the hourglass may intersect itself. Figure 10 shows an example where the subpath of the right shortest path π_2^r starting at edge b_0 intersects the segment from the right endpoint of $w(a)$ to the right endpoint of b_0. The last segment of the right shortest path π_2^r from $w(a)$ to b_2 is called *visibility-intersecting*, as it reaches into the area P_0' visible from $w(a)$. Visibility-intersecting segments may lead to wrong results in certain cases [3]. However, one can show that a segment is visibility-intersecting if and only if it intersects the previous window. Moreover, it can safely be omitted from the shortest path computed by the algorithm without affecting correctness. Thus, FMLP is restored with a simple additional intersection test. In summary, our algorithm consists of two steps (traversing the reachable boundary, running a modified version of FMLP), which clearly run in linear time. The resulting polygon has at most OPT + 2 segments. We rearrange it at intersections to obtain the range polygon [3].

5 Experiments

We implemented all approaches in C++, using g++ 4.8.3 (-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.

Our graph is based on the road network of Western Europe, kindly provided by PTV AG (`ptvgroup.com`). Edge lengths are set to given travel times. For EV range visualization, we also consider energy consumption derived from a detailed micro-scale emission model [18]. Removing edges without reasonable energy consumption (e. g., due to missing elevation data), we obtain a graph with 22 198 628 vertices and 51 088 095 edges. To improve spatial locality, we reorder these vertices according to a vertex partition [5]. During preprocessing, the graph is planarized (654 765 split edges, 293 741 dummy vertices) and triangulated.

Table 1 Computing isochrones and EV range for medium and long ranges. We report average figures for the number of components of the range polygon (Cp.), complexity of the range polygon (Seg.), number of self-intersections (Int.), and running time of the algorithm in ms (Time).

		Med. (16 kWh / 60 min)				Long (85 kWh / 500 min)			
	Algorithm	Cp.	Seg.	Int.	Time	Cp.	Seg.	Int.	Time
EV Range	RP-RC	41	19 396	—	4.50	131	92 554	—	9.46
	RP-TS	69	610	—	4.30	219	1 973	—	7.78
	RP-CU	41	561	—	10.15	131	1 820	—	25.11
	RP-SI	41	549	4.79	7.52	131	1 781	15.06	22.25
Isochrones	RP-RC	53	22 458	—	4.75	231	238 123	—	20.25
	RP-TS	151	1 076	—	4.65	694	4 981	—	14.96
	RP-CU	53	913	—	12.11	231	4 208	—	65.09
	RP-SI	53	881	9.95	8.70	231	4 055	45.80	51.94

We consider two scenarios, namely isochrones (i. e., travel time is the consumed resource) and range visualization of an EV. For both, we evaluate queries of medium (60 min and 16 kWh, respectively) and long ranges (500 min and 85 kWh). We focus on Steps 2 to 4 outlined in Section 2, since implementation of the first step was examined in previous work [4].

Evaluating Queries. Table 1 shows results for the different scenarios. Each figure is the average of 1 000 queries, with source vertices picked uniformly at random. For `RP-SI`, figures are reported as-is after running FMLP (i. e., for polygons with self-intersections). Thus, figures slightly change after resolving the intersections (both the number of components and the complexity may increase). All approaches perform excellently in practice, with timings of at most 65 ms even for long ranges. The simpler algorithms, `RP-RC` and `RP-TS` are faster by a factor of 2 to 5. On the other hand, range polygons generated by `RP-RC` have a much higher complexity, exceeding the optimum by more than an order of magnitude. For long ranges, polygons consist of more than 200 000 segments. This clearly justifies the use of our novel algorithms. Besides a more appealing visualization, a significant decrease in complexity enables fast rendering and more efficient transmission over mobile networks. The heuristic `RP-TS` provides much better results in terms of complexity, but is still outperformed by the other two approaches. Moreover, the triangular separation increases the number of components (i. e., the number of holes) in the result by up to a factor of 3, while all other approaches are optimal in this criterion. The two more involved approaches, `RP-CU` and `RP-SI`, keep the complexity close to the optimum, so the additional effort clearly pays off. Deriving lower bounds from the results of `RP-SI`, the average relative error of both `RP-CU` (at most 7 %) and `RP-SI` (4 %) is negligible in practice. The number of intersections produced by `RP-SI` is also rather low, but the majority of computed range polygons contains at least one intersection. Isochrones are slightly harder to solve in all cases. For long-range queries, this is due to larger border regions. Setting the resource limit to 500 minutes for isochrones yields one of the hardest scenarios in our instance. For medium ranges, border region sizes are similar in both scenarios. Here, differences in performance can be explained by different shapes of the border regions: Isocontours representing the range of an EV typically have a more circular shape (highways allow to move faster, but also consume more energy). On the contrary, isochrones require more segments and yield more challenging scenarios.

Table 2 Different phases (isochrones, 500 min), showing average time (in ms) for border region extraction (BE), connecting components (CC), range polygon computation with FMLP (RP), testing for self-intersections (SI), and total time (Total).

Algo.	BE	CC	RP	SI	Total
RP-RC	12.01	—	—	—	20.25
RP-TS	—	—	6.45	—	14.96
RP-CU	26.66	22.99	7.81	—	65.09
RP-SI	31.79	—	9.53	2.34	51.94

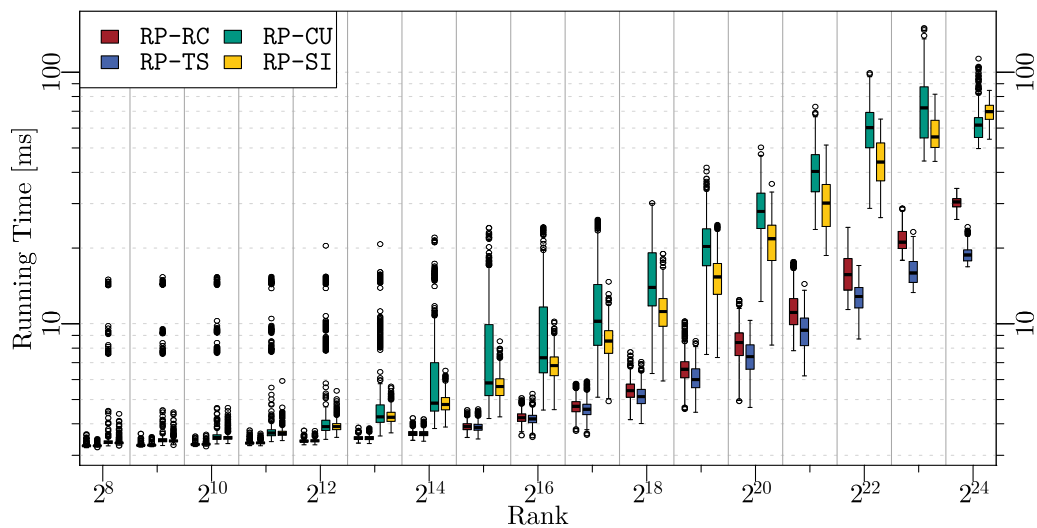

Figure 11 Running times of all approaches subject to Dijkstra rank. Smaller ranks indicate queries of shorter range. For each rank, we report results of 1 000 random queries.

Table 2 shows details on different phases of the algorithms for the hardest scenario (isochrones, 500 min). Step 2 (transferring input to the planar graph) is identical for all approaches, taking 8.2 ms on average (not reported in the table). Border region extraction does not apply to RP-TS, where this is done implicitly. Since RP-RC extracts only the reachable boundary, this takes less than half the time compared to RP-CU (the reachable boundary is typically smaller). On the other hand, RP-SI spends most time in this step, since it runs on the triangulated graph.

Despite its simplicity, extraction takes a major fraction of the total effort. This is due to the size of the border regions (500 000 segments per query), while only parts of them are visited in later phases. Consequently, connecting unreachable components takes less time for RP-CU. Running FMLP is fastest for RP-TS, since it visits only important triangles. The slowest approach in this phase is RP-SI, mostly because no artificial edges are added to border regions (windows become longer on average, increasing the number of visited triangles).

Evaluating Scalability. Figure 11 analyzes scalability of our algorithms. We follow the methodology of Dijkstra ranks [1], defined as the number of queue extractions performed by Dijkstra's algorithm in a shortest-path query. Higher ranks reflect harder queries. To generate queries, we ran Dijkstra's algorithm 1 000 times from sources chosen uniformly at random. For a source s, consider the resource consumption c at the vertex extracted from the queue in step 2^i of the algorithm. We consider a query from s with range c as a query of rank 2^i. For each rank in $\{2^1, \dots, 2^{\log |V|}\}$, we obtain 1 000 queries this way.

Query times of all approaches increase with the Dijkstra rank, which correlates well with the complexity of the border region. Scaling behavior is similar for all approaches: In accordance with our theoretical findings, it increases linearly in the size of the border region for queries beyond a rank of 2^{12}. For queries of lower rank, transferring the input to the planar graph dominates running time (which is linear in the graph size). The approach `RP-TS` is consistently the fastest on average for ranks beyond 2^{16}. Except for very few outliers, query times are well below 100 ms. For more local queries (i. e., smaller ranges), query times are much faster (20 ms and below if the rank is at most 2^{20}, corresponding to about a million vertices visited by Dijkstra's algorithm). Interestingly, the more expensive approaches have a higher variance and produce more outliers, which is explained by their more complex phases. For example, the performance of the BFS used in `RP-CU` heavily depends on how close unreachable components of the border region are in the dual graph.

Minimum-Link Path Computation. In Table 3, we evaluate FMLP in the four main scenarios (ranges for 16 kWh and 85 kWh batteries and isochrones for 60 min and 500 min). For each of the 1 000 queries per scenario, we modified the largest border region such that $|U| = 1$ (using `RP-CU`). Then, we added an edge connecting the two remaining components and computed a minimum-link path between its two sides. We also report figures for Suri's algorithm [26], which finds the next window starting from a window a by computing multiple visibility polygons in the following way. Starting with the polygon bounding all important triangles intersected by a, it doubles the number of important triangles until a triangle is (partially) invisible from a. To obtain the window, a final visibility polygon is computed in a polygon bounding the same set of important triangles together with all non-important triangles whose closest important triangle (wrt. distance in the dual graph) belongs to this set. The next window is an edge of this visibility polygon. While a fair comparison with running times of Suri's algorithm is beyond the scope of this work (as it requires an equally tuned implementation), we provide implementation-independent measures. In particular, we report the total number of visible triangles per query, in all polygons that require visibility computation. A recent experimental study on visibility polygon computation [7] presents a practical algorithm based on triangulations that processes only visible triangles, making this figure a good indicator for running time of an efficient implementation of Suri's algorithm.

Different scenarios in Table 3 represent certain levels of difficulty. As expected, the number of segments of the resulting path and the running time of FMLP increase with the complexity of the input. In all scenarios, Suri's algorithm requires several thousand calls to its subroutine for computing visibility polygons. The total number of segments in the input of this subroutine is beyond 1.5 million for long-range isochrones, which even rules out their explicit construction in practice. Additionally, the total number of triangles visited by Suri's algorithm (using the proposed subroutine [7]) is larger by a factor of 5 to 6. Moreover, we argue that this factor is a rather conservative estimate on the resulting speedup: While the workload per triangle is very low for FMLP, the proposed subroutine of Suri's algorithm is recursive and therefore possibly less cache efficient. Suri's algorithm also requires additional

Table 3 Average performance of minimum-link path algorithms. For each scenario, we report complexity of the input polygon ($|P|$) and minimum number of links in resulting paths (Seg.). For Suri's algorithm [26], we show the number of computed visibility polygons (V. Pol.), the total number of segments in the input for these computations (Pol. Seg.), and the total number of visible triangles (Trng.). For FMLP, we provide the number of visited triangles (Trng.) and running time in ms.

			Suri [26]			FMLP	
Scenario	$\|P\|$	Seg.	V. Pol.	Pol. Seg.	Trng.	Trng.	Time
EV, 16 kWh	134 049	415	2 010	307 583	48 762	8 901	0.74
Iso, 60 min	135 112	700	3 413	320 244	57 549	11 250	1.05
EV, 85 kWh	357 335	1 328	6 442	850 293	178 574	31 657	3.17
Iso, 500 min	637 224	3 203	15 655	1 547 962	359 969	66 163	6.67

overhead for generating input polygons and determining the actual windows. Given its simplicity, we conclude that FMLP is much more suitable for practical use. Even for input consisting of more than half a million segments, it takes less than 7 ms.

6 Conclusion

This work introduced algorithms for computing isocontours in large-scale road networks. Following the objectives of exact results, low result complexity, and practical performance, we presented three novel algorithms to compute near-optimal solutions in (almost) linear time. Their key ingredient is a new linear-time algorithm for minimum-link paths in simple polygons, making it the first practical approach to a problem well-studied in theory [21, 23, 24, 26]. Our experimental evaluation reveals that all approaches are fast enough for interactive applications on inputs of continental scale.

There are multiple lines of future work. Extending our algorithms to the case $|R| > 1$ is an open problem relevant for multi-source isocontours and multimodal networks [12]. For aesthetic reasons, one could aim at avoiding long straight segments in the range polygon (which are likely to occur in faces encompassing large areas corresponding to, e. g., big lakes or mountains). Such constraints could be integrated by adding (during preprocessing) artificial boundaries to faces whose area exceeds a certain threshold. Alternatively, one could further reduce result complexity at the cost of inexact results. However, such methods should avoid intersections between different components of the range polygon (i. e., maintain its topology), and error measures should consider the graph-based distance from the source to parts of the network that are classified incorrectly (since vertices close to each other wrt. Euclidean distance may in fact be far apart in the graph).

Acknowledgements. We thank Roman Prutkin for interesting discussions.

References

1 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 abs/1504.05140, ArXiv e-prints, 2015.

2 Veronika Bauer, Johann Gamper, Roberto Loperfido, Sylvia Profanter, Stefan Putzer, and Igor Timko. Computing Isochrones in Multi-Modal, Schedule-Based Transport Networks.

In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'08)*, pages 78:1–78:2. ACM, 2008.

3 Moritz Baum, Thomas Bläsius, Andreas Gemsa, Ignaz Rutter, and Franziska Wegner. Scalable Isocontour Visualization in Road Networks via Minimum-Link Paths. Technical Report abs/1602.01777, ArXiv e-prints, 2016.

4 Moritz Baum, Valentin Buchhold, Julian Dibbelt, and Dorothea Wagner. Fast Exact Computation of Isochrones in Road Networks. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA'16)*, volume 9685 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2016.

5 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 (GIS'13)*, pages 54–63. ACM, 2013.

6 John L. Bentley and Thomas A. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers*, 28(9):643–647, 1979.

7 Francisc Bungiu, Michael Hemmer, John E. Hershberger, Kan Huang, and Alexander Kröller. Efficient Computation of Visibility Polygons. In *Proceedings of the 30th European Workshop on Computational Geometry (EuroCG'14)*, 2014.

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

9 Alexandros Efentakis, Sotiris Brakatsoulas, Nikos Grivas, Giorgos Lamprianidis, Kostas Patroumpas, and Dieter Pfoser. Towards a Flexible and Scalable Fleet Management Service. In *Proceedings of the 6th ACM SIGSPATIAL International Workshop on Computational Transportation Science (IWTCS'13)*, pages 79–84. ACM, 2013.

10 Alexandros Efentakis, Nikos Grivas, George Lamprianidis, Georg Magenschab, and Dieter Pfoser. Isochrones, Traffic and DEMOgraphics. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'13)*, pages 548–551. ACM, 2013.

11 Alexandros Efentakis and Dieter Pfoser. GRASP. Extending Graph Separators for the Single-Source Shortest-Path Problem. In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA'14)*, volume 8737 of *Lecture Notes in Computer Science*, pages 358–370. Springer, 2014.

12 Johann Gamper, Michael Böhlen, and Markus Innerebner. Scalable Computation of Isochrones with Network Expiration. In *Proceedings of the 24th International Conference on Scientific and Statistical Database Management (SSDBM'12)*, volume 7338 of *Lecture Notes in Computer Science*, pages 526–543. Springer, 2012.

13 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

14 Ronald L. Graham. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, 1(4):132–133, 1972.

15 Leonidas J. Guibas and John E. Hershberger. Optimal Shortest Path Queries in a Simple Polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

16 Leonidas J. Guibas, John E. Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons. *Algorithmica*, 2(1):209–233, 1987.

17 Leonidas J. Guibas, John E. Hershberger, Joseph S. B. Mitchell, and J. S. Snoeyink. Approximating Polygons and Subdivisions with Minimum-Link Paths. *International Journal of Computational Geometry & Applications*, 3(4):383–415, 1993.

18 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.

19 Hiroshi Imai and Masao Iri. An Optimal Algorithm for Approximating a Piecewise Linear Function. *Journal of Information Processing*, 9(3):159–162, 1987.

20 Markus Innerebner, Michael Böhlen, and Johann Gamper. ISOGA: A System for Geographical Reachability Analysis. In *Proceedings of the 12th International Conference on Web and Wireless Geographical Information Systems (W2GIS'13)*, volume 7820 of *Lecture Notes in Computer Science*, pages 180–189. Springer, 2013.

21 Irina Kostitsyna, Maarten Löffler, Valentin Polishchuk, and Frank Staals. On the Complexity of Minimum-Link Path Problems. In *Proceedings of the 32nd Annual Symposium on Computational Geometry (SoCG'16)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

22 Sarunas Marciuska and Johann Gamper. Determining Objects within Isochrones in Spatial Network Databases. In *Proceedings of the 14th East European Conference on Advances in Databases and Information Systems (ADBIS'10)*, volume 6295 of *Lecture Notes in Computer Science*, pages 392–405. Springer, 2010.

23 Joseph S. B. Mitchell, Valentin Polishchuk, and Mikko Sysikaski. Minimum-Link Paths Revisited. *Computational Geometry*, 47(6):651–667, 2014.

24 Joseph S. B. Mitchell, Günter Rote, and Gerhard Woeginger. Minimum-Link Paths Among Obstacles in the Plane. *Algorithmica*, 8(1):431–459, 1992.

25 David O'Sullivan, Alastair Morrison, and John Shearer. Using Desktop GIS for the Investigation of Accessibility by Public Transport: An Isochrone Approach. *International Journal of Geographical Information Science*, 14(1):85–104, 2000.

26 Subhash Suri. A Linear Time Algorithm for Minimum Link Paths Inside a Simple Polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986.

27 Robert E. Tarjan. Efficiency of a Good But Not Linear Set Union Algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

28 Cao An Wang. Finding Minimal Nested Polygons. *BIT Numerical Mathematics*, 31(2):230–236, 1991.