A Linear-Time Algorithm to Compute the Optimal Bridge Connecting Two Disjoint Convex Polygons

Leonidas Palios
Department of Computer Science
University of Ioannina, Ioannina, Greece
palios@cs.uoi.gr

Abstract

We present a linear-time algorithm to solve the problem of computing the line segment connecting two disjoint convex polygons P and Q so that the maximum of the lengths of the paths from a point of P through the line segment to a point of Q is minimized. The algorithm relies on the notion of the "far-wedges" of a polygon, which we introduce. Additionally, we present a simple linear-time algorithm to compute the intersection points of the boundary of a convex polygon with the farthest-neighbor Voronoi diagram of its vertices, without computing the Voronoi diagram.

1 Introduction

Given two convex polygons P and Q that do not intersect, we are interested in finding the line segment connecting a point p of P and a point q of Q which minimizes the quantity

$$\max_{p' \in P}\{\, d(p', p) \,\} \;+\; d(p, q) \;+\; \max_{q' \in Q}\{\, d(q, q') \,\}, \tag{1}$$

where $d(a, b)$ denotes the Euclidean distance between points a and b. The line segment pq is the *optimal bridge* between P and Q which guarantees that the maximum length of any path from a point of P, through the bridge, to a point of Q is minimized.

The problem has been considered by Cai, Xu, and Zhu, who described an $O(n^2 \log n)$-time algorithm to compute the optimal bridge [3], where n is the total number of vertices of the two convex polygons P and Q. They also show that the minimum distance line segment between P and Q provides a 2-approximate solution which can be computed in linear time. Recently, a linear-time algorithm for this problem has been proposed by Kim and Shin [4] who have also addressed the related problems of bridging a convex and a simple polygon, and two simple polygons.

In this abstract, we outline a simple linear-time algorithm that computes the optimal bridge between two given disjoint convex polygons. Unlike Kim and Shin's algorithm which relies on the total monotonicity of a table of distances, our algorithm is geometric in nature and takes advantage of the properties of the *far-wedges* of a convex polygon, which we introduce. Additionally, we outline a simple linear-time algorithm to compute the intersection points of the boundary of a convex polygon with the farthest-neighbor Voronoi diagram of its vertices without computing the Voronoi diagram.

2 Theoretical Framework

We assume, without loss of generality, that the two given disjoint convex polygons P and Q are separated by a vertical line and P lies on its left and Q on its right.

Let us define the *in-hull chain* of a polygon P as the closure of the part of the boundary of P which lies in the interior of the convex hull of $P \cup Q$; similarly, we define the in-hull chain of Q. In [3], it is observed that:

Observation 1. ([3]) *The optimal bridge connecting two disjoint convex polygons has one endpoint on the in-hull chain of the first polygon and the other endpoint on the in-hull chain of the second polygon.*

The quantity $\max_{p' \in P}\{\, d(p,p')\,\}$ is equal to the distance of p to its *farthest neighbor* in P. Since the optimal bridge may not necessarily be attached to a vertex of P or a vertex of Q, we need to determine the farthest neighbor of every boundary point of P and similarly for Q. We can do that by computing the partition of the boundary of each of these polygons into sets of points with the same farthest neighbor. It is very interesting to note that:

Observation 2. *Suppose that we partition the boundary ϑP of a convex polygon P into sets of points with the same farthest neighbor in P. Then, if we consider these sets in counterclockwise (clockwise, respectively) order around ϑP, their farthest neighbors form a subsequence of P's vertices traversed in counterclockwise (clockwise, respectively) order.*

The observation implies that these sets are connected. In fact, each such set is the intersection of the boundary of the polygon with a region of the farthest-neighbor Voronoi diagram. Let these partition sets be $s_1 ... s_2$, $s_2 ... s_3$, $\ldots$, $s_k ... s_1$ in counterclockwise order around the boundary of P (starting at an arbitrary set). Moreover, let the corresponding farthest neighbors be v_{j_1}, v_{j_2}, $\ldots$, v_{j_k}; these are vertices of P and they form a subsequence of the counterclockwise order of P's vertices around the boundary starting at v_{j_1}. Each boundary set $s_i ... s_{i+1}$ is associated with the vertex v_{j_i}; this pair defines two lines, one determined by the points v_{j_i} and s_i, the other determined by v_{j_i} and s_{i+1}. These two lines define four wedges, one of which contains in its closure the boundary set $s_i ... s_{i+1}$; this is the *far-wedge* with *apex* v_{j_i} and *boundary chain* $s_i ... s_{i+1}$. Figure 1 depicts a convex polygon, the Voronoi diagram of its vertices and its far-wedges.

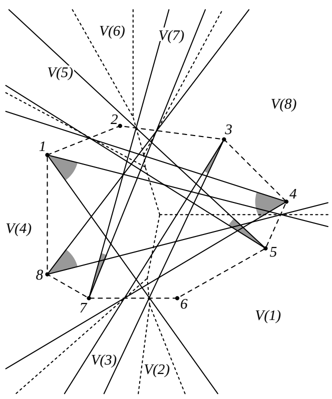

Figure 1

Lemma 1. ([6]) *No two far-wedges of a convex polygon P intersect in P's complement.*

Moreover, any two far-wedges whose boundary chains share an endpoint are separated in the complement of P by a wedge with apex the common endpoint (see Figure 1).

Our approach is based on reducing the computation of the optimal bridge connecting the convex polygons P and Q into computing the optimal bridges for all necessary pairs of a far-wedge of P and a far-wedge of Q. In light of Observation 1, we need only consider the far-wedges of the polygons P and Q whose boundary chains intersect the in-hull chain of P and of Q respectively; these are the *useful* far-wedges of P and Q respectively. The useful far-wedges can be ordered with respect to the order of their boundary chains in the in-hull chain of the corresponding polygon.

Consider all the useful far-wedges of the polygon Q. The polygon P intersects some of them and intersects some of the wedges between consecutive far-wedges, and is thus partitioned into slices. (Note that P may entirely belong to a far-wedge or to a wedge between consecutive far-wedges, so that it may consist of a single slice.) If the topmost slice belongs to a far-wedge with apex u, then we draw the upper tangent line from u to P. If it lies between far-wedges, then it lies in a wedge with apex a point, say, w, on the boundary of Q; then, we draw the upper tangent from w to P. Let s be the point at which the tangent touches P. We work in a similar fashion with the lowermost slice defining a lower tangent from a point of the boundary of Q to P and let t be the point of tangency at P. Then, the *reduced chain* of P is the part of the boundary clockwise from s to t. Since the tangents which define the endpoints of the reduced chain are drawn from points of Q, the reduced chain is a subset of the in-hull chain of P. It can be shown that:

Lemma 2. ([6]) *The optimal bridge is attached to the convex polygon P at a point of P's reduced chain.*

Let F be a far-wedge (with apex v and boundary chain $a...b$) of the polygon P, and let $S = [W_1^Q, \ldots, W_k^Q]$ be the ordered set of the useful far-wedges of the polygon Q from the upper tangent point counterclockwise to the lower tangent point. We partition the set S into three subsets $A = [W_1^Q, \ldots, W_{i-1}^Q]$, $B = [W_i^Q, \ldots, W_j^Q]$, and $C = [W_{j+1}^Q, \ldots, W_k^Q]$, where B is the set of far-wedges of Q which intersect the boundary chain $a...b$ of F. Note that any (but not all) of these three subsets may be empty. We can show that:

Lemma 3. ([6]) *Let P and Q be two disjoint convex polygons and let pq be the optimal bridge connecting them, where $p \in P$ and $q \in Q$. Moreover, let F be a far-wedge of P whose boundary chain intersects the reduced chain of P, and let $A = [W_1^Q, \ldots, W_{i-1}^Q]$, $B = [W_i^Q, \ldots, W_j^Q]$, and $C = [W_{j+1}^Q, \ldots, W_k^Q]$ be the partition of the set S with respect to F as described above. If the optimal bridge pq has its endpoint p on the boundary chain of F, then it has to be one of the (partial) optimal bridges connecting F to one of the far-wedges W_{i-1}^Q, W_i^Q, $\ldots$, W_{j+1}^Q.*

Lemmata 1 and 3 help us prove that the number of pairs of a far-wedge of P and a far-wedge of Q that we need to consider in order to compute the optimal bridge is linear in the total number of vertices of the polygons P and Q.

3 The Algorithm

The algorithm is outlined in Algorithm 1. Note that Step 5 of the algorithm implements Lemma 3; thus, the bridge computed is the optimal bridge sought. The computation of the optimal bridge for a pair of far-wedges is based on the five main cases shown in Figure 2, and can be done in constant time per pair. This along with Lemmata 1 and 3 and because g never moves backwards in L'_Q imply that Step 5 takes time linear in the total number of vertices of the polygons P and Q. Steps 2, 3 and 4 also take linear time and so does Step 1, which we describe in more detail below.

Computing the far-wedges of a convex polygon. We need to compute the points of intersection of the boundary of the convex polygon with the farthest-neighbor Voronoi diagram of its vertices. This can be done by explicitly computing the Voronoi diagram, which can be found in linear time [1]. Next, we give an alternative way to compute the intersection points in linear time without computing the Voronoi diagram.

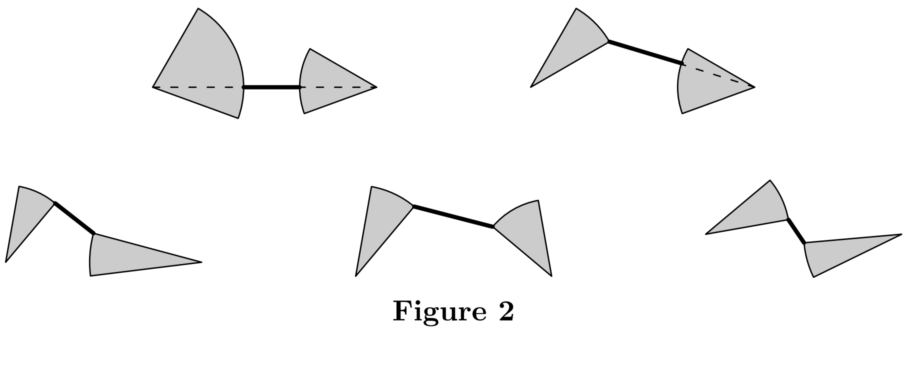

Figure 2

1. We compute the far-wedges of the two polygons P and Q.
2. We form a list L_P of the useful far-wedges of P ordered in clockwise order; similarly, we form a list L_Q of the useful far-wedges of Q ordered in counterclockwise order.
3. Starting at P's vertex with the largest x-coordinate and by moving up and down, we determine the reduced chain of P.
4. Let L'_P be the sublist of the far-wedges in L_P whose boundary chains intersect the reduced chain of P. Let L'_Q be the sublist of the far-wedges in L_Q which intersect the reduced chain of P; we clip the first and last element of L'_Q about the endpoints of the reduced chain of P and we augment L'_Q by including one far-wedge before and one after (if there exist).
5. $g \leftarrow$ first far-wedge in L'_Q;
   `for` each far-wedge f (with apex v and boundary chain $a...b$) in L'_P in order
   - `while` $g \neq$ one of the last 2 elements of L'_Q `&&` $a...b$ is below $next(g)$ in L'_Q
     - $g \leftarrow next(g)$ in L'_Q;
   - $t \leftarrow g$;
   - we compute optimal bridge of (f, t) and corresponding maximum path length ℓ;
   - `if` $\ell <$ minimum value min_ℓ of maximum path length computed so far
     - we update the overall optimal bridge and min_ℓ;
   - `if` $t \neq$ last element of L'_Q
     - $t \leftarrow next(t)$ in L'_Q;
     - `while` $t \neq$ last element of L'_Q `&&` f's boundary chain does not lie above t
       - we compute opt. bridge of (f, t) and corresponding max path length ℓ;
       - `if` $\ell <$ minimum value min_ℓ of maximum path length computed so far
         - we update the overall optimal bridge and min_ℓ;
       - $t \leftarrow next(t)$ in L'_Q;
     - we compute optimal bridge of (f, t) and corresponding max path length ℓ;
     - `if` $\ell <$ minimum value min_ℓ of maximum path length computed so far
       - we update the overall optimal bridge and min_ℓ;

Algorithm 1: Computing the optimal bridge connecting two disjoint convex polygons.

Algorithm for Computing the Far-wedges of a Convex Polygon

1. We compute the all-pairs farthest neighbors of all the vertices of the given convex polygon; this can be done in time linear in the size of the polygon by using the algorithm of Aggarwal et al [2].
2. For each edge uv of the polygon, we check the farthest neighbors of its endpoints u and v. If the vertices u and v have the same farthest neighbor, then the farthest neighbor of every point of the edge is the same as the (common) farthest neighbor of u and v. Otherwise, we refine the edge by partitioning it into subsegments each of which has a different farthest neighbor. This is done as follows. If the farthest neighbors of u and v are adjacent vertices of the polygon, then the edge uv is partitioned into two subsegments about its point of intersection with the perpendicular bisector of the line segment connecting the farthest neighbors of u and v (note that the perpendicular bisector intersects the edge uv). Suppose now that the farthest neighbors of u and v differ and they are not adjacent vertices of the polygon. We may assume without loss of generality that u is immediately before v in a clockwise traversal of the vertices around the boundary of the polygon. In order to compute the partition into subsegments and the associated farthest neighbors, we use a stack

to store the current list of farthest neighbors. Initially, we push in the stack the farthest neighbor of u and the polygon vertex adjacent to it in clockwise order around the polygon boundary. Then, we process each of the remaining vertices up to the farthest neighbor of v in the following way. We consider the line segment defined by the current vertex and the vertex at the top of the stack and the line segment defined by the two vertices at the top two positions of the stack. If the perpendicular bisectors of these two segments intersect on the boundary or outside the polygon, then the vertex at the top of the stack is popped, and the perpendicular bisector test is repeated for the line segment defined by the current vertex and the vertex which is now at the top of the stack and the line segment defined by the two vertices at the top of the stack. The process is repeated until either the stack ends up containing only one element or the bisectors intersect at a point in the interior of the polygon. Then, the current vertex is pushed in the stack; its processing is complete and we continue with the processing of the next vertex.

After all the vertices from the farthest neighbor of u clockwise to the farthest neighbor of v have been processed, the stack contains the farthest neighbors of the subsegments of the edge uv in order from u to v. The subsegments are determined by the points of intersection of the edge uv and the perpendicular bisectors of the line segments defined by pairs of vertices in consecutive positions of the stack.

3. The final partition of the boundary of the polygon is computed by merging edges and edge subsegments which are adjacent and are associated with the same farthest neighbor. The far-wedges are determined by each element of the partition and the associated farthest neighbor.

4 Extensions

The optimal bridge problem can be generalized in several ways. First, the convexity restriction on one or both polygons can be relaxed: compute the optimal bridge between a convex and a simple polygon, or between two disjoint simple polygons. Kim and Shin provided an $O((n+m)\log(n+m))$-time and an $O(n\,m+(n+m)\log(n+m))$-time algorithm respectively for these problems on two polygons of sizes n and m. Can these time complexities be improved? Additionally, the problem can be extended into three-dimensional space: compute the optimal bridge connecting two disjoint convex polyhedra. How fast can this be solved? What if we relax the convexity restriction on one or both polyhedra?

5 References

1. A. Aggarwal, L.J. Guibas, J. Saxe, and P.W. Shor, "A linear-time algorithm for computing the Voronoi diagram of a convex polygon," *Discrete and Computational Geometry* 4(6) (1989), 591–604.
2. A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, and R. Wilber, "Geometric applications of a matrix-searching algorithm," *Algorithmica* 2(2) (1987), 195–208.
3. L. Cai, Y. Xu, and B. Zhu, "Computing the optimal bridge between two convex polygons," *Information Processing Letters* 69 (1999), 127–130.
4. S.K. Kim and C.S. Shin, "Computing the optimal bridge between two polygons," *Research Report HKUST-TCSC-1999-14*, Hong-Kong University, 1999.
5. D.T. Lee and F.P. Preparata, "The all-nearest neighbor problem for convex polygons," *Information Processing Letters* 7(4) (1978), 189–192.
6. L. Palios, "A linear-time algorithm to compute the optimal bridge connecting two disjoint convex polygons," *Technical Report TR-22-2000*, Dept. of Computer Science, University of Ioannina, 2000.