Approximation Algorithms for Finding the Optimal Bridge Connecting Two Simple Polygons

Amit M. Bhosle *† Teofilo F. Gonzalez‡

Abstract

Given two simple polygons P and Q we define the weight of a bridge (p, q)*, with $p \in \delta(P)$ and $q \in \delta(Q)$, where $\delta()$ defines the boundary of the polygon, between the two polygons as $gd(p, P)+d(p, q)+gd(q, Q)$ where $d(p, q)$ is the Euclidean distance between the points p and q, and $gd(a, A)$ is the geodesic distance between a and its geodesic farthest neighbor on A. An optimal bridge (of minimum weight) can be found in $O(n \log^3 n)$ time as described in [13]. We present an approximation scheme that, given any positive integer k, constructs a bridge with objective function value within $(1 + \frac{2}{k+1})$ of optimal in $O(kn \log kn)$ time, thus solving an open problem stated in [10]. We also present a fully polynomial time approximation scheme that for any $\epsilon > 0$ generates a bridge with objective function within ϵ of optimal in $O(kn \log kn)$ time, where $k = 2 * \lceil \frac{1}{\log(1+\epsilon)} \rceil$. In contrast to the exact algorithm given in [13], our algorithm does not use complicated data structures and is amenable to efficient implementations. We also show that the claim of [13] that if (p, q) is the optimal bridge, then $gd(p, q) = d(p, q)$ is incorrect.*

1 Introduction

Given two disjoint simple polygons P and Q, a line segment that connects the two polygons is called a *bridge*. We may think of each polygon to be a community or region in which wireless communications may take place. Furthermore, the regions or communities are separated by mountains or obstacles which prevent wireless communications between the two communities. Our problem is to introduce a bridge or wire to allow for inter-community communication. The objective function is to find a bridge such that the Euclidean length of the path that every message travels through the bridge is least possible. Formally, for points $p \in \delta(P)$ and $q \in \delta(Q)$, where $\delta()$ defines the boundary of the polygon, we define the *weight* of the bridge (p, q) as

$$max_{p' \in \delta(P)}\{gd(p, p')\} + d(p, q) + max_{q' \in \delta(Q)}\{gd(q, q')\},$$

*YASU Technologies Pvt. Ltd., Begumpet, Hyderabad - 500 016, India. Email: bhosle@cs.ucsb.edu.

†This work was done while this author was at the Department of Computer Science, UC Santa Barbara.

‡Department of Computer Science, University of California, Santa Barbara, CA - 93106. Email: teo@cs.ucsb.edu.

where $d(x, y)$ denotes the Euclidean distance from x to y, and $gd(x, y)$ is the geodesic distance between x and y in their corresponding polygon (i.e., the shortest distance between x and y without leaving the polygon where they are located). A pair (p, q) that minimizes the above expression is called an *optimal bridge*. Contrary to Tan's claim [13], the end points of an optimal bridge are not necessarily *mutually visible* when the polygons are not convex. Figure [1] depicts a problem instance such that the bridge defined by the solid thick line has total cost $\sim 2x + 3y$, but any bridge between any two visible points (one in P and one in Q) has cost at least $3x$. Since $x >> y >> z$, it follows that this problem instance does not have an optimal bridge in which its two endpoints are mutually visible. Another way to define the bridge problem is replace $d(p, q)$ by $gd(p, q)$. We call this problem the *all-geodesic bridge* problem or simply the *ag-bridge* problem. Tan's algorithm is not set up to solve the ag-bridge problem. As shown in Figure [1] the endpoints in an optimal ag-bridge (dashed thick line) are not necessarily *mutually visible* when the polygons are not convex.

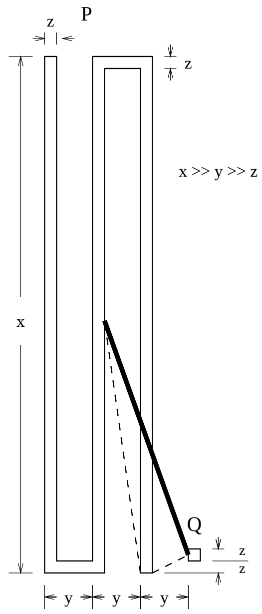

Figure 1: Problem instance that does not have an optimal bridge defined by two visible points

Note that an optimal bridge may touch a vertex of P or Q. Also, the geodesic farthest neighbors p' and q' of the end points p and q of the bridge respectively, are vertices of the corresponding polygons; otherwise, the solution is not optimal.

1.1 Related Work

The problem of finding the optimal bridge connecting two *convex* polygons has been solved to optimality. The problem was first studied by Cai, Xu and Zhu [6] who presented an $O(n^2 \log n)$ time algorithm to construct an optimal bridge between any two given disjoint *convex* polygons. Because of similarity to many geometric problems (e.g. diameter problems, minimum separations problems, and minimum spanning tree problems), much progress was made immediately after the problem was posed. For the convex polygon case, different optimal (linear time) algorithms have been developed independently by Bhattacharya, et al. [5], Tan [12] and Kim et. al [10].

The more general version of the problem, which deals with the case when the input polygons are not necessarily convex, has however resisted linear time algorithms. Bhattacharya, et al. [5] presented a quadratic time algorithm for the problem. Currently the fastest algorithm for the *exact* solution of the problem is by Tan [13], which runs in $O(n \log^3 n)$ time. This algorithm is quite complex and it makes substantial use of a hierarchical structure that consists of segment trees, range trees and persistent search trees, and a structure that supports dynamic ray shooting and shortest path queries. A restricted version, where the input polygons are simple but *rectilinear*, can be solved in linear time using Wang's algorithm [15]. The high-dimensional version of the problem has been studied in [12, 14].

A linear-time 2-approximation algorithm for convex polygons was presented in [6]. Kim and Shin [10] showed that this approximation strategy also works for the case when the polygons are not convex. The time complexity of this approximation scheme is that of finding the *least* distance between (the boundaries of) the two polygons, which can be done in linear time using the algorithm by Amato [1]. They raise the question as to whether or not a better approximation algorithm exists. In this paper we answer this question in the positive.

1.2 Main Results

Our main results include an approximation scheme that given any positive integer k constructs a bridge with objective function value within $1 + \frac{2}{k+1}$ of optimal in $O(kn \log kn)$ time. The scheme is presented in Section 2. In Section 3 we present a fully polynomial time approximation scheme that given any $\epsilon > 0$ generates a bridge with objective function within ϵ of optimal in $O(kn \log kn)$ time, where $k = 2 * \lceil \frac{1}{\log(1+\epsilon)} \rceil$. In Section 4 we discuss the changes we need to make in order to apply our results to the ag-bridge problem. Note that there is no known polynomial time algorithm to find an optimal ag-bridge between two polygons.

2 Approximation Schemes

In this section we present our approximation scheme for the optimal bridge problem. The idea is to introduce $k - 1$ artificial vertices along each edge of the input polygons and then find an optimal *vertex* bridge. An optimal vertex bridge is a bridge whose one end point is a vertex of P and the other is a vertex of Q. An $O(n^2)$ time algorithm for this restricted version of the problem for *convex* polygons was given in [6], and Kim [9] developed a linear time one. Kim [9] also developed an $O(n \log n)$ for the case when one of the two polygons is convex, and an $O(n^2)$ time algorithm for the case when neither of the polygons is convex. We present an efficient algorithm that solves this problem in $O(n \log n)$ time by reducing it to the *additively weighted nearest point* Voronoi diagram [3, 4] problem, which can be solved in $O(n \log n)$ time.

2.1 Algorithm for the Optimal Vertex Bridge Problem

In this section, we present an algorithm for finding an optimal vertex bridge, which is a bridge whose one end point is a vertex of P and the other is a vertex of Q. Note that the end points of an optimal vertex bridge may not be *mutually visible*. An $O(n^2)$ algorithm for this restricted version of the problem was given in [6] for convex polygons, and in [9] for simple polygons. We present an efficient algorithm that solves this problem by using the *additively weighted nearest point* Voronoi diagram [3, 4] problem, which can be built in $O(n \log n)$ time. The technique yields an $O(n \log n)$ time algorithm for the optimal vertex bridge problem. In an additively weighted nearest point Voronoi diagram, each input point s has an associated weight w_s. If V is the set of input points, for any given point p in the plane, a weighted-nearest query returns a point $v \in V$ such that

$$d(p, v) + w_v = MIN_{u \in V}\{d(p, u) + w_u\}$$

The additively weighted nearest point Voronoi diagram can be constructed in $O(n \log n)$ time given a set of n points and their associated weights. Furthermore, any query can be answered in $O(\log n)$ time. This data structure was used in [6] to obtain an $O(n^2 \log n)$ time algorithm for determining the minimum weight bridge connecting two convex polygons. Using this Voronoi diagram data structure, an optimal vertex bridge can be easily computed by procedure `OVB`.

To compute the weights associated with each point in the above Voronoi diagram, we compute the *farthest-geodesic-neighbor* in P for each vertex of P, and in Q for each vertex of Q. This particular problem can be solved in $O(n \log n)$ time using Suri's algorithm [11].

`Procedure OVB(P,Q): simple polygons P, Q`

Compute the geodesic-farthest-neighbor of each vertex in its polygon using Suri's algorithm [11].

For each vertex q of Q, set $w_q = gd(q, r)$ where $r \in Q$ is the geodesic-farthest-neighbor of q.

Using the w_q values computed above, build an additively weighted nearest point Voronoi diagram [3, 4] for the vertices of Q;

Using the above constructed data structure, for each vertex $p \in P$, find the vertex $q \in Q$ such that $gd(p, Q) = d(p, q) + w_q = MIN_{q' \in Q}\{d(p, q') + w_{q'}\}$;

The weight of the optimal bridge with one end point as $p \in P$ is $gd(p, P) + gd(p, Q)$, where $gd(p, P)$ is the geodesic distance between p and its geodesic-farthest neighbor in P.

Choose the vertex of P which supports the bridge of minimal weight;

`End Procedure OVB`

Theorem 2.1 *Given two simple polygons P and Q, algorithm* `OVB` *finds an optimal vertex bridge between P and Q in $O(n \log n)$ time.*

Proof: It is clear that the distance from every vertex $q \in Q$ to its geodesic-farthest point in Q is computed and stored correctly in w_q by Suri's algorithm [11]. From the additively weighted nearest point Voronoi diagram our procedure computes for every vertex p, $gd(p, Q)$, which is defined as the minimum value of $\{d(p, q) + w_q\}$ for any vertex $q \in Q$. In other words, that is the best possible vertex bridge with an endpoint at vertex p. In the last step one just finds the vertex p with least $gd(p, P) + gd(p, Q)$ which is the optimal vertex bridge.

Suri's algorithm for computing the geodesic farthest neighbors for all the points of a simple polygon takes $O(n \log n)$ time. Building the farthest point Voronoi diagrams take time $O(n \log n)$ where $n = MAX(|P|, |Q|)$. There are at most n such queries in the procedure OVB, thus taking a total time bounded by $O(n \log n)$. The queries on the additively weighted nearest point Voronoi diagram also take $O(\log n)$ time each. Thus, the overall time complexity of the algorithm OVB is $O(n \log n)$ time. □

2.2 Artificial Vertex Approximation Scheme

Our artificial vertex approximation scheme is a simple extension of the algorithm OVB presented above. Given any integer $k \geq 1$, we partition each edge of the given polygons in k equal intervals by introducing artificial vertices. We then invoke the algorithm OVB with the input polygons having as vertices the original vertices plus the artificial ones. Thus, the input polygons now have $O(kn)$ vertices each. Thus, the algorithm OVB takes $O(kn \log kn)$ time to compute a bridge which has as end points either the original vertices of the polygons or the newly introduced artificial vertices.

`Procedure AV_OVB(P,Q,k): simple P, Q, int` $k > 0$
  Introduce $k-1$ uniformly spaced points on each edge of the input polygons.
  Call these polygons with the newly introduced artificial vertices P' and Q', respectively.
  Output the result returned by $OVB(P', Q')$
`End Procedure AV_OVB`

Let us now establish our approximation bound for the Artificial Vertex Approximation Scheme.

Theorem 2.2 *Algorithm* `AV_OVB(P,Q,k)` *generates a bridge such that* $\hat{f}/f^* \leq 1 + \frac{2}{k}$ *in* $O(kn \log kn)$*, where* f^* *is the weight of the optimal bridge and* $\hat{f}$ *is the weight of the bridge computed by the procedure* `AV_OVB()`*.*

Proof: Assume (p, q) is the optimal bridge, with $p \in \delta(P)$ and $q \in \delta(Q)$ (see Figure 2). Let the optimal bridge have the point p lying on the edge (a, b) of P and q on the edge (c, d) of Q. Also, let u be the geodesic farthest neighbor of p, with $gd(p, u) = x$. Thus, the total weight of the bridge is $w(p, q) = d(p, q) + gd(p, u) + gd(q, Q) = d(p, q) + x + gd(q, Q)$.

Let us consider first the case when $k = 1$, i.e., there are no artificial vertices. The algorithm AV_OVB investigates the bridges through the points a and b of the polygon. We shall compare the weights of the bridges through these vertices to that of the optimal bridge.

Without loss of generality, assume that p is no farther from a than from b and q is no farther from c than from d. We investigate the bridge using a. The weight of the bridge (a, c) is $w(a, c) = gd(a, P) + d(a, c) + gd(c, Q)$.
Since $gd(p, P) = gd(p, u) = x$, we know that

$$d(p, a) = gd(p, a) \leq x$$

$$d(p, b) = gd(p, b) \leq x$$

$$\Rightarrow d(a, b) \leq 2x$$

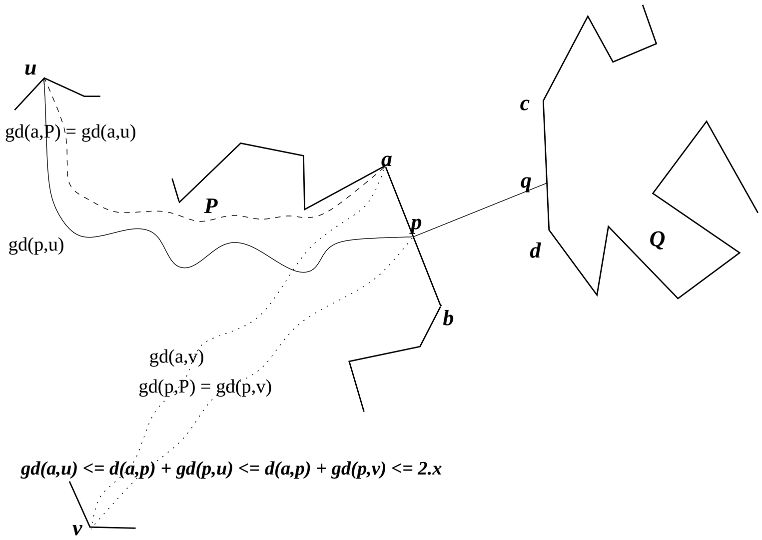

Figure 2: Case when $k = 1$ and p is on the line ab.

If $gd(a, P) = gd(a, v)$, then using triangle inequality, we have

$$gd(a, P) = gd(a, v) \leq d(a, p) + gd(p, v) \leq 2x$$

Similarly, $gd(c, Q) \leq 2y$, where $y = gd(q, Q)$ and (c, d) is the edge of the polygon Q which contains the point q. Applying the same arguments, we know that the length of the segment (a, c), is

$$d(a, c) \leq d(a, p) + d(p, q) + d(q, c) \leq x + d(p, q) + y.$$

Therefore, the weight of the bridge (a, c) is bounded by

$$w(a, c) \leq 2x + (x + d(p, q) + y) + 2y = 3x + d(p, q) + 3y$$

Hence, $\frac{w(a,c)}{w(p,q)} \leq \frac{3x+d(p,q)+3y}{x+d(p,q)+y} \leq 3$.

Let us now consider the case when $k > 1$. Extending the analysis for this case is straightforward (see Figure 3). Since $d(a, b) \leq 2x$, for any $1 \leq i \leq k - 1$, $d(a_i, a_{i+1}) \leq \frac{2x}{k}$. The algorithm AV_OVB investigates the bridges through all the vertices of the polygons and all the artificial vertices introduced on the edges. In other words, if the optimal bridge (p, q) has p lying between (a_i, a_{i-1}) for some $1 \leq i \leq k - 1$, the bridges through a_i and a_{i-1} would also be investigated. Proceeding in the fashion exactly as described for the case $k = 1$ above, we have

$$w(a_i, c_j) \leq (x + \frac{x}{k}) + (\frac{x}{k} + d(p, q) + \frac{y}{k}) + (\frac{y}{k} + y) = x(1 + \frac{2}{k}) + d(p, q) + y(1 + \frac{2}{k})$$

Thus the approximation ratio is

$$\frac{w(a_i, c_j)}{w(p, q)} \leq \frac{x(1 + \frac{2}{k}) + d(p, q) + y(1 + \frac{2}{k})}{x + d(p, q) + y} \leq 1 + \frac{2}{k}$$

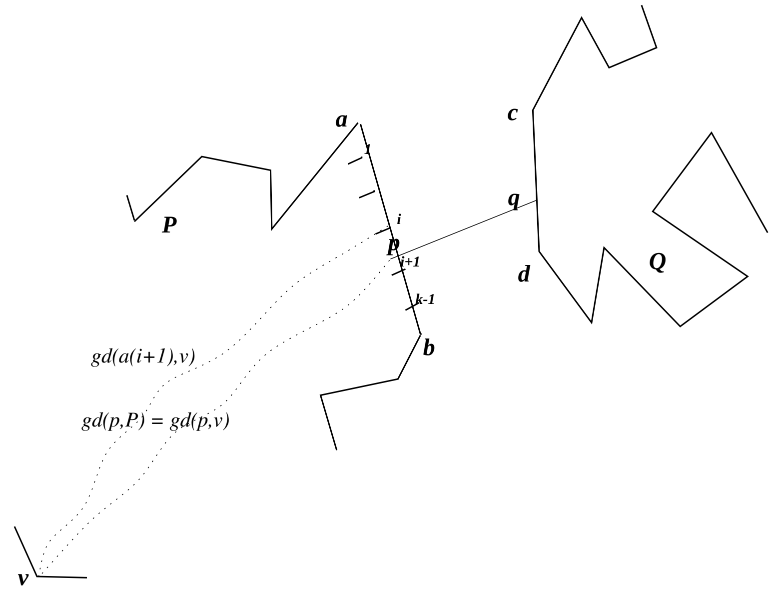

Figure 3: General case: $k-1$ artificial vertices. Point p is in the interval (a_i, a_{i+1})

Since the procedure OVB take $O(n \log n)$ time, where n is the number of vertices and the procedure AV_OVB's main work is invoking procedure OVB with $k * n$ points, it follows that is overall time complexity of AV_OVB is bounded by $O(kn \log kn)$. □

Figure 4 (b) gives the bridge generated by algorithm AV-OVB with $k = 1$ and Figure 4 (a) gives an optimal bridge for the problem instance. Figure 4 (d) gives the bridge generated by algorithm AV-OVB with $k = 2$ and Figure 4 (c) gives an optimal bridge for the problem instance. These problem instances can be easily generalized to give problem instances with an approximation bounds of about to $1 + \frac{2}{k}$ for k even, and $1 + \frac{2}{k+1}$ for k odd. Therefore our approximation bounds are almost tight. The approximation bound we have established for our approximation scheme cannot be improved significantly.

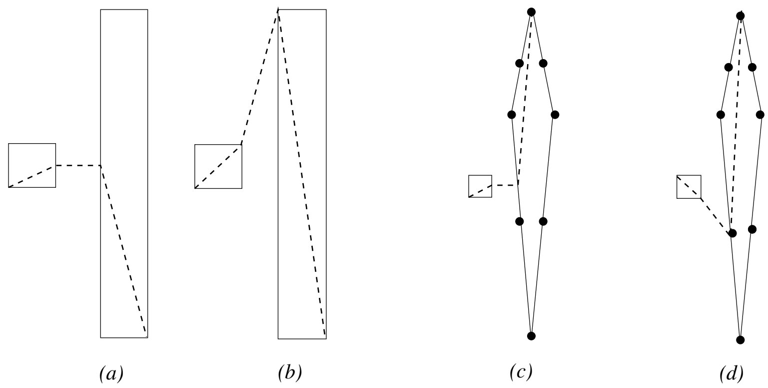

Figure 4: Worst case example for the case when $k = 1$. (a) Optimal solution, and (b) Solution generated by AV-OVB with $k = 1$. Worst case example for the case when $k = 2$. (c) Optimal solution, and (b) Solution generated by AV-OVB with $k = 2$.

2.3 Improved Approximation Scheme

In this subsection we present an improved approximation algorithm with worst case approximation bound of $1 + \frac{2}{k+1}$. However, this bound does not seem to be tight. The examples that achieve the approximation bound $1 + \frac{2}{k+1}$ given in the previous subsection, have a slightly smaller approximation bound when executed by the algorithm in this subsection.

The main idea behind our algorithm is similar to the one in the previous subsection. The main difference is the length of the intervals. In the previous subsection, the intervals had uniform lengths. But the intervals close to the center of the line segment (a, b) have the largest approximation error when the optimal bridge includes a point inside the interval, whereas the ones closer to the end points have a smaller approximation error. The reason for this is that the bound for x (i.e., the distance to the geodesic-farthest point in the polygon) increases as we move from the center of the line segment (a, b) towards the extreme points, a and b. This is because the maximum distance from any point p to a or b increases as we move p from the center of the line to the extreme points, a and b.

Our strategy is to define the interval(s) closest to the center smaller than the ones in the previous subsection so as to decrease the approximation error when the optimal bridge has an endpoint inside the interval. Then we need to compensate for the remaining intervals, i.e, make the intervals larger, so that the total number of artificial points remain equal. Following the same lines for the analysis given in the previous subsection, it is sufficient to establish that for each of the k intervals, $d(a_{i-1}, a_i) \leq 2x/(k+1)$ where $x = gd(p, P)$ with p being the end point of the bridge on $\delta(P)$ and p located in the interval (a_{i-1}, a_i).

Let us consider first the case when the number of intervals k is even. The number of points is $m = k - 1$. Let $d = d(a, b)$. Let y be the point in the center of the line from a to b. Define $\delta_1 = 1/(k+1)$, $\delta_2 = 1/k$, $\delta_3 = 1/(k-1)$, and for $4 \leq i \leq (m-1)/2$ let $\delta_i = 1/k$. There is a point located at y, and for $1 \leq i \leq (m-1)/2$ there is a point at $y + \sum_{j=1}^{i} \delta_j * d$ and at $y - \sum_{j=1}^{i} \delta_j * d$. We shall refer to the algorithm AV_OVB that uses this Non-Uniform distanced artificial vertices as procedure NAV_OVB.

Lemma 2.1 *For each interval I as defined above, $d(I) \leq 2x/(k+1)$, where $x = gd(p, P)$ with p being the end point of the bridge on $\delta(P)$ and p located in the interval I.*

Proof: Since the intervals are symmetric, we only consider the intervals between the points located at $y - \sum_{j=1}^{i} \delta_j * d$ for $1 \leq i \leq (m-1)/2$ and a. There are four types of intervals I.

Case 1: Interval I from y to $y - \delta_1$.

Since vertex b is located at a distance $\geq d/2$ from every point in the interval, if the optimal bridge has an end point in this interval, we know that $x \geq d/2$ (or equivalently $d \leq 2x$). By definition the interval has length $d/(k+1)$. Substituting $d \leq 2x$ in the previous expression we know that the length of the interval is at most $2x/(k+1)$.

Case 2: Interval I from $y - \delta_1$ to $y - \delta_1 - \delta_2$.

Since vertex b is located at a distance $\geq d/2 + d/(k+1)$ from every point in the interval, we know that $x \geq d/2 + d/(k+1)$ (or equivalently, $d \leq 2(k+1)x/(k+3)$). By definition the interval has length d/k. Substituting $d \leq 2(k+1)x/(k+3)$ in the previous expression we know that the length of the interval is at most $2(k+1)x/(k(k+3))$ which is at most $2x/(k+1)$ since $k \geq 1$.

Case 3: Interval I from $y - \delta_1 - \delta_2$ to $y - \delta_1 - \delta_2 - \delta_3$.
Since vertex b is located at a distance $\geq d/2 + d/(k+1) + d/k$ from every point in the interval, we know that Clearly $x \geq d/2 + d/(k+1) + d/k \geq d/2 + 2d/(k+1)$ (or equivalently, $d \leq 2(k+1)x/(k+5)$). By definition the interval has length $d/(k-1)$. Substituting $d \leq 2(k+1)x/(k+5)$ in the previous expression we know that the length of the interval is at most $2(k+1)x/((k+5)(k-1))$ which is at most $2x/(k+1)$ since $k \geq 2$.

Case 4: Interval I from $y - \sum_{j=1}^{i-1} \delta_j$ to $y - \sum_{j=1}^{i} \delta_j$, for $i \geq 3$.
The proof in this case is as in Case 2 since we know that $x > d/2 + d/(k+1)$. This completes the proof of the lemma. □

The proof when k is odd is similar, and for brevity we do not include it here. The overall approximation bound for both cases is established in the following theorem.

Theorem 2.3 *Algorithm* `NAV_OVB(P,Q,k)` *generates a bridge such that* $\hat{f}/f^* \leq 1 + \frac{2}{k+1}$ *in* $O(kn \log kn)$.

Proof: The proof follows from Theorem 2.2 and Lemma 2.1, and the discussion just before this theorem. □

3 Fully Polynomial Time Approximation Scheme

In this section we present a fully polynomial time approximation scheme that given any constant $\epsilon > 0$ generates a bridge such that $\hat{f}/f^* \leq 1 + \epsilon$. The idea is similar to the one in the previous section but instead of having three different types of intervals, the length of the intervals vary, and are such that they generate an error bounded by ϵ when an optimal bridge has a point inside the interval. The interval(s) closest to the center of the line from a to b are the smallest. The length of the interval increases as they are located farther from the center. The reason why we selected three different types of intervals in the previous section, was because defining the length of these intervals to generate the same error for a given value of k is complex. The idea in this section is to allow the user to specify the maximum error which can be tolerated, and then we compute the number of intervals needed and hence the number of points to be introduced along each line.

Lets consider first the case when k is even. Let $d = d(a,b)$. Let y be the point in the center of the line segment from a to b. We will define the values $\delta_1, \delta_2, ...$ in terms of ϵ. There is a point at y as well as at $y + \sum_{j=1}^{i} \delta_j * d$ and at $y - \sum_{j=1}^{i} \delta_j * d$, for $i = 1, 2, \ldots,$ until we pass points a and b. We define $\delta_i = \frac{\epsilon(1+\epsilon)^{(i-1)}}{2}$ for $i \geq 0$. We shall refer to the algorithm AV_OVB that uses artificial vertices defined in terms of ϵ as procedure AVe_OVB.

Lemma 3.1 *The length of each of the interval* $\delta_i * d$ *is at most* ϵx*, where* $x = gd(p,P)$ *with* p *being the end point of the bridge on* $\delta(P)$ *and* p *located in the current interval.*

Proof: Consider the ith interval from the center. Clearly $x \geq d/2 + \sum_{j=1}^{i-1} \delta_j d$ (or equivalently, $d \leq 2x/(1 + 2\sum_{j=1}^{i-1} \delta_j)$.

The length of the ith interval is $\delta_i d$. Substituting the above expression we know that the length of the interval is at most $2\delta_i x/(1+2\sum_{j=1}^{i-1}\delta_j)$.

Inductively one can establish that $1+2\sum_{j=1}^{i}\delta_j = (1+\epsilon)^i$ because

$$1+2\sum_{j=1}^{i}\delta_j = 1+2\sum_{j=1}^{i-1}\delta_j + 2\delta_i = (1+\epsilon)^{i-1} + 2*(\epsilon/2)(1+\epsilon)^{i-1} = (1+\epsilon)^i$$

Substituting $1+2\sum_{j=1}^{i}\delta_j = (1+\epsilon)^i$ as well as the definition for δ_i in $2\delta_i x/(1+2\sum_{j=1}^{i-1}\delta_j)$, we know that the length of the interval is at most $2\delta_i x/(1+2\sum_{j=1}^{i-1}\delta_j) = \epsilon x$.

This completes our proof of the lemma. $\square$

Theorem 3.1 *Algorithm* `AVe_OVB(P,Q,k)` *generates a bridge such that* $\hat{f}/f^* \leq 1+\epsilon$ *in* $O(kn\log kn)$*, where* k*, the number of intervals in each line, is even. The time complexity of the algorithm is* $O(kn\log kn)$*, where* $k = 2*\lceil\frac{1}{\log(1+\epsilon)}\rceil$.

Proof: The proof follows from Theorem 2.2 and Lemma 3.1, and the discussion just before this theorem.

Now, since $\epsilon > 0$, we know $\sum_{j=1}^{m}\delta_j$ is $S_m = ((1+\epsilon)^m - 1)/2$. The least value of m such that $S_m \geq 1/2$ indicates that to achieve the approximation error ϵ one needs $2(m-1)+1 = 2m-1$ points, with the intervals having length defined by $\delta_i * d$ for $1 \leq i \leq m-1$. Therefore, $m = \lceil 1/\log(1+\epsilon)\rceil$ and the total number of intervals $k = 2m = 2\lceil 1/\log(1+\epsilon)\rceil$. $\square$

For the case when we introduce an odd number of intervals the situation is slightly different. For this case $\delta_i = \frac{\epsilon}{2+\epsilon}(1+\epsilon)^{i-1}$ and the artificial points are located at $y + 0.5\delta_1 d + \sum_{j=2}^{i}\delta_j * d$, and $y - 0.5\delta_1 d - \sum_{j=2}^{i}\delta_j * d$ for $i = 1, 2, \ldots$ until we pass points a and b. Remember that y is the center point in line ab.

For this case we can prove a lemma identical to Lemma 3.1 and the theorem becomes

Theorem 3.2 *Algorithm* `AVe_OVB(P,Q,k)` *generates a bridge such that* $\hat{f}/f^* \leq 1+\epsilon$ *in* $O(kn\log kn)$*, where* k*, the number of intervals in each line, is odd. The time complexity of the algorithm is* $O(kn\log kn)$*, where* $k = 2*\lceil 1+\log(2+\epsilon)/\log(1+\epsilon)\rceil + 1$.

Proof: The proof is similar to Theorem 3.1, but follows the definitions in the above discussion. $\square$

For some values of ϵ it is better to choose the number of intervals odd and for other cases it is better to choose it even. The following table lists, for a set of ϵ values the least number of points needed to achieve the approximation bound ϵ as well as the δ_i values.

4 Concluding Remarks

In this paper we have presented approximation schemes for the optimal bridge problem, all of which use an efficient algorithm for the optimal *vertex* bridge problem. We have also investigated a few other variations of the basic schemes presented here. One of them is based on the idea of introducing a number of artificial points per line segment depending on the

Table 1: Number of points needed to achieve the approximation bound ϵ

ϵ	$k-1$	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6	δ_7	δ_8
0.6667	2	0.2500	0.4167	-	-	-	-	-	-
0.5000	3	0.2500	0.3750	-	-	-	-	-	-
0.4000	4	0.1667	0.2333	0.3267	-	-	-	-	-
0.3333	4	0.1429	0.1905	0.2540	-	-	-	-	-
0.2857	5	0.1429	0.1837	0.2362	-	-	-	-	-
0.2500	6	0.1111	0.1389	0.1736	0.2170	-	-	-	-
0.2222	6	0.1000	0.1222	0.1494	0.1826	-	-	-	-
0.2000	7	0.1000	0.1200	0.1440	0.1728	-	-	-	-
0.1818	8	0.0833	0.0985	0.1164	0.1376	0.1626	-	-	-
0.1667	9	0.0833	0.0972	0.1134	0.1323	0.1544	-	-	-
0.1538	9	0.0769	0.0888	0.1024	0.1182	0.1363	-	-	-
0.1429	10	0.0667	0.0762	0.0871	0.0995	0.1137	0.1300	-	-
0.1333	11	0.0667	0.0756	0.0856	0.0970	0.1200	0.1247	-	-
0.1250	11	0.0625	0.0703	0.0791	0.0890	0.1001	0.1126	-	-
0.1176	12	0.0556	0.0621	0.0694	0.0776	0.0867	0.0969	0.1083	-
0.1111	13	0.0556	0.0617	0.0686	0.0762	0.0847	0.0941	0.1045	-
0.1053	13	0.0526	0.0582	0.0643	0.0711	0.0785	0.0868	0.0959	-
0.1000	14	0.0476	0.0524	0.0576	0.0633	0.0697	0.0767	0.0844	0.0928
0.0952	15	0.0476	0.0522	0.0571	0.0626	0.0685	0.0751	0.0822	0.0900
0.0909	15	0.04545	0.0496	0.0541	0.0590	0.0644	0.0702	0.0766	0.0836

length of the segment. In general, this strategy allows the same approximation bound while reducing the overall time complexity.

Our algorithms are amenable to efficient implementations, and the only external data structures and algorithms required are those for the *additively-weighted* nearest point Voronoi diagram and Suri's algorithm [11] for computing the geodesic-farthest neighbors of all vertices of a simple polygon. A *geodesic-farthest-point* Voronoi diagram [2] supporting geodesic-farthest-point queries in $O(\log n)$ time after an $O(n \log n)$ time preprocessing stage can also be used instead of Suri's algorithm.

Our approximation can also be applied to the ag-bridge problem. To do this we need to compute $gd(p, q)$ for every pair of points belonging to different polygons. If $T_{gdsp}(n)$ represents the time required to build the tree of *geodesic shortest paths* from a point to a given set of $O(n)$ other points in presence of polygonal obstacles in the plane, we can achieve the same approximation bounds for the ag-bridge problem in $O(k^2 \cdot nT_{gdsp}(n))$ time since the shortest paths tree needs to be computed for every vertex of the polygon P (or Q). The algorithm by Hershberger and Suri [7] builds the geodesic shortest paths tree of a point in $O(n \log n)$ time. This gives us an approximation of the ag-bridge problem in $O(k^2 n^2 \log n)$ time. Since the number of *obstacles* is fixed to 2 (the polygons P and Q) in our problem, the algorithm by Kapoor, et al. [8] can be used to build the tree in $O(n)$ time, thus producing an approximately optimal ag-bridge in $O(k^2 n^2)$ time. Note that there is no known polynomial time algorithm for computing the optimal ag-bridge between two polygons.

Some of the interesting open problems suggested by our work include design of approximation schemes that do not use an optimal vertex bridge algorithm. Also, it would be challenging to improve the $O(n \log^3 n)$ bound for the exact solution of the problem, as presented by Tan [13]. Another obvious open question is developing an algorithm for the *exact* solution of the ag-bridge problem (since, as illustrated by Figure [1], Tan's algorithm does not solve this problem). The time required for approximating the ag-bridge problem is also pretty high, and admits a scope for improvement.

References

[1] N. M. Amato. Determining the separation of simple polygons. *International Journal of Computational Geometry and Applications*, 4(4):457–474, 1994.

[2] B. Aronov, S. Fortune, and G. Wilfong. The furthest-site geodesic voronoi diagram. *Discrete Comput. Geom.*, 9:217–255, 1993.

[3] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structures. *ACM Comput. Surveys*, 23(3):343- 405, 1991.

[4] F. Aurenhammer and H. Imai. Geometric relations among voronoi diagrams. In *Geom. Dedicata 27*, pages 65- 75, 1988.

[5] B. Bhattacharya and R. Benkoczi. On computing the optimal bridge between two convext polygons. *Information Processing Letters*, 79(5):215–221, 2001.

[6] L. Cai, Y. Xu, and B. Zhu. Computing the optimal bridge between two convex polygons. *Information Processing Letters*, 69:127- 130, 1999.

[7] J. Hershberger and S. Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM J. Comput.*, 28:2215 – 2256, 1999.

[8] S. Kapoor, S. N. Maheshwari, and J. S. B. Mitchell. Efficient algorithms for euclidean shortest paths among polygonal obstacles in the plane. In *Symposium on Computational Geometry*, pages 172–182, 1988.

[9] S. K. Kim. The canal problems. *J. Korea Inform. Sci. Soc.*, 20(12):1891–1901, 1993 (in Korean).

[10] S. K. Kim and C. S. Shin. Computing the optimal bridge between two polygons. *Theory of Computing Systems*, 34(4):337–352, 2001.

[11] S. Suri. Computing geodesic furthest neighbors in simple polygons. *J. of Comp. and Sys. Sciences*, 39:220–235, 1989.

[12] X. Tan. On optimal bridges between two convex regions. *Information Processing Letters*, 76:163–168, 2000.

[13] X. Tan. Finding an optimal bridge between two polygons. *International Journal of Computational Geometry and Applications*, 12(3):249–262, 2002.

[14] T. Tokuyama. Efficient algorithm for the minimum diameter bridge problem. *Lect. Notes Comput. Sci.*, 2098:362–369, 2001.

[15] D. P. Wang. An optimal algorithm for constructing an optimal bridge between two simple rectilinear polygons. *Information Processing Letters*, 79:229–236, 2001.