Computing Minimum-Link Separating Polygons in Practice*

Moritz Baum† Thomas Bläsius‡ Andreas Gemsa† Ignaz Rutter† Franziska Wegner†

Abstract

We tackle the problem of separating two given sets of polygons by a polygon with minimum number of segments. As the complexity in our specific setting is unknown, we propose heuristics that 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.

1 Introduction

We study a problem motivated by isocontour visualization in road networks, where the goal is to separate the *reachable* subgraph for a given resource limit from the remaining *unreachable* part. Given an embedding of (a planarization of) such a network into the plane, the geometric subproblem is to separate reachable and unreachable *boundaries* by a simple polygon. We consider three objectives for such *range polygons*. They must be exact (i. e., correctly separate the boundaries); they should be of low *complexity* (i. e., have few segments) for an appealing visualization and efficient rendering; the algorithms should be fast in practice, even on large inputs. In this extended abstract, we focus on geometric aspects of this problem; see the full version for omitted details and proofs [1].

Fig. 1 shows an example of a *border region* B, the input of the geometric subproblem. It is defined by two sets R and U of hole-free plane polygons, containing boundaries of the reachable and unreachable part, respectively. We seek a simple polygon with minimum number of links that separates U from R. In general, this is an $\mathcal{NP}$-complete problem [3]. In our case, we have $|R| = 1$ since the reachable part is connected by definition, which, to the best of our knowledge, yields an unresolved open problem [3]. First, consider a border region B with $|R| = |U| = 1$. A polygon of minimum complexity that separates the polygons can be found in $O(n \log n)$ time [7]. However, the algorithm is rather involved and requires computation of several minimum-link paths. We propose a simpler algorithm

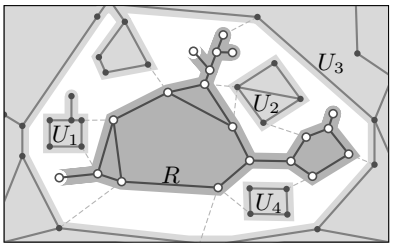

Figure 1: A border region (white area) with a reachable boundary R and an unreachable boundary U with components U_1 to U_4. Shaded areas show reachable (dark gray) and unreachable (light gray) parts.

that uses at most two additional segments, runs in linear time, and requires a single run of a minimum-link path algorithm. It adds an edge e to B that connects both boundaries. In the resulting polygon B', it computes a path with minimum number of segments that connects the two sides of e. The algorithm of Suri [6] computes such a minimum-link path π in linear time. We obtain a separating polygon S by connecting the endpoints of π along e. For practical performance, Section 2 proposes a simpler linear-time algorithm for the key ingredient of our approach, the computation of a minimum-link path. Section 3 then covers the general case of $|U| \geq 1$. Since its complexity is unknown, we focus on heuristic approaches that work well in practice, but do not give guarantees on the complexity of the resulting range polygons. We evaluate our algorithms on realistic input in Section 4.

2 A Practical Minimum-Link Path Algorithm

We address the subproblem of computing a *minimum-link path* (a polygonal path with minimum number of segments) between two edges a and b of a simple polygon P that lies in the interior of P. The algorithm of Suri [6] starts by triangulating the input polygon. In our scenario, we preprocess this step by triangulating all faces of the planar 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 calls to a subroutine computing visibilty polygons are necessary. While this is sufficient to prove linear running time, it seems wasteful from a practical point of view. In the following, we present a simpler linear-time algorithm for computing the windows. It can be seen as a generalization of an algo-

*Partially supported by the EU FP7 under grant agreement no. 609026 (project MOVESMART)

†Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Germany. Email: first.last@kit.edu

‡Karlsruhe Institute of Technology / Hasso Plattner Institute, Germany. Email: thomas.blaesius@hpi.de

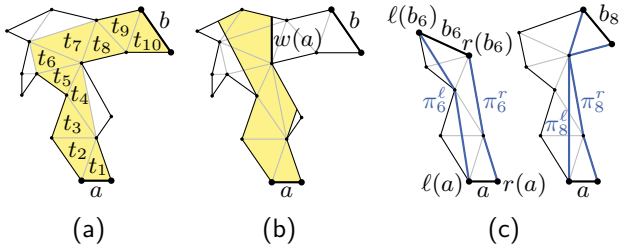

Figure 2: (a) Important triangles. (b) The window $w(a)$ is an edge of $V(a)$ (shaded). (c) The shortest paths (blue) intersect for $i = 8$ but not for $i = 6$.

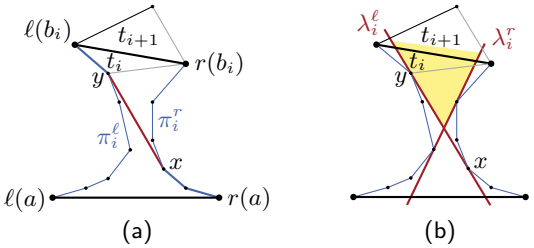

Figure 3: (a) Shortest path from $r(a)$ to $\ell(b_i)$. (b) Visibility lines spanning the (shaded) visibility cone.

rithm for approximating piecewise linear functions [4].

Let G be the (embedded) graph obtained by triangulating P. Let t_a and t_b be the triangles incident to a and b, respectively. The (weak) dual graph of G 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 Fig. 2a. These triangles are *important* and their position in the path is their *index*. The *visibility polygon* $V(a)$ of the edge a in P is the polygon that contains a point p if and only if there is a point q on a such that the line segment pq lies inside P. Let j be the highest index such that the intersection of the triangle t_j with $V(a)$ is not empty. The *window* $w(a)$ is the edge of $V(a)$ that intersects t_j closest (wrt. minimum Euclidean distance) to the edge between t_j and t_{j+1}; see Fig. 2b. Note that $w(a)$ separates the polygon P into two parts. Let P' be the part containing b. A minimum-link path from a to b in P is 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 from $w(a)$.

We describe how to compute the first window. Let G_i be the subgraph of G induced by the triangles $t_1, \ldots, t_i$ and let P_i be the polygon bounding the outer face of G_i. Then P_i has two special edges, namely a and the edge shared by t_i and t_{i+1}, called 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 $r(a)$, $\ell(a)$, $\ell(b_i)$, $r(b_i)$; see Fig. 2c. The *left shortest path* π_i^ℓ is the shortest polygonal path (wrt. Euclidean length) in P_i that connects $\ell(a)$ with $\ell(b_i)$. The *right shortest path* π_i^r is defined symmetrically; see Fig. 2c. One can show that t_{i+1} is (partially) visible from a if and only if the left and right shortest paths in P_i have empty intersection. Moreover, if these paths do not intersect, they are *outward convex*, i. e., π_i^ℓ and π_i^r have only left and right bends, respectively [2]. These two paths together with a and b_i are called *hourglass*. To keep track of parts of t_{i+1} visible from a, we use two visibility lines. To define them, consider the shortest path in the hourglass connecting $r(a)$ with $\ell(b_i)$; see Fig. 3a. It is the concatenation of a prefix of π_i^r, a line segment between vertices x and y, and a suffix of π_i^ℓ. The straight line through x and y is the *left visibility line* denoted by λ_i^ℓ. The *right visibility line* λ_i^r is defined symmetrically. The region between λ_i^ℓ and λ_i^r is the *visibility cone*; see Fig. 3b. A point in t_{i+1} is visible from a if and only if it lies in the visibility cone.

These observations justify the following approach for computing the window $w(a)$. 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. As the left and the right shortest paths did not intersect in P_{i-1}, the triangle t_i is visible from a. Recall that $w(a)$ is the edge of the visibility polygon $V(a)$ that intersects t_i closest to the edge between t_i and t_{i+1}. Thus, $w(a)$ is a segment of one of the two visibility lines. It remains to fill out the details (how to compute the paths and the visibility lines) and describe later steps, when we start at a window instead of an edge.

We start with the details. 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 computed the left and right shortest paths π_{i-1}^ℓ and π_{i-1}^r as well as the visibility lines λ_{i-1}^ℓ and λ_{i-1}^r in a previous step. Without loss of generality, let the three corners of t_i be $\ell(b_{i-1})$, $\ell(b_i)$, and $r(b_i) = r(b_{i-1})$ (as in Fig. 4). There are three possibilities; see Fig. 4. The new vertex $\ell(b_i)$ lies either in the visibility cone spanned by λ_{i-1}^ℓ and λ_{i-1}^r, to the left of λ_{i-1}^ℓ, or to the right of λ_{i-1}^r. We know that 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 Fig. 4e. In this case, we can stop and output the desired window $w(a)$, which is a segment of λ_{i-1}^r; see Fig. 4f. In the other two cases (Fig. 4a and Fig. 4c), we have to compute the new left and right shortest paths π_i^ℓ and π_i^r and the visibility lines λ_i^ℓ and λ_i^r (Fig. 4b and Fig. 4d). Note that the right shortest path and the right visibility line remain unchanged, i.e., $\pi_i^r = \pi_{i-1}^r$ and $\lambda_i^r = \lambda_{i-1}^r$. The new path π_i^ℓ is obtained by concatenating the prefix of π_{i-1}^ℓ with endpoint x and the segment from x to $\ell(b_i)$, where x is the latest vertex on π_{i-1}^ℓ such that the result is outward convex. To get the new left visibility line λ_i^ℓ, we distinguish the two remaining cases. If $\ell(b_i)$ lies to the left of λ_{i-1}^ℓ (Fig. 4c), we obtain $\lambda_i^\ell = \lambda_{i-1}^\ell$; see Fig. 4d. If $\ell(b_i)$ lies in the visibility cone (Fig. 4a), the

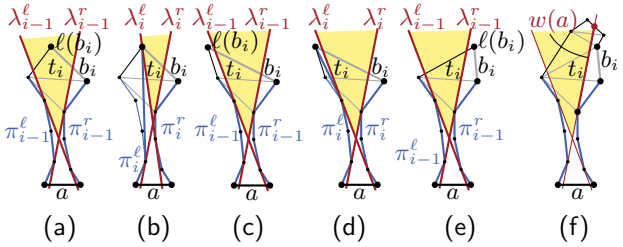

Figure 4: The three cases for the position of $\ell(b_i)$ and the updated shortest paths and visibility lines.

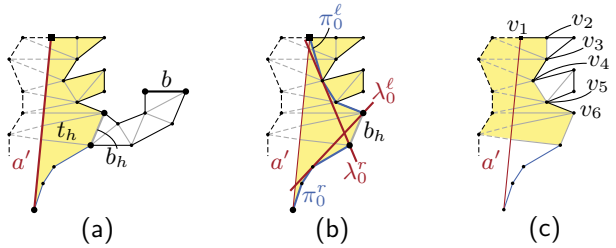

Figure 5: Initial steps for computing the next window, when starting at the previous window a'.

visibility line changes. Let x be the latest vertex on π_i^r 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. Then λ_i^ℓ is the line through x and $\ell(b_i)$; see Fig. 4b. One can show that the point x where λ_i^ℓ intersects π_i^r moves forward along π_i^r during the algorithm (which is relevant for the running time).

Lemma 1 *Let t_h be the triangle with the highest index that is visible from a. Then our algorithm computes the first window $w(a)$ in $O(h)$ time.*

The window $a' = w(a)$ separates P into two smaller polygons. Let P' be the part including the edge b. 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 retriangulate P'. More precisely, let t_h be the triangle with highest index that is visible from a; see Fig. 5a. Then b_h separates P' into an initial part P'_0 and the rest (having b on its boundary). The latter part is properly triangulated, but P'_0 is not. While we could retriangulate P'_0, this would require an efficient subroutine for triangulation and a dynamic data structure. Instead, we propose a much simpler method for computing the next window.

The idea is to compute shortest paths in P'_0 from $\ell(a')$ to $\ell(b_h)$ and from $r(a')$ to $r(b_h)$; see Fig. 5b. We denote these paths by π_0^ℓ and π_0^r, respectively. Moreover, we 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, consider the polygon induced by the triangles intersected by a'; see Fig. 5c. Let $v_1, \ldots, v_g$ be the sequence of vertices on the outer face of this polygon (in clockwise direction) from $\ell(a') = v_1$ to $\ell(b_h) = v_g$. To obtain π_0^ℓ, we start with an empty path and iteratively append the next vertex from this sequence while maintaining the path's outward convexity by successively removing the second to last vertex if necessary. It remains to compute the initial visibility lines λ_0^ℓ and λ_0^r. Note that the whole edge b_h is visible from a', since a' intersects the triangle t_h. It follows that λ_0^ℓ is the line that goes through $\ell(b_h)$ and through the unique vertex on π_0^r such that λ_0^ℓ is tangent to π_0^r; see Fig. 5b. The same holds for the right visibility line. With these insights, it is not hard to compute the paths π_0^ℓ and π_0^r and 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 first edge a by a straight line [6]. Lemma 1 and our considerations concerning initial paths imply the following theorem.

Theorem 2 *Given two edges a and b of a simple polygon P, our algorithm computes a minimum-link path from a to b contained in P in linear time.*

3 Heuristics for the General Case

We outline heuristics for the case of $|U| \geq 1$. Fig. 6 sketches results in a small example. First, `RP-RC` (*range polygon*, extracted *reachable components*) simply returns the reachable boundary R. This approach is similar to previous algorithms for isochrones [5]. Second, `RP-TS` (*triangular separators*) uses the triangulation to separate B along edges for which both endpoints are in R. The modified instances consist of single unreachable components that are separated by the algorithm from Section 2. Third, `RP-CU` (*connecting unreachable* components) inserts new edges that connect the components of U to create an instance with $|U| = 1$. Finally, `RP-SI` (*self-intersecting* polygons) modifies the approach described in Section 2 to compute a possibly self-intersecting minimum-link path separating R and U. To obtain the range polygon, it is rearranged at intersections.

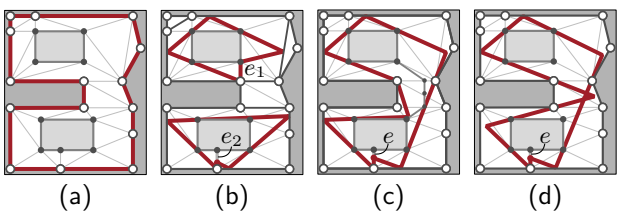

Figure 6: Results (red) of `RP-RC` (a), `RP-TS` (b), `RP-CU` (c), and `RP-SI` (d), starting at indicated edges.

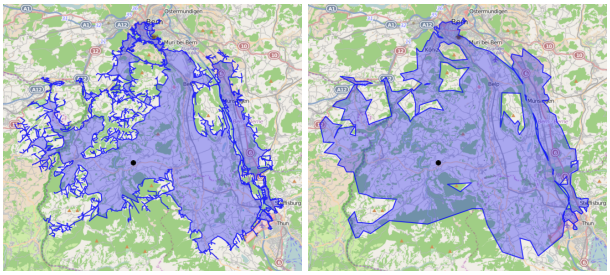

Figure 7: Real-world example of isocontours showing the range of an electric vehicles at the black disk (near Bern, Switzerland) and a state of charge of 2 kWh.

4 Evaluation

We evaluate our approaches (implemented in C++) on a graph representing the road network of Europe, with 22 million vertices and 52 million edges. Fig. 7 shows isocontours visualizing the range of an electric vehicle. The result of `RP-RC` (left) has more than 10 000 segments, even in this medium-range example. The result of `RP-CU` (right) uses much fewer (416) segments to represent the same isocontour. Note that the isocontour contains holes, due to unreachable high-ground areas. Hence, it has several border regions.

For our analysis, we focus on large ranges (roughly 500 km with 85 kWh batteries). Table 1 shows results of all heuristics from Section 3, averaged for 1 000 queries from sources picked uniformly at random. Timings include extraction of the border region from the input graph, which is part of the work required in our scenario. For `RP-SI`, we report figures for unprocessed polygons with self-intersections. Running times are well below 30 ms for all approaches. The simpler heuristics, `RP-RC` and `RP-TS`, are faster by a factor of 2 to 3. However, polygons computed by `RP-RC` have a much higher complexity (by about a factor of 50). Results of `RP-TS` have low complexity, but the triangular separation increases the number of components significantly (all other approaches have the minimum number of components, i. e., the number of border regions). For `RP-CU` and `RP-SI`, the additional effort pays off, as they keep complexity close to the optimum (off by at most 6 % according to a lower bound induced by the results of `RP-SI`).

Table 1: Results for isocontours. We show the number of components of the result (Cp.), complexity (Seg.), self-intersections (Int.), and running time in ms.

Algorithm	Cp.	Seg.	Int.	Time
`RP-RC`	131	92 554	—	9.46
`RP-TS`	219	1 973	—	7.78
`RP-CU`	131	1 820	—	25.11
`RP-SI`	131	1 781	15.06	22.25

Table 2: Minimum-link path algorithm performance. We report input complexity ($|P|$), visited triangles (v. Tr.), links in the result (Seg.), and time in ms.

Scenario	$\|P\|$	v. Tr.	Seg.	Time
EV, 16 kWh	134 049	9 334	416	0.72
Iso, 60 min	135 112	11 965	701	1.03
EV, 85 kWh	357 335	33 030	1 329	3.14
Iso, 500 min	637 224	69 398	3 204	6.57

We evaluate the minimum-link path algorithm from Section 2 in four scenarios (ranges for 16 kWh and 85 kWh batteries and isochrones for 60 and 500 minutes). For each of 1 000 random queries, we modified the largest border region such that $|U| = 1$ (using `RP-CU`). Then, we added an edge connecting the two components and computed a minimum-link path between its two sides. Recall that triangulation of the input is part of preprocessing, hence it is not reported in the table. As Table 2 shows, the running time increases with input complexity. However, our algorithm runs in less than 10 milliseconds on average in all cases. Isochrone scenarios are slightly harder due to the different shape of the border regions.

Overall, we see that our approaches are suitable for interactive applications on inputs of continental scale.

Acknowledgments. We thank Roman Prutkin for interesting discussions.

References

[1] M. Baum, T. Bläsius, A. Gemsa, I. Rutter, and F. Wegner. Scalable Isocontour Visualization in Road Networks via Minimum-Link Paths. Technical Report abs/1602.01777, ArXiv e-prints, 2016.

[2] L. J. Guibas, J. E. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons. *Algorithmica*, 2(1-4):209–233, 1987.

[3] L. J. Guibas, J. E. Hershberger, J. 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.

[4] H. Imai and M. Iri. An Optimal Algorithm for Approximating a Piecewise Linear Function. *Journal of Information Processing*, 9(3):159–162, 1987.

[5] S. Marciuska and J. Gamper. Determining Objects within Isochrones in Spatial Network Databases. In *Advances in Databases and Information Systems*, volume 6295 of *LNCS*, pages 392–405. Springer, 2010.

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

[7] C. A. Wang. Finding Minimal Nested Polygons. *BIT Numerical Mathematics*, 31(2):230–236, 1991.