Constructing Pairwise Disjoint Paths with Few Links

Himanshu Gupta
SUNY, Stony Brook
and
Rephael Wenger
Ohio State University

Let P be a simple polygon and let $\{(u_1, u'_1), (u_2, u'_2), \ldots, (u_m, u'_m)\}$ be a set of m pairs of distinct vertices of P where for every distinct $i, j \leq m$, there exist pairwise disjoint (nonintersecting) paths connecting u_i to u'_i and u_j to u'_j. We wish to construct m pairwise disjoint paths in the interior of P connecting u_i to u'_i for $i = 1, \ldots, m$, with minimal total number of line segments. We give an approximation algorithm that constructs such a set of paths using $O(M)$ line segments in $O(n \log m + M \log m)$ time, where M is the number of line segments in the optimal solution and n is the size of the polygon.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: link paths, noncrossing, polygon, isomorphic triangulations

1. INTRODUCTION

Let P be a simple polygon and let u and u' be two distinct vertices of P. The *link distance* from u to u' is the minimum number of line segments (also called *links*) required to connect u to u' by a polygonal path lying in P. A polygonal path that uses the minimum number of required line segments is called a *minimum link path*. Clearly, there may be more than one such path. Suri [1986a] gave a linear time algorithm for determining the link distance and a minimal link path between two vertices.

A minimum link path from u to u' may intersect the boundary of P at many points other than u and u'. The *interior link distance* from u to u' is the minimum number of line segments required to connect u to u' by a polygonal path through the interior of P. Such an interior polygonal path that uses the minimum number

Extended Abstract appeared in *Proceedings of the Intl. Workshop on Algorithms and Data Structures (WADS)*, 1997.
Authors Address: Himanshu Gupta, SUNY, Stony Brook NY 11794. Rephael Wenger, Ohio State University, Columbus, OH 43210 (supported by NSA grant MDA904-93-H-3026).

of required line segments is called a *minimum link interior path*. The interior link distance from u to u' may differ greatly from the link distance between the two points. (See Figure 1). The (interior) *link diameter* of a polygon is the greatest (interior) link distance between any two points in the polygon.

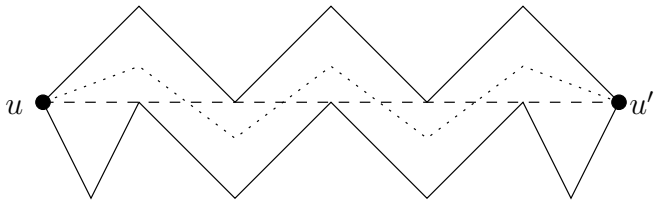

Fig. 1. Minimum link and minimum interior link paths.

Let u_1, u_1', u_2, u_2' be four vertices lying in the given order around P. By virtue of the relative locations of these four vertices, there are nonintersecting paths, ζ_1 and ζ_2, connecting u_1 to u_1' and u_2 to u_2', respectively. However, it is possible that every minimum interior link path connecting u_1 to u_1' intersects every minimum interior link path connecting u_2 to u_2'. See Figure 2. To simultaneously connect u_1 to u_1' and u_2 to u_2' by nonintersecting interior paths requires more line segments. In general, two additional line segments suffice to construct two such nonintersecting interior paths [Gupta and Wenger 1997].

Pairwise Disjoint Link Paths (PDLP) Problem. A set $\{(u_1, u_1'), (u_2, u_2'), \ldots, (u_m, u_m')\}$ of m pairs of distinct vertices of P is *untangled* if there exists a set of pairwise disjoint paths connecting each u_i to u_i'. The *pairwise disjoint link paths (PDLP)* problem can be defined as follows. Given an untangled set $\{(u_1, u_1'), (u_2, u_2'), \ldots, (u_m, u_m')\}$ of m pairs of distinct vertices of P, construct an optimal set of pairwise disjoint (nonintersecting) interior paths that connects each u_i to u_i' and uses minimum total number of line segments. We were unable to give a polynomial time algorithm for this problem or to determine if the problem is NP-hard. Instead we present an algorithm in Sections 4 that finds a solution within a constant factor of the optimal solution. We note here that the problem of finding m vertex-disjoint paths in a planar graph for m arbitrary pairs of vertices is known to be NP-complete [Garey and Johnson 1979]. For the case of $m = 2$, the above vertex-disjoint paths problem in planar graphs is solvable in polynomial time [Shiloach 1980].

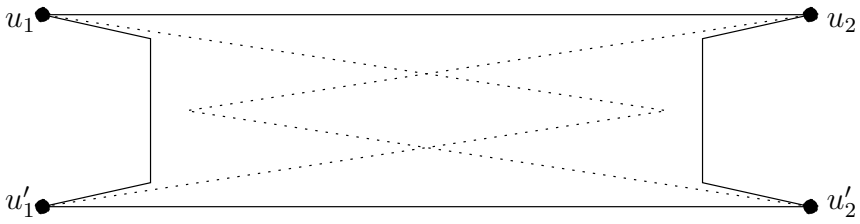

Fig. 2. Intersecting minimum link paths.

Motivation. The motivation for reducing the number of links in a path comes from environments such as motion planning, broadcasting transmission, or VLSI, where a turn is more expensive than moving along a straight-line motion (see [Suri 1986b]). In particular, in VLSI circuits the number of bends corresponds to the number of vias used. Thus, link distance problems have been studied extensively in computational geometry [Suri 1986b; Aggarwal et al. 1993; 1994; Alsuwaiyel and Lee 1993; Arkin et al. 1995; 1992]. Motivation for the noncrossing paths comes from VLSI layout design ([Takahashi et al. 1992; 1996; 1993; Lee and Yang 1996; Yang et al. 1997]). Our motivation for the PDLP problem comes from our earlier work on constructing isomorphic triangulations between simple polygons in [Gupta and Wenger 1997]. A triangulation T_P of P (possibly with interior vertices) is *isomorphic* to a triangulation T_Q of Q if there is a one-to-one, onto mapping f between the vertices of T_P and the vertices of T_Q such that p, p', p'' are vertices of a triangle in T_P if and only if $f(p), f(p'), f(p'')$ are vertices of a triangle in T_Q. The size of a triangulation is the total number of vertices, edges and triangles in the triangulation. The main result in this paper improves the output size of our approximation algorithm [Gupta and Wenger 1997] for constructing isomorphic triangulations from $O(M_1 \log n + n \log^2 n)$ to $O(M_1 \log n)$, where n is the input size and M_1 is the size of the optimal solution. We outline this improvement in Section 6.

Related Work. Takahashi et al. [1992; 1996] present an $O(n \log n)$ time optimal algorithm for the problem of finding noncrossing paths between pairs of vertices on two specified face boundaries (one outer and one inner face) of a planar graph of size n. In their subsequent work, Takahashi et al. [1993] address the problem of finding noncrossing rectilinear paths inside an outer rectangle with rectangular obstacles. All of the above works have considered the problem of finding noncrossing paths in limited contexts with the aim of minimizing the total geodesic distance, i.e., the total Euclidean distance. For a simple polygon of n vertices, Papadopoulou [1999] presents an $O(n+m)$ time algorithm to optimally compute the set of m noncrossing shortest (geodesically) paths between m pairs of points on the polygon's boundary. In all of the above cases, the paths are allowed to overlap but not cross. The link distance related problems seem to more difficult to solve than corresponding problems using the geodesic distance measure. The difficulties stem from the fact that several minimum link paths may exist connecting a given pair of vertices, while the minimum geodesic path is always unique.

In [Yang et al. 1997], the authors address the problem of finding two noncrossing rectilinear paths within a rectilinear polygon, with an aim to minimize both the total geodesic length and the number of bends. They present a linear time optimal algorithm for the problem. A survey of finding noncrossing paths in rectangular domain appears in [Lee and Yang 1996].

Prior Work. The PDLP problem was first discussed in [Gupta and Wenger 1997], where we proposed an algorithm that returns a solution of size $O(n + L + m \log m)$ in $O(n+L+m \log m)$ time, where L is the sum of the interior link distances between each pair of given vertices. Note that the optimal size is at least L. In [Gupta and Wenger 1997], we also claimed without proof that there are instances where the optimal solution requires at least $O(n + L + m \log m)$ line segments. In this article,

we start with constructing a class of polygons wherein the optimal solution of the PDLP problem requires $O(n + L + m \log m)$ links. The main result of this article is an approximation algorithm that constructs pairwise disjoint paths using $O(M)$ line segments in $O(n \log m + M \log m)$ time, where M is the minimum number of line segments required. In essence, the approximation algorithm is an improvement over our prior result in [Gupta and Wenger 1997] when $M = o(m \log m)$.

Notation $\{x_i\}$. Throughout this article, we use the notation $\{x_i\}$ to denote the set of elements $\{x_1, \ldots, x_y\}$, where the value y is either evident from the context or not relevant. For instance, we use $\{u_i\}$ to denote the set of vertices $\{u_1, u_2, \ldots, u_y\}$ where the value y depends on the context, $\{s_i\}$ to to denote the set of segments $\{s_1, s_2, \ldots, s_m\}$, etc.

Article Organization. The rest of the article is organized as follows. In the next section, we present some definitions and notations related to visibility in a polygon. In Section 3, we show by construction that there are instances of the PDLP problem that require $\Omega(n \log n)$ links where n is the size of the input polygon. We note that subsequent sections do *not* depend on the discussion in Section 3. In Section 4, we describe our approximation algorithm for the PDLP problem. Section 5 contains the proof of a lemma (about partitioning a set of line segments) used in Section 4.1. In Section 6, we use our approximation algorithm for the PDLP problem to improve our prior result in [Gupta and Wenger 1997] on construction of isomorphic triangulations between simple polygons.

2. VISIBILITY

The construction of pairwise disjoint paths uses many of the ideas and notation developed in the study of polygon visibility. (See [Goodman and O'Rourke 1997, Chapter 25].) In this section, we present some basic definitions and notation that we will use throughout this paper.

DEFINITION 1. (Interior of an Edge.) Interior of an edge e is the open line segment of e without its end points. We use $\text{int}(e)$ to denote interior of an edge e. □

DEFINITION 2. (Visibility; Clear Visibility.) Consider points p and p' in a polygon P. Point p is *visible* from point p' if $p = p'$ or if P contains the open line segment (p, p'). Point p is *clearly visible* from point p' if $p = p'$ or if the *interior* of P contains the open line segment (p, p').

Let P be a simple polygon and T_P be a triangulation of P. Let t_1 be a point in P, or an edge of P or T_P, or a triangle of T_P. Similarly, let t_2 be a point in P, or an edge of P or T_P, or a triangle of T_P. We say that t_1 is *(clearly) visible*[1] from t_2 in P if there are points $p_1 \in t_1$ and $p_2 \in t_2$ such that p_1 is (clearly) visible from p_2. □

[1]This definition of visibility is sometimes called *weak visibility* as opposed to *strong visibility* where every point in t_1 must be visible from every point in t_2. Throughout this paper, visibility refers to weak visibility.

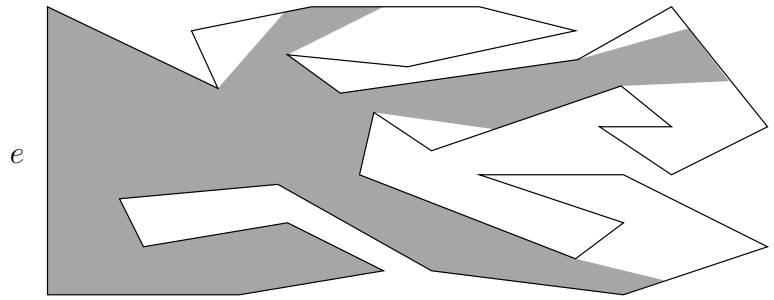

Fig. 3. The clear visibility polygon $\widehat{\text{Vis}}(e)$ for an edge e.

DEFINITION 3. (Visibility Polygons.) The *(clear) visibility polygon* from a point p in P is the set of points in P (clearly) visible from p. The *(clear) visibility polygon* from a line segment $e \subseteq P$ is the set of points in P (clearly) visible from e. (See Figure 3.)

We denote the visibility polygons as $\text{Vis}_P(p)$ and $\text{Vis}_P(e)$ and the clear visibility polygons as $\widehat{\text{Vis}}_P(p)$ and $\widehat{\text{Vis}}_P(e)$. We use $\text{Vis}(p), \text{Vis}(e), \widehat{\text{Vis}}(p)$, and $\widehat{\text{Vis}}(e)$ whenever the polygon P is evident from the context. Note that $\text{Vis}(p)$ and $\text{Vis}(e)$ are closed sets, while $\widehat{\text{Vis}}(p)$ and $\widehat{\text{Vis}}(e)$ are not. □

Constructing Clear Visibility Polygons. Chazelle [1991] gives a linear time algorithm for constructing a triangulation T_P of P (using no vertices other than those of P). Using this triangulation, Guibas et al. [1987] give a linear time algorithm for constructing $\text{Vis}(e)$ for an edge e of P. The algorithm in [Guibas et al. 1987] can be easily modified to construct a clear visibility polygon $\widehat{\text{Vis}}(e)$ in $O(n_e)$ time (after the $O(|P|)$ triangulation construction), where n_e is the number of triangles of T_P intersected by $\widehat{\text{Vis}}(e)$. Below, we describe the modifications required.

For each vertex u of P, the algorithm by Guibas et al. constructs a representation of the shortest (minimum Euclidean length) paths $\pi_1(u)$ and $\pi_2(u)$ to the two endpoints of e. Then, it uses that information to determine the subsegment $\text{Vis}(e) \cap d$, for each diagonal (triangulation edge) d of the given triangulation T_P of P. Finally, it traverses P in clockwise order connecting these subsegments by segments in P.

A diagonal d is visible but not clearly visible from e if and only if $\text{Vis}(e) \cap d$ is a single point. Otherwise, $\widehat{\text{Vis}}(e) \cap d$ equals $\text{Vis}(e) \cap d$ except perhaps for the endpoints. Thus, Guibas et al.'s algorithm can also construct the subsegments $\widehat{\text{Vis}}(e) \cap d$ and then, connect these subsegments by segments in P to form $\widehat{\text{Vis}}(e)$. Moreover, the algorithm by Guibas et al. also constructs the points on e that are visible to d, for each diagonal d visible to e. Thus, their algorithm can also be used to determine $\widehat{\text{Vis}}(d) \cap e$ for all diagonals d of T_P.

Time Analysis. Instead of first constructing the shortest paths from the endpoints of e to all the vertices, we check for clearly visible diagonals as the shortest paths are constructed. Whenever a diagonal d is found that is not visible to e, the entire region that is separated from e by d is not visible to e and hence, does not need

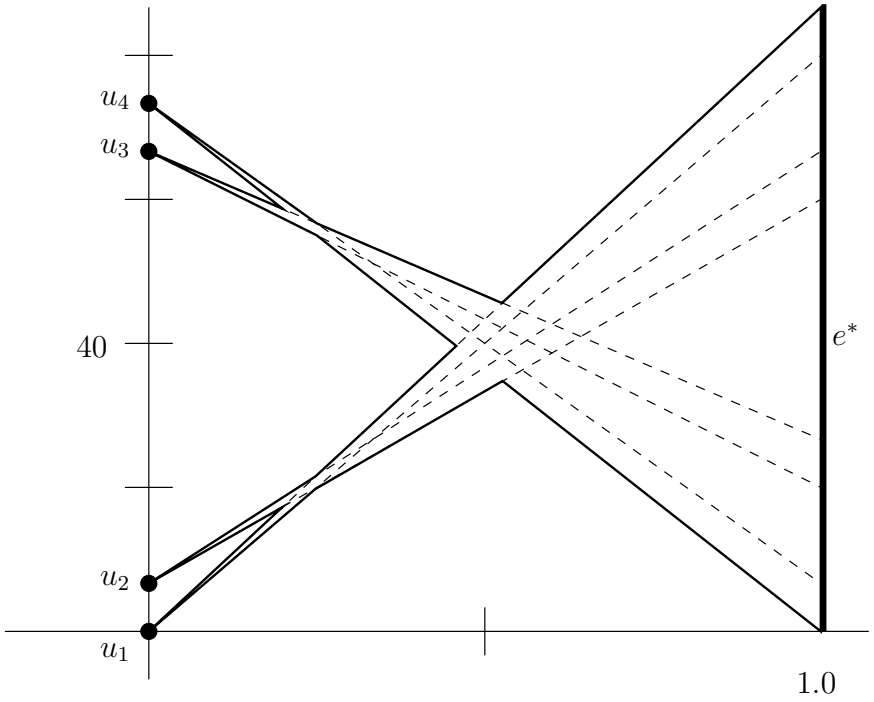

Fig. 4. Polygon with link diameter two such that pairwise disjoint paths from u_1 to u_4 and u_2 to u_3 require six line segments.

to be processed. Thus, the construction of $\widehat{\text{Vis}}(e)$ of e and the set of segments $\{e \cap \widehat{\text{Vis}}(d) : d \text{ is a diagonal of } T_P)\}$ as described above can be done in $O(n_e)$ time. In summary, we have the following theorem.

THEOREM 1. *Let P be a simple polygon and e be an edge of P. Given a triangulation T_P of P, the clear visibility region $\widehat{\text{Vis}}(e)$ of e and the set $\{e \cap \widehat{\text{Vis}}(d) : d$ is a diagonal of $T_P\}$ can be constructed in $O(n_e)$ time, where n_e is the number of triangles in T_P intersected by $\widehat{\text{Vis}}(e)$.* ∎

3. WORST CASE BOUNDS

In this section, we construct polygons with n vertices, interior link diameter of two, and $(n+1)/8$ pairs of vertices, such that any set of pairwise disjoint paths connecting the pairs requires a total of $\Omega(n \log n)$ links. We first give a general outline of the construction.

Our polygon P will have a set $\{u_1, u_2, \ldots, u_{(n+1)/4}\}$ of $(n+1)/4$ vertices on the vertical line $x = 0$. These vertices will be endpoints for the $(n+1)/8$ paths. One edge e^* of the polygon will lie on the vertical line $x = 1$. All other vertices and edges of the polygon will be between these two vertical lines. See Figure 4. Each point in this polygon is visible from e^*. The clear visibility polygon, $\widehat{\text{Vis}}(u_i)$, from each vertex u_i is a "visibility" triangle with one edge on e^*. These visibility triangles, $\widehat{\text{Vis}}(u_i)$, intersect and "cross" over one another. Thus, every pair (u_i, u_j) can be

connected by a line segment from u_i to a point p in $\widehat{\text{Vis}}(u_i) \cap \widehat{\text{Vis}}(u_j)$ and then by a line segment from p to u_j. On the other hand, given the two pairs (u_1, u_4) and (u_2, u_3), either the path from u_1 to u_4 or the path from u_2 to u_3 must have an additional vertex in $\widehat{\text{Vis}}(u_1) \cap \widehat{\text{Vis}}(u_2)$. Similarly, one of those paths must have an additional vertex in $\widehat{\text{Vis}}(u_3) \cap \widehat{\text{Vis}}(u_4)$. Thus, pairwise disjoint paths connecting those pairs must use a total of six, not four, line segments.

More generally, we can imagine the set of vertices $\{u_i\}$ as leaves of a balanced binary tree. The clear visibility polygons, $\{\widehat{\text{Vis}}(u_i)\}$ will again be "visibility" triangles which intersect and "cross" over one another. Each internal node of the binary tree represents the intersections of the visibility triangles of the leaves of the left subtree with the intersections of the visibility triangles of the leaves of the right subtree. Each such set of intersections will be clustered in a quadrilateral. Sweeping the polygon and quadrilaterals from left ($x = 0$) to right ($x = 1$) will correspond to sweeping the binary tree from bottom to top. This arrangement of the visibility triangles will force many of the pairwise disjoint paths to have $\Omega(\log n)$ line segments. It is easy to construct polygons as described above. Below, we give a complete construction of such a polygon with precise coordinates.

Construction of $\Omega(n \log n)$ case. We now give a formal construction of a polygon P with $n = 2^{k+1} - 1$ vertices and $(n+1)/8$ pairs of vertices such that any set of pairwise disjoint paths connecting the pairs requires a total of $\Omega(n \log n)$ links. We start by defining two functions f and g that we will use in specifying the line segments that bound the polygon P.

DEFINITION 4. (Functions $f(a)$ and $g(a)$.) Each non-negative integer $a < 2^k$ can be represented in binary as the sum of powers of two as follows.

$$a = \sum_{i=0}^{k-1} \alpha_i 2^i, \alpha_i \in \{0, 1\}.$$

We define $f(a)$ and $g(a)$ as:

$$f(a) = \sum_{i=1}^{k-1} \alpha_i 8^i,$$

$$g(a) = \sum_{i=0}^{k-1} (1 - \alpha_i) 4^i.$$

Note that $f(a) = f(a+1)$ when a is even, as in the definition of f the summation starts from $i = 1$. □

DEFINITION 5. (l_a, e^*, and the polygon P.) As shown in Figure 5, we define the following. Here, we use $[p, q]$ to denote the closed line segment connecting points p and q.

— $l_a = [(0, f(a)), (1, 2^{k-1} g(a))]$.[2]

[2]Using $(1, g(a))$ will work just as well but the 2^{k-1} factor generates a more symmetric polygon.

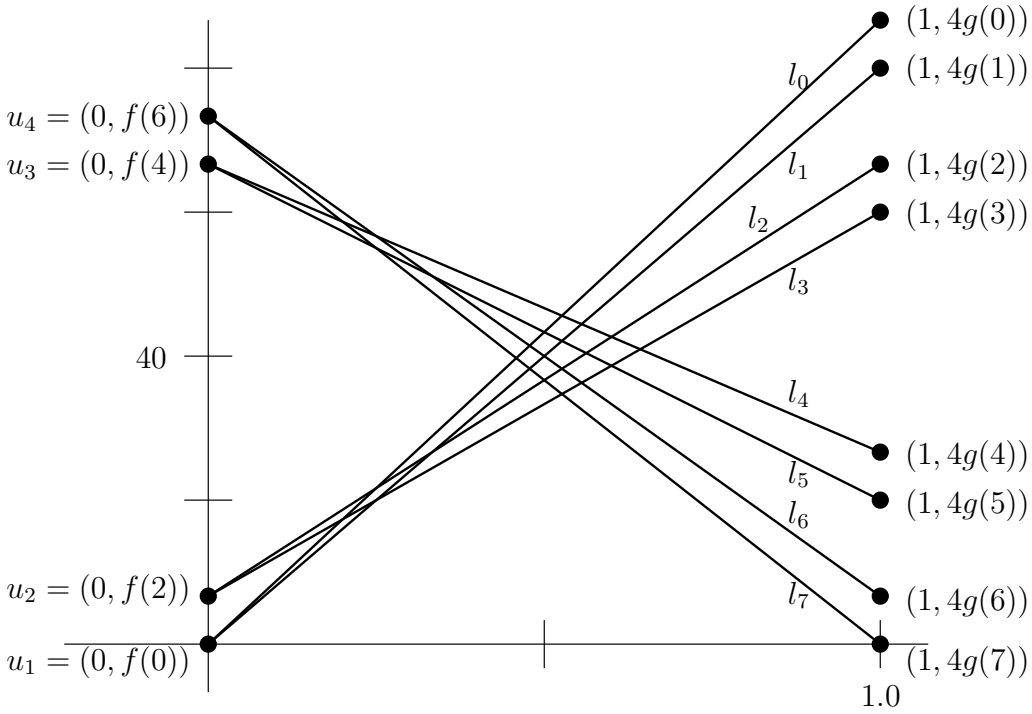

Fig. 5. Line segments l_a connecting the points $(0, f(a))$ to $(1, 2^{k-1}g(a))$ for $a = 0, \ldots, 7$ for $k = 3$. Note that $(0, f(a)) = (0, f(a+1))$ for even a.

— $e^* = [(1, 2^{k-1}g(0)), (1, 0)]$, i.e., joining the extreme points on $x = 1$.
— Polygon P is defined as the union of all the bounded regions in the arrangement of $\{l_a : a = 0, \ldots, 2^k - 1\} \cup \{e^*\}$. See Figure 4.

□

For even a, we have $f(a) = f(a+1)$ and thus, the line segments l_a and l_{a+1} intersect on the y-axis. Let $u_i = (l_{2i-1} \cap l_{2i-2})$. The vertices of P on the y-axis are exactly these intersection points viz. $\{u_1, u_2, \ldots, u_{2^{k-1}}\}$. Note that $u_i = (0, f(2i-1)) = (0, f(2i-2))$. Now, pair each vertex u_i with the vertex $u_{2^{k-1}-(i-1)}$ for $i = 1, \ldots, 2^{k-2}$, constructing $2^{k-2} = (n+1)/8$ pairs of vertices of P. We claim that any set of non-intersecting paths connecting these pairs in P requires at least $(1/32)n \log_2 n$ links for even k. We first show that the relative position of the intersection points of two lines l_a and l_b can be determined by the most significant bit in which a and b differ.

LEMMA 1. *If the most significant bit in which a and b differ is less than the most significant bit in which a' and b' differ, then $l_a \cap l_b$ is to the left of $l_{a'} \cap l_{b'}$.*

Proof: The intersection of segment l_a and l_b has x-coordinate

$$\left(1 + 2^{k-1}\frac{g(b) - g(a)}{f(a) - f(b)}\right)^{-1},$$

whenever $a \neq b$ (and hence, $f(a) \neq f(b)$). If $a = \sum_{i=0..k-1} \alpha_i 2^i$, $\alpha_i \in \{0,1\}$, and $b = \sum_{i=0..k-1} \beta_i 2^i$, $\beta_i \in \{0,1\}$, then

$$f(a) - f(b) = \sum_{i=1}^{k-1} (\alpha_i - \beta_i) 8^i, \quad \text{and}$$
$$g(b) - g(a) = \sum_{i=0}^{k-1} (\alpha_i - \beta_i) 4^i.$$

Let $j > 0$ be the most significant bit in which a and b differ. Then,

$$\frac{g(b)-g(a)}{f(a)-f(b)} \leq \frac{4^j + 4^{j-1} + 4^{j-2} + \cdots + 4 + 1}{8^j + 8^{j-1} + 8^{j-2} + \cdots + 8} = \frac{7}{3}\left(\frac{4^{j+1}-1}{8^{j+1}-8}\right), \quad \text{and}$$
$$\frac{g(b)-g(a)}{f(a)-f(b)} \geq \frac{4^j - 4^{j-1} - 4^{j-2} - \cdots - 4 - 1}{8^j - 8^{j-1} - 8^{j-2} - \cdots - 8} = \frac{7}{3}\left(\frac{2\times 4^j + 1}{6\times 8^j + 8}\right).$$

Thus,

$$\frac{7}{3}\left(\frac{2\times 4^j + 1}{6\times 8^j + 8}\right) \leq \frac{g(b)-g(a)}{f(a)-f(b)} \leq \frac{7}{3}\left(\frac{4\times 4^j - 1}{8\times 8^j - 8}\right).$$

If $j+1$ is the most significant bit in which a' and b' differ, then

$$\frac{g(b')-g(a')}{f(a')-f(b')} \leq \frac{7}{3}\left(\frac{4\times 4^{j+1} - 1}{8\times 8^{j+1} - 8}\right) < \frac{7}{3}\left(\frac{2\times 4^j + 1}{6\times 8^j + 8}\right) \leq \frac{g(b)-g(a)}{f(a)-f(b)}.$$

Thus,

$$\left(1 + 2^{k-1}\frac{g(b')-g(a')}{f(a')-f(b')}\right)^{-1} > \left(1 + 2^{k-1}\frac{g(b)-g(a)}{f(a)-f(b)}\right)^{-1},$$

and the intersection point of l_a and l_b is to the left of the intersection point of $l_{a'}$ and $l_{b'}$. This is also true if a and b differ only in the zero'th bit, since then $l_a \cap l_b$ lies on the y-axis (extreme right). ∎

Recall that the polygon P is a union of all the bounded regions in the arrangement of $\{l_a : a = 0, \ldots, 2^k - 1\} \cup \{e^*\}$, where e^* is the line segment joining the extreme points on the line $x = 1$. (See Figure 4.) The vertices of P on the y-axis are $\{u_1, u_2, \ldots, u_{2^{k-1}}\}$, where $u_i = (0, f(2i-1))$.

PROPOSITION 1.

(1) *Polygon P has $n = 2^{k+1} - 1$ vertices.*

(2) *Polygon P has interior link diameter two.*

(3) *Any family of pairwise disjoint paths in P connecting u_i to $u_{2^{k-1}-(i-1)}$ for $i = 1, \ldots, 2^{k-2}$ must contain $\Omega(k2^k) = \Omega(n \log_2 n)$ line segments.*

Proof: Let $\{u_1, u_2, \ldots, u_{2^{k-1}}\}$, the set of vertices of P on the y-axis.

Number of Vertices. Consider the vertices $\{u_i\}$ as leaves of a balanced binary tree $\mathcal{T}$, where the leaves are ordered by i. Associate the line segments l_{2i-2} and l_{2i-1} with the leaf containing u_i. Also, associate each internal node μ with the set $\mathcal{I}_\mu$ of intersections of the line segments of the left subtree leaves with the line segments of

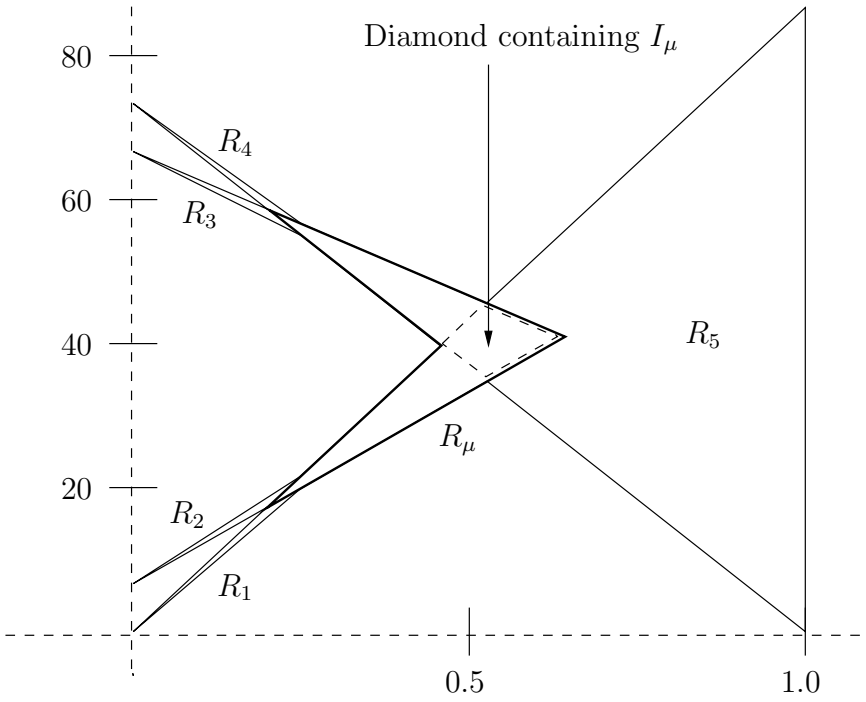

Fig. 6. The diamond containing the intersection points in I_μ, and the subpolygon R_μ dividing polygon P into subpolygons R_1, R_2, R_3, R_4 and R_5. Here, $k = 3$ and μ is the root of $\mathcal{T}$ (i.e., $h = k - 1$).

the right subtree leaves. For each leaf μ, let $\mathcal{I}_\mu = \{u_i = l_{2i-2} \cap l_{2i-1}\}$. By Lemma 1, for each of the $2^{k-1} - 1$ internal nodes $\mathcal{I}_\mu$, the intersection points of all pairs of line segments of the left subtree leaves lie to the left of any point in $\mathcal{I}_\mu$. Thus, the "lowest" and "highest" line segments of the left subtree leaves bound all the points in $\mathcal{I}_\mu$. Similarly, the lowest and highest line segments of the right subtree leaves bound all the points in $\mathcal{I}_\mu$. The intersection of these four lowest/highest line segments gives four unique points in $\mathcal{I}_\mu$ whose convex hull is a *diamond* ($\diamond$) containing $\mathcal{I}_\mu$. (See Figures 6 and 7). The three leftmost vertices of this diamond are vertices of P. Thus the $2^{k-1} - 1$ internal nodes generate $3(2^{k-1} - 1)$ polygon vertices. Polygon P has 2^{k-1} vertices on the line $x = 0$ and two more vertices on the line $x = 1$, so the total number n of vertices of P is $2^{k+1} - 1$.

Interior Link Diameter. First, we show that for any pair of points q, q' in P, $\widehat{\text{Vis}}(q) \cap \widehat{\text{Vis}}(q') \neq \emptyset$. If $q \in \widehat{\text{Vis}}(u_i)$ and $q' \in \widehat{\text{Vis}}(u_j)$ for some u_i, u_j, then $\widehat{\text{Vis}}(q) \cap \widehat{\text{Vis}}(q') \neq \emptyset$, since $\widehat{\text{Vis}}(u_i)$ and $\widehat{\text{Vis}}(u_j)$ intersect for any u_i, u_j. If q (or q') doesn't belong to $\widehat{\text{Vis}}(u_i)$ for any u_i, then $e^* \in \widehat{\text{Vis}}(q)$ and hence, $\widehat{\text{Vis}}(q) \cap \widehat{\text{Vis}}(q')$ is not null. Thus, for any q, q' in P, $\widehat{\text{Vis}}(q) \cap \widehat{\text{Vis}}(q') \neq \emptyset$. Now, q can be connected to any other point q' by a line segment from q to $p \in \widehat{\text{Vis}}(q) \cap \widehat{\text{Vis}}(q')$ and then, by a line segment from p to q'. Thus, the interior link diameter of P is two.

Pairwise Disjoint Paths. For each internal node μ in the balanced binary tree

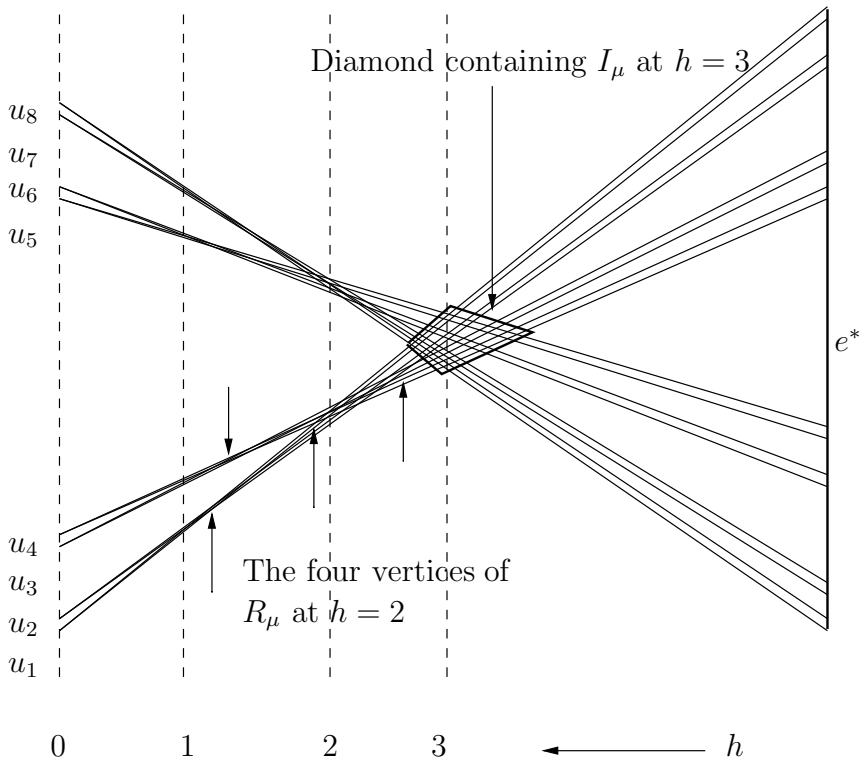

Fig. 7. Polygon P for $k = 4$, the diamond containing the intersection points in I_μ at $h = 3$, and the subpolygon R_μ (the four vertices of R_μ pointed by the arrows shown) for a node μ at height $h = 2$. The corresponding subpolygon R_5 (not explicitly shown) contains the edge e^* and the vertices u_5 to u_8.

$\mathcal{T}$, let q^l_μ and q^r_μ be the leftmost and rightmost points in $\mathcal{I}_\mu$. As shown in Figure 6, let R_μ be the simple polygon with vertices $\{q^l_{\mu.left}, q^r_\mu, q^l_{\mu.right}, q^l_\mu\}$, where $\mu.left$ and $\mu.right$ are the left and right children of μ. Let $\{\zeta_i\}$ be a set of pairwise disjoint paths connecting u_i to $u_{2^k-(i-1)}$ for $i = 1, \ldots, 2^{k-2}$. Let h be the height of μ in the tree $\mathcal{T}$. In the next paragraph, we will show that R_μ must contain at least 2^{h-2} path vertices of $\{\zeta_i\}$ when $1 < h < (k-1)$, at least 2^{k-2} path vertices when $h = k-1$, and at least one path vertex when $h = 1$. Assume k is even. Since the polygons R_μ where μ has odd height are pairwise disjoint and there are 2^{k-1-h} polygons R_μ at height h, we find that the polygons R_μ where μ has odd height together contain at least $2^{k-2} + ((k-6)/2+1)2^{(h-2)+(k-1-h)} + 2^{k-2} = (k+4)2^{k-4}$ path vertices. Thus, the pairwise disjoint paths $\{\zeta_i\}$ contain at least $(1/32)n \log_2 n$ line segments. For odd values of k, a similar argument gives an $\Omega(n \log n)$ lower bound with a somewhat lower constant.

Minimum number of vertices of $\{\zeta_i\}$ in R_μ. To complete the proof, we now show that R_μ must contain at least 2^{h-2} path vertices of $\{\zeta_i\}$ when $1 < h < (k-1)$, at least 2^{k-2} path vertices when $h = k-1$, and at least one path vertex when $h = 1$. Deleting polygon R_μ from polygon P divides P into five pieces, four to the left and

one to the right. Label the regions to the left R_1, R_2, R_3, R_4, from bottom to top, and the region to the right R_5.

— When $h = k - 1$. In this case, μ is the root of $\mathcal{T}$, as is the case in Figure 6. If ζ_i has an endpoint in R_1, then the other endpoint of ζ_i must be in R_4. Such a path ζ_i connecting a point in R_1 to a point in R_4 must contain a vertex in R_μ, since any segment in P from R_1 to R_5 crosses (in R_μ) any segment from R_4 to R_5. Thus, the polygon R_μ must contain a vertex of each of the 2^{k-3} paths connecting R_1 to R_4. Similarly, R_μ must contain a vertex of each of the 2^{k-3} paths connecting R_2 to R_3. Thus, R_μ contains at least 2^{k-2} path vertices.

— When $1 < h < k - 1$. Any line segment in P with endpoints in R_1 and R_5 must intersect any line segment in P with endpoints in R_4 and R_5. See Figure 7. Thus, if ζ_i and ζ_j are pairwise disjoint paths connecting R_1 and R_4 to R_5 respectively, then either ζ_i or ζ_j has a vertex in R_μ. Now, if ζ_i has an endpoint in R_1, R_2, R_3 or R_4, then the other endpoint of ζ_i must be in R_5. Since 2^{h-2} paths have endpoints in R_1 and 2^{h-2} paths have endpoints in R_4, polygon R_μ contains at least 2^{h-2} vertices.

— When $h = 1$. If μ is an internal node of height one, then two paths ζ_i and ζ_{i+1} must cross R_μ. By reasoning similar to that given above, R_μ contains a vertex (other than the endpoints) of at least one of these paths.

■

4. APPROXIMATION ALGORITHM

In this section, we design an approximation algorithm for the PDLP problem. We develop our approximation algorithm using the following high-level steps.

(1) In Section 4.1, we motivate and define a function $\mathcal{F}$ for a partitioning of a given set of line segments on a real line. We state (proof in Section 5) that there is always a way to partition a given set of m line segments such that the value of $\mathcal{F}$ for the partition is at least $m/40$. Such a partition of segments is key to our design of an approximation algorithm for construction of disjoint paths in a polygon.

(2) Section 4.2 contains the core idea of our article. Here, we design an approximation algorithm for the problem of constructing disjoint paths from a given set of m vertices to a *distinguished edge* e^* in a simple polygon P. We refer to the above approximation algorithm as `PathsToEdge`. In Section 4.3, we use the `PathsToEdge` algorithm to design an approximation algorithm for the PDLP problem. The basic idea of `PathsToEdge` algorithm is as follows. First, we construct appropriate segments (segments visible from diagonals bounding the subpolygon $\widehat{\mathrm{V}}\mathrm{is}(e^*)$) on the distinguished edge e^*, and partition them using the result of Section 4.1. The obtained partitioning of segments is used to construct a subpolygon Γ in P that satisfies certain key properties. One of the properties allows construction of disjoint paths in Γ from e^* to required boundary chords (m in number) using $O(m)$ links; second property of Γ guarantees that at least $\Omega(m)$ links are needed to construct such disjoint paths. Disjoint paths are con-

structed recursively (and independently) in subpolygons obtained by deleting Γ from P.

(3) In Section 4.3, we use the `PathsToEgde` algorithm to design an approximation algorithm for the PDLP problem as follows. Given an instance of a PDLP problem, viz., a polygon P and a set of vertex pairs, we determine a triangle $t_{u,u'}$ for each vertex pair (u, u'), and then, associate u and u' with appropriate edges of the triangle $t_{u,v}$. We then use `PathsToEdge` algorithm to connect such edge-vertex pairs by pairwise disjoint paths. Finally, the endpoints (on the edges) of these paths are connected within the corresponding triangle.

4.1 The Function $\mathcal{F}$ – Partitioning of Visible Segments in an Edge-Visible Polygon

Consider a polygon P such that *every* point in P is clearly visible from a distinguished edge e^*, and let $\{u_1, u_2, \ldots, u_m\}$ be a subset of vertices in $(P - e^*)$ listed in clockwise order around P starting from e^*. Consider the *open non-null* line segments $\{s_i = \widehat{\mathrm{V}}\mathrm{is}(u_i) \cap\ e^*\}$ on the edge e^*. In this section, we show how to partition the line segments $\{s_i\}$ in a way that *helps* us construct disjoint paths from $\{u_i\}$ to e^* using near-optimal number of links. In particular, we define a function $\mathcal{F}$ such that a partitioning of $\{s_i\}$ segments with large $\mathcal{F}$ value provides a way to construct disjoint paths from $\{u_i\}$ to e^* using near-optimal number of links. The algorithm for such a partitioning of $\{s_i\}$ is key to the design of our approximation algorithm for the PDLP problem. As motivation, we consider two extremely specialized cases based on the relative positions of the $\{s_i\}$ line segments.

Fully Ordered Case. Assume that the midpoints of the line segments $s_1, \ldots, s_m$ lie in counter-clockwise order[3] around P. Then, each u_i could be connected by a single line segment to the midpoint of s_i, and these connecting line segments would be pairwise disjoint. Actually, this condition is far too strong. As shown in Figure 8 (a), we need to only choose some point $\gamma_i \in s_i$, not necessarily a midpoint, such that the points $\gamma_1, \ldots, \gamma_m$ lie in counter-clockwise order around P. In particular, if the line segments $\{s_i\}$ share some common interval, then the points $\{u_i\}$ could be connected by disjoint line segments to points in the common interval.

Two Partitions in Reverse Order. Consider the case, where there is a point γ that separates $\{s_1, \ldots, s_{\lfloor m/2 \rfloor}\}$ from $\{s_{\lfloor m/2 \rfloor+1}, \ldots, s_m\}$ such that the segments from $\{s_{\lfloor m/2 \rfloor+1}, \ldots, s_m\}$ precede γ which precedes $\{s_1, \ldots, s_{\lfloor m/2 \rfloor}\}$ in counter-clockwise order around P, as shown in Figure 8 (b). Let $\zeta_1, \ldots, \zeta_m$ be a set of pairwise disjoint paths connecting $u_1, \ldots, u_m$ to e^*. The endpoints of $\zeta_1, \ldots, \zeta_m$ on e^* must lie in counter-clockwise order around P. Now, either the endpoints of $\{\zeta_1, \ldots, \zeta_{\lfloor m/2 \rfloor}\}$ on e^* are separated by γ from $\{s_1, \ldots, s_{\lfloor m/2 \rfloor}\}$, or the endpoints on e^* of $\{\zeta_{\lfloor m/2 \rfloor+1}, \ldots, \zeta_m\}$ are separated by γ from $\{s_{\lfloor m/2 \rfloor+1}, \ldots, s_m\}$. Thus, the $\{\zeta_i\}$ paths must contain at least $\lfloor m/2 \rfloor$ bends/turns.

Functions $f(\gamma, \mathcal{S})$ and $\mathcal{F}(\gamma, \mathcal{S}, \mathcal{S}')$. Of course, the arrangement of segments $\{s_i\}$ could be much more complicated than the simple cases above. We may need to

[3]Throughout this article, we consider counter-clockwise order on e^*, and clockwise order around the rest of P.

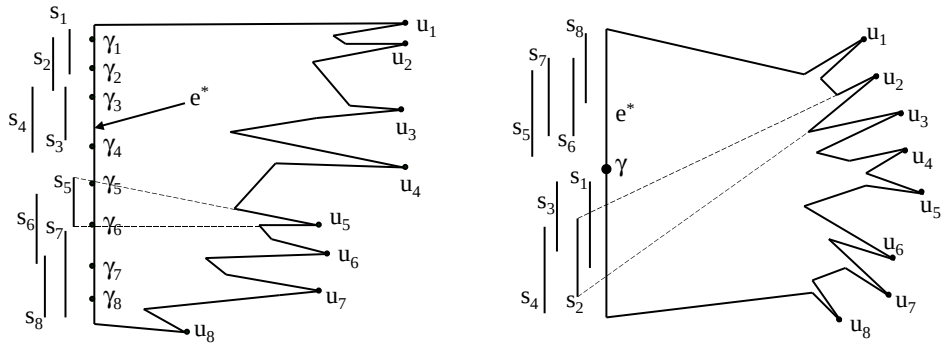

Fig. 8. Points u_i lying in a clockwise order around P and segments $s_i = \widehat{\text{Vis}}(u_i) \cap e^*$ (s_5 and s_2 illustrated in (a) and (b) respectively). (a) Fully ordered case. (b) Two partitions in reverse order.

partition the sequence $(s_1, \ldots, s_n)$ into many subsequences. Below, we define functions f and $\mathcal{F}$ that help measure the usefulness of a partition in construction of disjoint paths with near-optimal number of links.

DEFINITION 6. (Functions $f(\gamma, \mathcal{S})$ and $\mathcal{F}(\gamma, \mathcal{S}, \mathcal{S}')$.) Consider a polygon P with a distinguished edge e^*. Let $\mathcal{S}$ be a set of open non-null line segments on e^*. The line segments may not be distinct, and in the extreme case, a line segment may be a mere point. Let the endpoints of e^* be q_0^* and q_1^* in counter-clockwise order. For each point $\gamma \in e^*$, define

— $f(\gamma, \mathcal{S})$ as the number of line segments of $\mathcal{S}$ that contain the point γ,
— $f^-(\gamma, \mathcal{S})$ as the number of line segments of $\mathcal{S}$ that fully precede γ in the counter-clockwise order around P, i.e., the number of line segments of $\mathcal{S}$ that are contained in the open interval (q_0^*, γ).
— $f^+(\gamma, \mathcal{S})$ as the number of line segments of $\mathcal{S}$ that fully follow γ in the counter-clockwise order around P, i.e., the number of line segments of $\mathcal{S}$ that are contained in the open interval (q_1^*, γ).

For example, in Figure 8(a), if $\mathcal{S} = \{s_1, s_2, \ldots, s_8\}$, then $f(\gamma_5, \mathcal{S}) = 1, f^-(\gamma_5, \mathcal{S}) = 4, f^+(\gamma_5, \mathcal{S}) = 3$. Note that $f(\gamma, \mathcal{S}) + f^-(\gamma, \mathcal{S}) + f^+(\gamma, \mathcal{S})$ equals $|\mathcal{S}|$.

For a pair of sets of line segments $\mathcal{S}$ and $\mathcal{S}'$ on e^*, we define $\mathcal{F}(\gamma, \mathcal{S}, \mathcal{S}')$ as

$$\mathcal{F}(\gamma, \mathcal{S}, \mathcal{S}') = f(\gamma, (\mathcal{S} \cup \mathcal{S}')) + \min(f^+(\gamma, \mathcal{S}), f^-(\gamma, \mathcal{S}')).$$

□

General Partition of Segments. Let $\mathcal{S} = (s_1, \ldots, s_j)$ and $\mathcal{S}' = (s_{j+1}, \ldots, s_m)$ be a partition of $(s_1, \ldots, s_m)$ into two *contiguous* subsequences. Then, $f(\gamma, \mathcal{S} \cup \mathcal{S}')$

is the number of vertices u_i that lie on the boundary of $\widehat{\text{Vis}}(\gamma)$. By using two line segments[4] per path, these $f(\gamma, \mathcal{S} \cup \mathcal{S}')$ vertices can be connected by pairwise disjoint paths to e^*. On the other hand, the value $\min(f^+(\gamma, \mathcal{S}), f^-(\gamma, \mathcal{S}'))$ is a lower bound on the number of the bends in the disjoint paths connecting $\{u_i\}$ to e^*. Thus if $\mathcal{F}(\gamma, \mathcal{S}, \mathcal{S}')$ is large, then γ can either be used to connect many near-optimal paths consisting of exactly two line segments or it can be used to show a lower bound on the number of bends required.

However, there may be no single point γ that produces sufficiently large $\mathcal{F}(\gamma, \mathcal{S}, \mathcal{S}')$. Instead, we may need to partition the sequence $(s_1, \ldots, s_m)$ into many pairs of contiguous subsequences $\{(\mathcal{S}_j, \mathcal{S}'_j)\}$, each pair associated with a γ_j, such that the sum of $\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j)$ is large. Lemma 2 below presents such a partition algorithm.

Partitioning Lemma. Since the partition algorithm applies to arbitrary segments in $\mathbb{R}^1$, we state the lemma without the context of a polygon. In the context of $\mathbb{R}^1$, $f^-(\gamma, \mathcal{S})$ and $f^+(\gamma, \mathcal{S})$ are the number of line segments of $\mathcal{S}$ that are contained in the open intervals $(-\infty, \gamma)$ and $(\gamma, +\infty)$ respectively. We will prove the lemma in Section 5.

DEFINITION 7. (Point $g(\mathcal{S})$, the Median of Line Segments' Midpoints) Let $\mathcal{S}$ be a set of open line segments (not necessarily distinct) in $\mathbb{R}^1$. Let $\mathcal{M}$ be the list of midpoints (ordered in increasing order on $\mathbb{R}^1$) of line segments in $\mathcal{S}$. The point $g(\mathcal{S})$ is defined as the $\lceil |\mathcal{M}|/2 \rceil^{th}$ point (median point) in $\mathcal{M}$. □

LEMMA 2. *Let $\{s_1, \ldots, s_m\}$ be a set of open non-null line segments on $\mathbb{R}^1$. In $O(m \log m)$ time, the sequence $(s_1, s_2, \ldots, s_m)$ can be partitioned into $2h$ (for some $h > 0$) non-empty* contiguous[5] *subsequences $\mathcal{S}_1, \mathcal{S}'_1, \mathcal{S}_2, \mathcal{S}'_2, \ldots, \mathcal{S}_h, \mathcal{S}'_h$ viz. $\mathcal{S}_1 = (s_1, s_2, \ldots, s_{i_1})$, $\mathcal{S}'_1 = (s_{i_1+1}, s_{i_1+2}, \ldots, s_{i_2}), \mathcal{S}_2 = (s_{i_2+1}, s_{i_2+2}, \ldots, s_{i_3}), \ldots, \mathcal{S}'_h = (s_{i_{2h-1}+1}, \ldots, s_m)$, such that:*

(1) $|\mathcal{S}_j| = |\mathcal{S}'_j|$ *or* $|\mathcal{S}_j| = |\mathcal{S}'_j| + 1$,

(2) $\gamma_1, \gamma_2, \ldots, \gamma_h$ *lie in increasing order on $\mathbb{R}^1$ (counter-clockwise order on e^* in the context of a polygon P), where γ_j is $g(\mathcal{S}_j \cup \mathcal{S}'_j)$, the median of the midpoints of line segments in $\mathcal{S}_j \cup \mathcal{S}'_j$,*

(3) $\sum_{j=1..h} \mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq m/40$.

■

Significance of Lemma 2. We use the above partitioning algorithm over $\{s_i\}$ segments on e^* to construct near-optimal disjoint paths in a polygon P visible from e^* as follows. We start with constructing a subpolygon Γ (composed of triangles of a triangulation of P) with the following two properties. First, any set of m disjoint paths connecting e^* to the appropriate boundary edges of Γ must contain at least $\sum_j F(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j)$ ($\geq m/40$ by Lemma 2) bends. Second, we can construct disjoint paths connecting e^* to the appropriate boundary edges of Γ using 3 line

[4]We need two line segments to avoid intersection at γ.

[5]We require the subsequences to be contiguous for the working of our approximation algorithm. In particular, contiguity of the subsequences is required to construct disjoint line segments $\{\lambda_1, \ldots, \lambda_m\}$ and polygonal paths in Γ (see Lemma 3).

segments per path in linear time. (Lemma 4 and Lemma 3 in Section 4.2 prove the above two properties respectively for an appropriately constructed subpolygon Γ.) Recursively constructing disjoint paths in the subpolygons obtained by deletion of Γ from P, we can construct full disjoint paths connecting $\{u_i\}$ to e^* using $120M$ line segments, where M is the minimum number of links required. In the next subsection, we extend the above ideas to design an approximation algorithm for constructing disjoint paths to a distinguished edge e^* in a simple (not necessarily edge-visible) polygon P.

4.2 Constructing Paths to a Polygon Edge

We now present an approximation algorithm to construct pairwise disjoint paths to a distinguished edge e^* in a simple polygon P. For the discussion in this section, let T_P be a triangulation of polygon P using no vertices other than those of P. We use the term *diagonal* or *chord* to refer to a triangulation edge of T_P. Also, a *subpolygon* in P is a union of connected triangles of T_P. We start with a few definitions.

DEFINITION 8. (Separating Diagonals or Triangles.) Let each u_1 and u_2 be either a point in P, an edge of triangulation T_P, or a triangle of T_P. A diagonal or a triangle t *separates* u_1 from u_2 if every path that connects u_1 to u_2 through the interior of P must intersect t. □

DEFINITION 9. (First Separating Diagonals.) Let u be a point in P. A diagonal d of triangulation T_P *first separates* u from an edge e^* if d lies on a triangle containing u and separates u from e^*. It can be shown by induction on $|P|$ that d is unique for any given e^* and u, as long as u is not in the triangle (of T_P) containing e^*. □

DEFINITION 10. (Connectors.) Consider a polygon P with a distinguished edge e^*, a point u_i on the boundary of P, and a subpolygon Γ of P such that $e^* \in \Gamma$.

The *connector* in Γ for u_i is denoted as ξ_i and is defined as follows. If $u_i \in \Gamma$, then $\xi_i = u_i$, else ξ_i is the chord c on the boundary of Γ that separates u_i from e^*. Note that any path connecting e^* to u_i must intersect ξ_i. □

Below, we present the *PathsToEdge* Algorithm that constructs pairwise disjoint interior paths connecting a set of vertices in a polygon P to the interior of an edge e^* in P using $O(M)$ line segments where M is the optimal number of line segments required.

PathsToEdge Algorithm

Given: Let P be a simple polygon on n vertices with distinguished edge e^*. Let $\{u_1, u_2, \ldots, u_m\}$ be a subset of vertices of P labeled in clockwise order around P starting at e^*. We assume that no u_i is an endpoint of e^*.

Output: A set of pairwise disjoint paths connecting the points $\{u_i\}$ to e^* through the interior of P, using at most $120M$ total links where M is the optimal number of links required.

Basic Idea. As suggested in the previous subsection, the basic idea of the PathsToEdge Algorithm is to construct a subpolygon Γ containing e^* such that it satisfies the following two properties. Here, ξ_i is the connector in Γ for the point u_i.

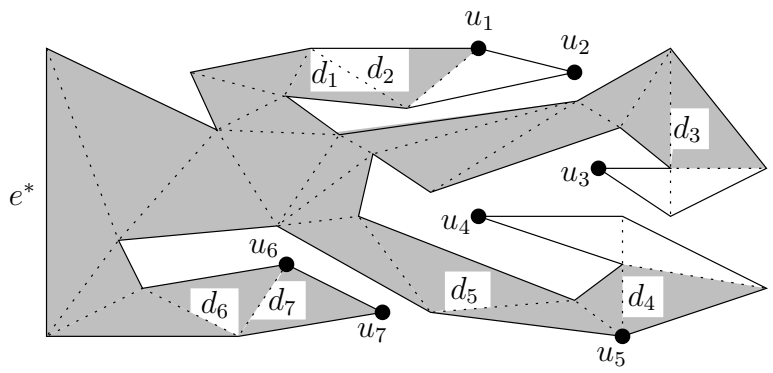

Fig. 9. The triangles of T_P visible from e^*. The diagonal d_i is the farthest diagonal visible from e^* that separates u_i from e^*.

(1) Using $O(m)$ line segments, we can construct a set of disjoint paths that connect e^* to $\{\xi_i\}$.

(2) Any set of disjoint paths connecting e^* to $\{\xi_i\}$ uses at least $\Omega(m)$ bends. Here, we count each end point u_i as one bend too.

Let $\{P_c\}$ be the set of subpolygons formed by deleting such a Γ from P. Once such a subpolygon Γ has been constructed, we can recursively and independently construct disjoint paths in each subpolygon P_c connecting $\{u_i\}$ points to the boundary of Γ. See Figure 13. The second property of Γ will ensure the near-optimality of the number of links used by our recursive algorithm.

The intuition behind the construction of the subpolygon Γ that satisfies both the properties is as follows. Observe that $\Gamma = \widehat{\text{Vis}}(e^*)$ satisfies the second property (since if $u_i \notin \Gamma$, then no point in ξ_i is visible from e^*), but may not satisfy the first property (consider polygon P of Section 3 as Γ). On the other hand, $\Gamma = \widehat{\text{Vis}}(\gamma)$ for some $\gamma \in e^*$ satisfies the first property (since each ξ_i can be connected to a point in the neighborhood of γ using two links), but not the second property. However, if $\mathcal{S}$ and $\mathcal{S}'$ is a partition of appropriate segments on e^* such that $\mathcal{F}(\gamma, \mathcal{S}, \mathcal{S}') = \Omega(m)$, then it can be shown (as in Lemma 4) to satisfy the second property also. More generally, we construct an appropriate segment $s_i \in e^*$ for each u_i, and use Lemma 2 to partition the segments $\{s_i\}$. Using the resulting points $\gamma_1, \ldots, \gamma_h$ on e^*, we construct Γ as the union of triangles that are visible from some γ_j and separate a corresponding u_i from e^*. We will show that such a subpolygon Γ satisfies both the above mentioned properties. A full description of the PathsToEdge Algorithm follows.

Algorithm Description

Throughout the description, we use the polygon of Figure 9 as our running example to illustrate various steps of PathsToEdge Algorithm. We start with constructing a triangulation T_P of P. For now, lets assume that no u_i belongs to the triangle containing e^*.

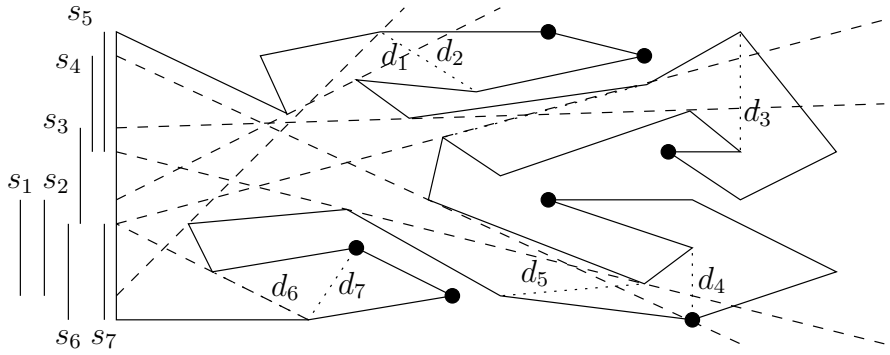

Fig. 10. Line segments $s_i = \text{int}(e^*) \cap \widehat{\text{V}}\text{is}(d_i)$.

Determine Diagonals d_i. For each u_i, let d_i be the "farthest" diagonal visible from e^* that separates u_i from e^*. More formally, consider the set D of diagonals that are visible from e^* and separate u_i from e^*. The diagonal d_i is the "farthest" diagonal in D from e^*, i.e., d_i is such that every other diagonal in D separates d_i from e^*. See Figure 9.

Determine Segments s_i; Apply Lemma 2. Let $s_i = \text{int}(e^*) \cap \widehat{\text{V}}\text{is}(d_i)$. Each s_i is a non-null open line segment[6] lying on e^*. See Figure 10. Now, use Lemma 2 to partition the sequence $(s_1, \ldots, s_m)$ of line segments into $2h$ non-empty contiguous subsequences $\mathcal{S}_1, \mathcal{S}'_1, \mathcal{S}_2, \mathcal{S}'_2, \ldots, \mathcal{S}_h, \mathcal{S}'_h$, and let $\gamma_1, \ldots, \gamma_h$ be the points in counterclockwise order on e^* as derived in Lemma 2. Finally, define

$$\mathcal{U}_j = \{u_i : s_i \in (\mathcal{S}_j \cup \mathcal{S}'_j)\}.$$

Construct Subpolygon Γ. We construct a subpolygon Γ as follows. A triangle t of T_P is in Γ iff it intersects $\widehat{\text{V}}\text{is}(\gamma_j)$ and separates a point in $\mathcal{U}_j$ from e^* for some $1 \leq j \leq h$. See Figure 11 - 13, where we assume that $\mathcal{U}_1 = \{u_1, u_2, \ldots, u_5\}$ and $\mathcal{U}_2 = \{u_6, u_7\}$. We will show that the above defined subpolygon Γ allows us to construct disjoint paths from e^* to connectors $\{\xi_i\}$ (of $\{u_i\}$ in Γ) using total $3m$ line segments, while an optimal solution must use at least $m/40$ bends (including the end points u_i; see Lemma 4). Recall that m is the total number of u_i vertices.

Define Subpolygons P_c; Apply Recursion. Let chord c be a triangulation edge of T_P that bounds Γ. Each such chord separates P into two subpolygons. Let P_c be the subpolygon not containing Γ. Recursively, construct pairwise disjoint paths connecting the $\{u_i : u_i \in (P_c - c)\}$ points to c. See Figure 13.

Construct Paths in Γ. For each point $u_i \in (P_c - c)$ for some c, let u_i^0 be the endpoint on c of the path connecting u_i to c. For all other points u_i, which are in Γ, let $u_i^0 = u_i$. We now show how to construct pairwise disjoint paths from points $\{u_i^0\}$ on the boundary of Γ to the edge e^* using three line segments each. We start with an informal description of the construction of paths.

[6]Note that s_i may equal s_j (and d_i may equal d_j) for many distinct points u_i, u_j.

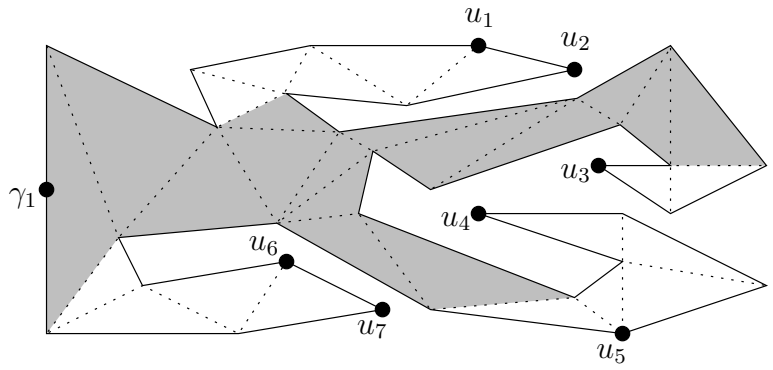

Fig. 11. Triangles in T_P that intersect $\widehat{\text{Vis}}(\gamma_1)$ and separate some point in $\mathcal{U}_1 = \{u_1, u_2, \ldots, u_5\}$ from e^*.

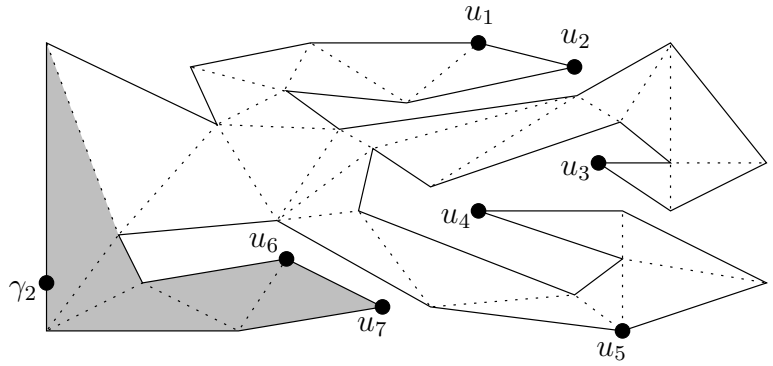

Fig. 12. Triangles in T_P that intersect $\widehat{\text{Vis}}(\gamma_2)$ and separate some point in $\mathcal{U}_2 = \{u_6, u_7\}$ from e^*.

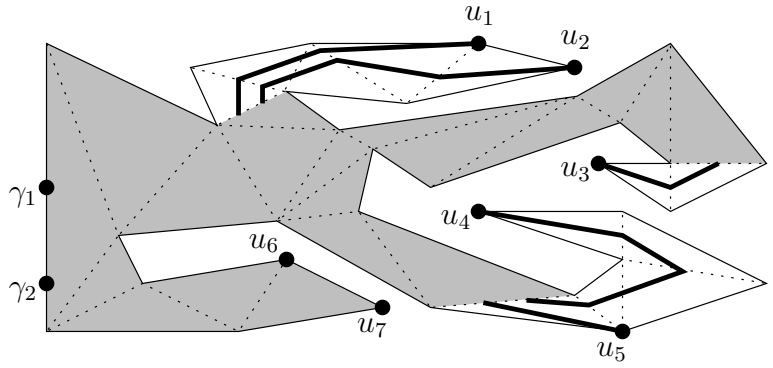

Fig. 13. Partition of P into Γ (shaded and formed by the union of triangles shown in Figure 11 and Figure 12) and three subpolygons P_c (non-shaded), and (recursively constructed) paths in P_c to the boundary of Γ.

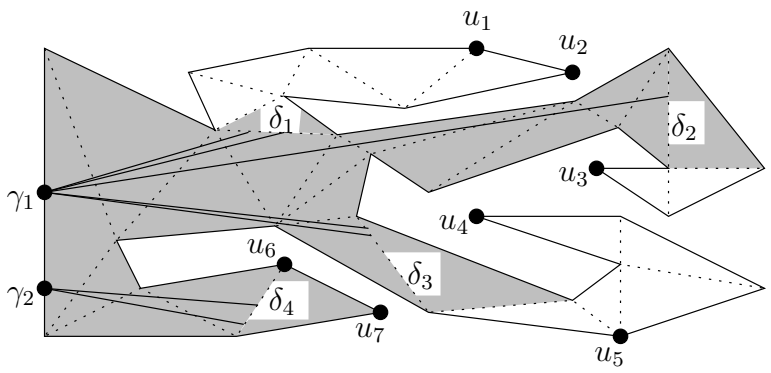

Fig. 14. Diagonals $\{\delta_1, \ldots, \delta_l\}$, which separate a single triangle of Γ from e^*, and line segments $\{\lambda_1, \ldots, \lambda_m\}$.

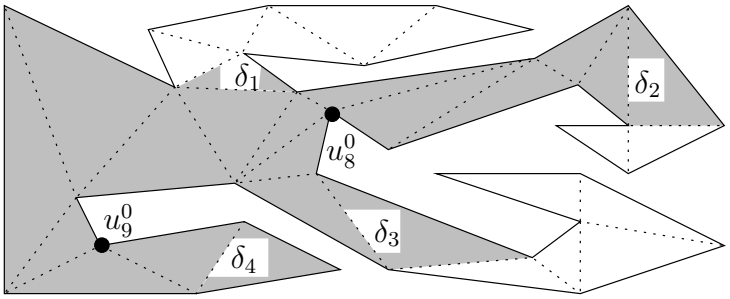

Fig. 15. Illustrating the `responsible` function. Here, δ_3 is `responsible` for u_8^0, while δ_4 is `responsible` for u_9^0.

Basic Idea. Let $\{\delta_1, \delta_2, \ldots, \delta_l\}$ be the diagonals that separate from e^* a single triangle of Γ as well as some point u_i. The diagonals are labeled in clockwise order around P starting from e^*. See Figure 14. For simplicity, lets assume that none of the given u_i vertices is in Γ, i.e., each of the u_i^0 point is on a *chord* c of T_P (and *not* on an edge of the polygon P) bounding Γ. For such a case, construction of disjoint paths from $\{u_i^0\}$ to e^* can be done in a relatively straightforward way as follows. First, we constuct total m disjoint line segments from $\{\delta_1, \ldots, \delta_l\}$ to the points $\{\gamma_1, \ldots, \gamma_h\}$ on e^*, with appropriate number of segments from each δ_k to some γ_j. See Figure 14. Each u_i^0 can be easily connected to one of the above m segments, since each u_i^0 lies in a triangle containing a diagonal δ_k. See Figure 16. The above construction of paths can be generalized to the case when some of the u_i vertices may be in Γ; we formally describe the general case below.

Formal Construction. To generalize the above idea, we need to define a term and introduce some additional notations.

— *Defining* `Responsible`. Let d_i^0 be the diagonal that first separates u_i^0 from e^*. For each u_i^0, we designate a certain δ_k, that is separated by d_i^0 from e^*

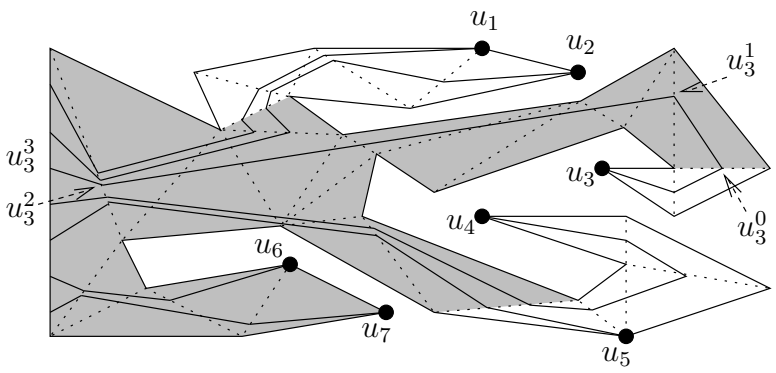

Fig. 16. Pairwise disjoint paths connecting $\{u_i\}$ to e^*.

and is "closest" to u_i^0, as `responsible` for constructing a path from u_i^0 to e^*. More formally, if $u_i^0 \in \delta_k$ or $d_i^0 = \delta_k$ for some k, then δ_k is designated as `responsible` for u_i^0. Otherwise, among the diagonals δ_k that are separated by d_i^0 from e^*, pick the one that is closest (preceding or following) to u_i^0 in the clockwise order around Γ starting at e^* as the `responsible` diagonal. Note that such a δ_k exists (see Lemma 3). See Figure 15.

— *Constructing Segments from* $\{\delta_1, \ldots, \delta_l\}$ *to* e^*. For each diagonal δ_k in $\{\delta_1, \ldots, \delta_l\}$, choose the minimum j such that δ_k is clearly visible to γ_j and separates a point in $\mathcal{U}_j$ from e^*. Since, δ_k is a diagonal of Γ, such a j exists. Construct R distinct line segments connecting the interior of δ_k to γ_j in the interior of Γ, where R is the number of u_i^0 points for which δ_k is `responsible` as defined above. A total of m line segments are thus constructed. Label these line segments $\lambda_1, \lambda_2, \ldots, \lambda_m$, as their endpoints appear in clockwise order around P starting at e^*. See Figure 14. We show in Lemma 3 that the line segments $\{\lambda_1, \ldots, \lambda_m\}$ intersect only at their endpoints.

— *Constructing Paths From* $\{u_i^0\}$ *to* e^*. Let u_i^1 be $d_i^0 \cap \lambda_i$, where d_i^0 (as defined before) is the diagonal that first separates u_i^0 from e^*. Lemma 3 observes that u_i^1 exists. Let u_i^2 be the intersection of λ_i with an edge (other than e^*) of the triangle containing e^*. Place m points equally spaced on e^*; let u_i^3 be the i^{th} such point on e^*, ordered counter-clockwise around P. Now, connect u_i^0 to e^* using a path of three links connecting the points $u_i^0, u_i^1, u_i^2, u_i^3$. See Figure 16. We show in Lemma 3 that these polygonal paths do not intersect.

Finally, to relax the assumption made earlier that no u_i is in the triangle containing e^*, we can now connect such a vertiex u_i to e^* using at most 2 line segments.

This completes the description of PathsToEdge Algorithm. In the following lemmas, we prove the correctness (Lemma 3), near-optimality of the output size (Lemma 5), and time complexity of the PathstoEdge Algorithm (Lemma 6).

LEMMA 3. *PathsToEdge Algorithm constructs pairwise disjoint paths connecting* $\{u_i\}$ *to* e^* *in the interior of* P. *Note that PathsToEdge Algorithm uses 3 line segments per path to connect* e^* *to* u_i^0.

Proof: Since the subpolygons P_c and Γ are mutually disjoint, it suffices to show that the polygonal path connecting $\{u_i^0\}$ to e^* do not intersect, i.e., for any $i \neq j$, $(u_i^0, u_i^1, u_i^2, u_i^3)$ doesn't intersect any other polygonal path $(u_j^0, u_j^1, u_j^2, u_j^3)$. Below, we show the above in parts.

Disjointness of $\{\lambda_1, \ldots, \lambda_m\}$. We claim that the line segments $\{\lambda_1, \ldots, \lambda_m\}$ connecting the diagonals $\{\delta_1, \ldots, \delta_l\}$ to e^* intersect only at their endpoints. Let λ_i and $\lambda_{i'}$ be line segments connecting distinct γ_j and $\gamma_{j'}$ to diagonals δ_k and $\delta_{k'}$ respectively. Without loss of generality, assume that j is less than j'. Note that diagonal δ_k separates some point of $\mathcal{U}_j$ from e^*, and diagonal $\delta_{k'}$ separates some point of $\mathcal{U}_{j'}$ from e^*. Since j is less than j', points $\gamma_j, \gamma_{j'}$ lie in counter-clockwise order on e^*, while $\mathcal{U}_j, \mathcal{U}_{j'}$ lie in clockwise order around P (since $\{\mathcal{U}_1, \ldots, \mathcal{U}_h\}$ are contiguous subsequences). Thus, λ_i and $\lambda_{i'}$ don't intersect.

Existence of u_i^1. Consider d_i^0, the diagonal that first separates u_i^0 from e^*. Either $d_i^0 = \delta_k$ for some k, or d_i^0 separates more than one triangle from e^*. In the latter case, d_i^0 separates one or more diagonal in $\{\delta_1, \ldots, \delta_l\}$ from e^*. Thus, for each u_i^0, a diagonal δ_k exists that can be designated as `responsible`. Since diagonals $\{\delta_1, \ldots, \delta_l\}$ and segments $\{\lambda_1, \ldots, \lambda_m\}$ are ordered clockwise around P and by the definition of `responsible`, we see that λ_i does originate from the diagonal δ_k `responsible` for u_i^0. Thus, $\lambda_i \cap d_i^0 = u_i^1$ exists.

Disjointness of $\{(u_i^0, u_i^1)\}$. We now claim that the line segment (u_i^0, u_i^1) does not intersect any (u_j^0, u_j^1) for $i \neq j$. Without loss of generality, let us assume that $i < j$. If (u_i^0, u_i^1) intersects (u_j^0, u_j^1), then u_i^0 and u_j^0 lie in the same triangle t of T_P and $d_i^0 = d_j^0$. Now, in the triangle t, u_i^0 and u_j^0 lie in clockwise order, while u_i^1 and u_j^1 lie in counter-clockwise order. Thus, (u_i^0, u_i^1) doesn't intersect (u_j^0, u_j^1).

Disjointness of (u_i^0, u_i^1) and λ_j. Let us show that the line segment (u_i^0, u_i^1) does not intersect any λ_j for $i < j$. Without loss of generality, let us assume that $i < j$. Suppose λ_j originates from δ_k. If diagonal δ_k separates u_i^0 from e^*, then (u_i^0, u_i^1) and λ_j are trivially disjoint (since δ_k separates them from each other). Let us assume that δ_k doesn't separate u_i^0 from e^*. Then, δ_k fully follows u_i^0 in clockwise order around Γ (since $i > j$). On the other hand, intersection of λ_i and λ_j on d_i^0, i.e., u_i^1 and $(\lambda_j \cap d_i^0)$, lie in a counter-clockwise order (since λ_i intersects λ_j only at e^*, and $i < j$). Now, since (u_i^0, u_i^1) lies completely in the triangle containing d_i^0 and separated by d_i^0 from e^*, (u_i^0, u_i^1) does not intersect λ_j.

Conclusion: Disjointness of full paths. Since $(u_i^1, u_i^2) \subseteq \lambda_i$, and line segments $\{\lambda_1, \ldots, \lambda_m\}$ are disjoint, (u_i^1, u_i^2) also doesn't intersect any (u_i^1, u_j^2) for $i \neq j$. Finally, note that all the line segments (u_i^2, u_i^3) lie in a single triangle of T_P that doesn't contain any other line segments. Thus, the polygonal path $(u_i^0, u_i^1, u_i^2, u_i^3)$ doesn't intersect any other polygonal path $(u_j^0, u_j^1, u_j^2, u_j^3)$ for $i \neq j$. ∎

The above lemma shows that the constructed subpolygon Γ satisfies its first desired property. In particular, it proves that the correctness of the PathsToEdge Algorithm. The following lemma shows that Γ also satisfies the second desired property, i.e., any set of disjoint paths connecting e^* to $\{\xi_i\}$ uses at least $\Omega(m)$ bends. We will use the following lemma to show the near-optimality of the PathsToEdge Algorithm (Lemma 5).

LEMMA 4. *Any set of pairwise disjoint paths connecting e^* to connectors $\{\xi_i\}$ in Γ must use at least $m/40$ bends. As mentioned before, we count the end point u_i as one bend too.*

Proof: Let $\{\eta_i\}$ be the optimal set of disjoint paths connecting e^* to $\{\xi_i\}$. Below, we restrict our attention to paths $\{\eta_i\}$ that correspond to points u_i such that the corresponding line segment s_i is in $\mathcal{S}_j$, and consider two cases based on the location of line segment s_i with respect to γ_j on e^*.

When $\gamma_j \in s_i$. For this case, we show that η_i contains at least one bend. If $u_i \in \Gamma$, then η_i trivially contains one bend (since we count the end point u_i as one bend). Thus, let us assume that $u_i \notin \Gamma$. Now, the diagonal d_i is clearly visible from γ_j (since $s_i = e^* \cap \widehat{\mathrm{V}}\mathrm{is}(d_i)$), and hence, d_i is a diagonal of Γ. Now, d_i separates u_i from e^* (by definition of d_i), ξ_i separates u_i from e^* (since $u_i \notin \Gamma$), and d_i separates ξ_i from e^* (as d_i is a diagonal of Γ and ξ_i is on Γ's boundary). Thus, by definition of d_i, ξ_i is *not* visible from e^*, and the path η_i from e^* to ξ_i contains at least one bend.

When $s_i \in \mathcal{S}_j$ fully follows γ_j counter-clockwise. For this case, consider another point $u_{i'}$ such that $s_{i'} \in \mathcal{S}'_j$ and $s_{i'}, \gamma_j, s_i$ lie in counter-clockwise order around P. Here, neither s_i nor $s_{i'}$ contain γ_j. Note that u_i and $u_{i'}$ lie in clockwise order around P, and hence, ξ_i and $\xi_{i'}$ lie in clockwise order around Γ. See Figure 17. We show that either η_i or $\eta_{i'}$ must contain at least one bend. If either u_i or $u_{i'}$ is in Γ, the claim is trivial. So, lets assume that $u_i, u_{i'} \notin \Gamma$. Thus, ξ_i and $\xi_{i'}$ are chords on the boundary of Γ. Since paths η_i and $\eta_{i'}$ are pairwise disjoint, either the endpoint of η_i on e^* must precede γ_j or endpoint of $\eta_{i'}$ on e^* must follow γ_j in the counter-clockwise order around P. Without loss of generality, let us assume that the endpoint ρ_i of η_i on e^* precedes γ_j in the counter-clockwise order as in Figure 17. In that case, γ_j lies between ρ_i and s_i. Let d be the farthest diagonal of P clearly visible from ρ_i and separating u_i from ρ_i. Since ρ_i is not clearly visible from d_i (as $\rho_i \notin s_i$), diagonal d must separate d_i from e^* and hence, d is clearly visible to s_i. Now, since γ_j lies between ρ_i and s_i, diagonal d is also clearly visible to γ_j and hence, d is a diagonal of Γ (since, d also separates $u_i \in \mathcal{U}_j$ from e^*). Thus, d also separates ξ_i from e^*, and hence, the path η_i (connecting ρ_i to ξ_i) must contain at least one bend (since, d is the farthest diagonal visible from ρ_i, and ξ_i is a diagonal in P). Similarly it can be shown that if the endpoint of $\eta_{i'}$ on e^* follows γ_j in the counter-clockwise order, the path $\eta_{i'}$ must contain at least one bend. Thus, it follows that either η_i or $\eta_{i'}$ must contain at least one bend.

By the analysis of the above two cases, the optimal set of paths $\{\eta_i\}$ connecting the connectors $\{\xi_i\}$ to e^* must contain at least

$$f(\gamma_j, \mathcal{S}_j) + f(\gamma_j, \mathcal{S}'_j) + \min(f^+(\gamma_j, \mathcal{S}_j), f^-(\gamma_j, \mathcal{S}'_j)) = \mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j)$$

bends. Here, as defined before, $f(\gamma_j, \mathcal{S}_j)$ is the number of line segments of $\mathcal{S}_j$ that contain the point γ_j, $f^+(\gamma_j, \mathcal{S}_j)$ is the number of line segments of $\mathcal{S}_j$ that follow γ_j, and $f^-(\gamma_j, \mathcal{S}'_j)$ is the number of line segments of $\mathcal{S}'_j$ that precede γ_j in the counter-clockwise order around P. Thus, the total number of bends contained in the optimal set of paths $\{\eta_i\}$ is at least $\sum_{j=1..h} \mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j)$ ($\geq m/40$, from Lemma 2). ∎

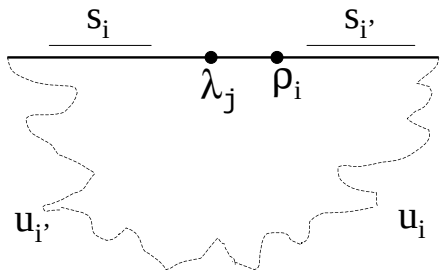

Fig. 17. The case when $s_i \in \mathcal{S}_j$, $s_{i'} \in \mathcal{S}'_j$, $s_{i'}, \gamma_j, s_i$ lie in counter-clockwise order, and the end point ρ_i of the path η_i precedes γ_j in the counter-clockwise order around P.

LEMMA 5. *The PathsToEdge Algorithm connects $\{u_i\}$ points to e^* using a total of at most $120M$ line segments where M is the minimum number of line segments used by an optimal solution.*

Proof: Consider the set $\{O_i\}$ of disjoint paths connecting e^* to $\{u_i\}$ in an optimal solution. Let $\eta_i \subseteq O_i$ be the *subpath* of O_i from e^* to ξ_i. From Lemma 4, we know that the subpaths $\{\eta_i\}$ contain at least $m/40$ bends. In contrast, our PathsToEdge Algorithm uses at most $3m$ line segments (and hence, bends) in Γ to connect $\{\xi_i\}$ to e^*. As each subpolygon P_c is disjoint from Γ, each path O_i is broken into only disjoint subpaths for the above comparison. Thus, the PathsToEdge solution size (in terms of bends or line segments) is at most 120 times the optimal size. ∎

LEMMA 6. *The PathstoEdge Algorithm runs in $O(n \log m + M \log m)$ time, where n is the size of the polygon P, m is the total number of points in $\{u_i\}$, and M is the total number of line segments in an optimal solution.*

Proof: Constructing the initial triangulation T_P takes $O(n)$ time and is done only once. Let n^* be the number of triangles in T_P that are visible from e^*.

Non-Recursive Steps. By Theorem 1, the visibility region $\widehat{\text{Vis}}(e^*)$, the diagonals d_i, and the segments s_i can all be constructed in $O(n^* + m)$ time. By Lemma 2, partitioning the set of segments $\{s_i\}$ takes $O(m \log m)$ time. The subpolygon Γ can be constructed in $O(n^*)$ time as follows. Using Theorem 1, we first construct $\text{int}(e^*) \cap \widehat{\text{Vis}}(d)$ for every diagonal d visible from e^* in $O(n^*)$ time. Now, a diagonal d visible from e^* is an internal diagonal of Γ if and only if d is visible from some point γ_j and separates a point in $\mathcal{U}_j$ from e^*, which can be determined in constant time per diagonal. Thus, Γ can be constructed in $O(n^*)$ time. Construction of $\{\delta_1, \ldots, \delta_l\}$, finding `responsible` diagonals, and the construction of the line segments $\{\lambda_1, \ldots, \lambda_m\}$ and the polygonal paths $\{(u_i^0, u_i^1, u_i^2, u_i^3)\}$ can all be done in $O(n^* + m)$ time. Thus, the non-recursive steps of the PathsToEdge Algorithm (apart from construction of T_P) take $k_1 n^* + k_2 m \log m$ time for some constants k_1, k_2.

Total Running Time. We show that the total running time of the algorithm is less than $k_0(n \log m + M \log m)$ for some constant k_0. Let $\mathcal{C}$ is the set of chords that bound Γ. For each subpolygon P_c, let n_c be the number of vertices of P_c,

$\mathcal{U}(c) = \{u_i : u_i \in (P_c - c)\}$, $m_c = |\mathcal{U}(c)|$, and M_c be the size of an optimal solution connecting $\mathcal{U}(c)$ to c. By induction, we have

$$\text{Total Running Time} = k_1 n^* + k_2 m \log m + \sum_{c \in \mathcal{C}} k_0 (n_c \log m_c + M_c \log m_c).$$

We partition the above expression into T_1 and T_2 as defined below.

$$\begin{aligned} T_1 &= k_2 m \log m + \sum_{c \in \mathcal{C}} k_0 (M_c \log m_c), \\ T_2 &= k_1 n^* + \sum_{c \in \mathcal{C}} k_0 (n_c \log m_c). \end{aligned}$$

We show that $T_1 \leq k_0 M \log m$ and $T_2 \leq k_0 n \log n$ for $k_0 > \max(40k_2, k_1/\log 1.5)$.

Bounding T_1. Since any optimal solution must have at least $m/40$ line segments contained in Γ, we have $m/40 + \sum_{c \in \mathcal{C}} M_c \leq M$. Since $k_0 > 40k_2$,

$$T_1 \leq (40k_2 m/40 + \sum_{c \in \mathcal{C}} k_0 M_c) \log m \leq k_0 M \log m.$$

Bounding T_2. Let Γ_* be the subpolygon (set of triangles) visible from e^*. The difficulty in bounding T_2 is that the triangles in $\Gamma - \Gamma_*$ may be processed multiple times during recursion. Fortunately, we can absorb the running time contribution of these triangles in the $\sum_{c \in \mathcal{C}} k_0 (n_c \log m_c)$ term.

Let $\mathcal{C}^*$ be the set of chords that bound Γ as well as Γ_*. Since, the sets of triangles in Γ_* and P_c for $c \in C^*$ are mutually disjoint, we have

$$n^* \leq n - \sum_{c \in \mathcal{C}^*} n_c = n - (\sum_{c \in \mathcal{C}} n_c - \sum_{c \in \mathcal{C} \setminus \mathcal{C}^*} n_c).$$

We show in Lemma 7 that m_c is at most $2m/3$ for each $c \in (\mathcal{C} \setminus \mathcal{C}^*)$. Thus, we get

$$\sum_{c \in \mathcal{C}} n_c \log m_c \leq \sum_{c \in \mathcal{C}} n_c \log m - \sum_{c \in \mathcal{C} \setminus \mathcal{C}^*} n_c \log 1.5.$$

Multiplying the above two equations by k_1 and k_0 respectively, adding the results, and rearranging terms, we get

$$\begin{aligned} T_2 &\leq k_1 (n - \sum_{c \in \mathcal{C}} n_c) + k_0 \sum_{c \in \mathcal{C}} n_c \log m + \sum_{c \in \mathcal{C} \setminus \mathcal{C}^*} n_c (k_1 - k_0 \log 1.5) \\ &\leq k_1 (n - \sum_{c \in \mathcal{C}} n_c) + k_0 \sum_{c \in \mathcal{C}} n_c \log m && \text{as } k_0 \log 1.5 \geq k_1 \\ &\leq k_0 \log m (n - \sum_{c \in \mathcal{C}} n_c) + k_0 \sum_{c \in \mathcal{C}} n_c \log m && \text{as } k_0 \log m \geq k_1 \\ &\leq k_0 n \log m. \end{aligned}$$

Thus, the total running time $(T_1 + T_2)$ of PathsToEdge Algorithm is $O(n \log m + M \log m)$. ∎

We now prove the claim used in the above Lemma 6.

LEMMA 7. *Using the notation from Lemma 6: For each* $c \in (\mathcal{C} \backslash \mathcal{C}^*)$, $|\mathcal{U}(c)| (= m_c)$ *is at most* $2m/3$.

Proof: Consider a chord $c \in (\mathcal{C} \backslash \mathcal{C}^*)$. Let s_c be the open line segment $\text{int}(e^*) \cap \widehat{\text{Vis}}(c)$ lying on e^*. We make the following observations.

(1) $s_i \subseteq s_c$ for every $u_i \in \mathcal{U}(c)$, since c lies in Γ.
(2) If $(\mathcal{U}_j \cap \mathcal{U}(c))$ is not empty, then $\gamma_j \notin s_c$. This can be proved by contradiction. If $\gamma_j \in s_c$, then c would be clearly visible to γ_j and would lie in the *interior* of Γ. But, c lies on the boundary of Γ.
(3) $\mathcal{U}_j \not\subseteq \mathcal{U}(c)$ for any j. To prove this, assume the contrary. Since γ_j is a midpoint of some $s_i \in (\mathcal{S}_j \cup \mathcal{S}'_j)$ and $s_i \subseteq s_c$, we have $\gamma_j \in s_c$ (and thus, contradicting (2) above).

From (3) above, the points in $\mathcal{U}(c)$ must be split between exactly two sets $\mathcal{U}_j$ and $\mathcal{U}_{j+1}$. Without loss of generality, assume that $\mathcal{U}_j$ contains at least half the points of $\mathcal{U}(c)$. If $|\mathcal{U}(c)| > 2m/3$, then $|\mathcal{U}_j \cap \mathcal{U}(c)| > m/3$ and $|\mathcal{U}_j - \mathcal{U}(c)| < m/3$. The line segments s_i for $u_i \in (\mathcal{U}_j \cap \mathcal{U}(c))$ lie on one side of γ_j, since $s_i \subseteq s_c$ and γ_j does not lie in s_c (from (1) and (2) above). Since $|\mathcal{U}_j - \mathcal{U}(c)| < m/3$, there are less than $m/3$ other segments of $\mathcal{S}_j \cup \mathcal{S}'_j$, so γ_j would not evenly split the midpoints of $\mathcal{S}_j \cup \mathcal{S}'_j$, contrary to the definition of γ_j. We conclude that $|\mathcal{U}(c)|$ is at most $2m/3$ for each $c \in (\mathcal{C} \setminus \mathcal{C}^*)$. ∎

4.3 Constructing Paths between Pairs of Vertices

In this section, we use PathsToEdge Algorithm to design an approximation algorithm that constructs pairwise disjoint paths connecting given pairs of vertices in a polygon P using $O(M)$ links where M is the number of links used by an optimal solution.

THEOREM 2. *Let* P *be a simple polygon on* n *vertices and* $\mathcal{U} = \{(u, u')\}$ *be an untangled set of* m *pairs of distinct vertices of* P. *A set of* m *pairwise disjoint interior paths connecting* u *to* u' *for each* (u, u') *in* $\mathcal{U}$ *can be constructed in* $O(n \log m + M \log m)$ *time using* $O(M)$ *line segments where* M *is the minimum number of line segments used by an optimal solution.*

Proof: We start with describing the approximation algorithm.

Algorithm. Construct a triangulation T_P of P, and arbitrarily choose an edge e^* of P. Consider the dual tree τ of the triangulation T_P with the triangle containing e^* as the root. For each $(u, u') \in \mathcal{U}$, let $t_{u,u'}$ be the least common ancestor in τ of the triangles[7] containing u and u'. See Figure 18. Now, for each point u, where (u, u') or (u', u) is in $\mathcal{U}$, we define e_u as follows. If $u \in t_{u,u'}$ then e_u is an edge of $t_{u,u'}$ containing u, else e_u is the edge of $t_{u,u'}$ that separates $t_{u,u'}$ from u. We observe the following properties of $t_{u,u'}$ and e_u.

. P1: If $u \notin e_u$, then e_u separates u from e^*. This is true because $t_{u,u'}$ separates u from e^* (as $t_{u,u'}$ is a proper ancestor of the triangles containing u in the dual tree τ), and e_u separates $t_{u,u'}$ from u.

[7]Note that more than one triangle may contain a point u.

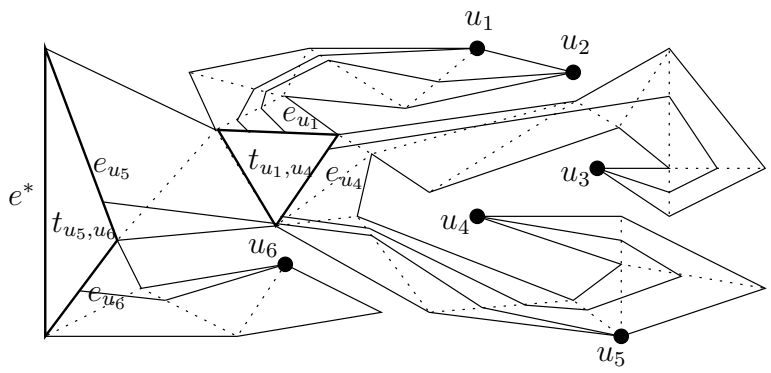

Fig. 18. The triangle $t_{u,u'}$ is the least common ancestor of triangles containing u and u', and the edge e_u is an edge of $t_{u,u'}$ that separates $t_{u,u'}$ from u. The figures corresponds to the following instance of vertex pairs: $\{(u_1, u_4), (u_2, u_3), (u_5, u_6)\}$. Here, $e_{u_2} = e_{u_1}$, $e_{u_3} = e_{u_4}$, and t_{u_2,u_3} is the same as the shown t_{u_1,u_4}.

. P2: A path from u to e_u, that intersects e_u only at the path's endpoint, does not intersect the interior of $t_{u,u'}$. This follows from the definition of e_u.

. P3: Any closed path from u to u' in P must intersect the closed triangle $t_{u,u'}$.

Let Π be the set of $m'(= 2m)$ pairs (u, e_u), where (u, u') or (u', u) is in $\mathcal{U}$. We construct m' pairwise disjoint interior paths $\{\zeta_u\}$ from u to e_u for each pair $(u, e_u) \in \Pi$ as follows. Let Π^* be the set of points for which e_u is the distinguished edge e^*. If Π^* is empty (which is actually the case initially), let Γ be the triangle containing e^*. Otherwise, construct the region Γ as in the PathsToEdge Algorithm using only the points in Π^*. Removing Γ subdivides P into subpolygons P_c. Recursively solve the problem of connecting vertex-edge pairs in each subpolygon P_c. Finally, as in PathsToEdge Algorithm, connect each path with endpoint on the boundary of Γ to the appropriate edge in Γ. The only difference with PathsToEdge Algorithm is that if $u \in e_u$ then the path ζ_u is just the point u itself. Otherwise, due to property P1, the edge e_u separates u from e^*, and hence, the connection is not to e^* but to e_u on the way to e^*.

Now, for each $(u, u') \in \mathcal{U}$, connect the paths ζ_u and $\zeta_{u'}$ by a line segment from e_u to $e_{u'}$ in $t_{u,u'}$ (this is possible due to property P2). See Figure 19. This completes the construction of pairwise disjoint interior paths connecting each pair of vertices (u, u') in $\mathcal{U}$.

Output Size Analysis. Let M' be the minimum total number of line segments necessary to connect the vertex-edge pairs in Π by pairwise disjoint paths. As argued in Lemma 5, the constructed paths connecting e^* to Π^* use $3|\Pi^*|$ line segments in Γ while the optimal solution requires at least $|\Pi^*|/40$ line segments before leaving Γ. Each path whose endpoint is not on e^* but on some other edge lying in Γ may also require three line segments. Since this can happen only once per path, the ζ_u paths use a total of at most $120M' + 3m'$ line segments.

Let M be the minimum total number of line segments necessary to connect the (u, u') pairs in $\mathcal{U}$. Because of property P3, the m paths connecting pairs of vertices

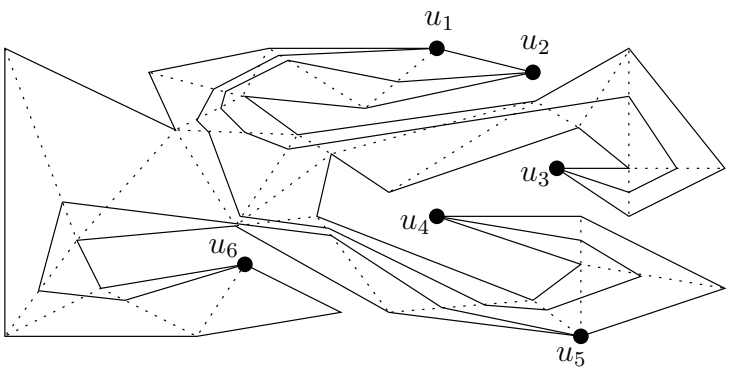

Fig. 19. Pairwise disjoint interior paths connecting the following pairs of vertices: $\{(u_1, u_4), (u_2, u_3), (u_5, u_6)\}$.

in $\mathcal{U}$ can be cut to form $m' = 2m$ pairwise disjoint paths connecting vertex-edge pairs in Π using at most $M + m$ line segments. Thus, $M' \leq M + m$. Our algorithm uses at most $120M' + 3m'$ line segments to connect the vertex-edge pairs in Π and at most $120M' + 3m' + m$ line segments to connect the vertex-vertex pairs in $\mathcal{U}$. Thus, our solution uses at most $120M + 127m$ line segments.

Time Analysis. Since least common ancestor queries can be answered in constant time after linear processing of the tree [Harel and Tarjan 1984; Bender and Farach-Colton 2000], we can determine $t_{u,u'}$ for all pairs in $\mathcal{U}$ in $O(n + m)$ time. Now, it follows from Lemma 6 that the above described algorithm runs in $O(n \log m + M \log m)$ time. ∎

5. PARTITIONING A SEQUENCE OF LINE SEGMENTS

In this section, we describe and analyze the Partition-Segments Algorithm that partitions a sequence of line segments as suggested in Lemma 2. We recall the relevant definitions from before. Given a sequence of line segments $\mathcal{S}$ and a point γ in $\mathbb{R}^1$, $f(\gamma, \mathcal{S})$ is the number of line segments of $\mathcal{S}$ that contain the point γ_j, $f^-(\gamma, \mathcal{S})$ and $f^+(\gamma, \mathcal{S})$ are the number of line segments of $\mathcal{S}$ that are contained in the open intervals $(-\infty, \gamma)$ and $(\gamma, +\infty)$ respectively. Given two sets of line segments $\mathcal{S}_j$ and $\mathcal{S}'_j$, $\mathcal{F}(\gamma, \mathcal{S}_j, \mathcal{S}'_j)$ is defined as

$$\mathcal{F}(\gamma, \mathcal{S}_j, \mathcal{S}'_j) = f(\gamma, (\mathcal{S}_j \cup \mathcal{S}'_j)) + \min(f^+(\gamma, \mathcal{S}_j), f^-(\gamma, \mathcal{S}'_j)).$$

Also, $g(\mathcal{S})$ is the median of the midpoints of the segments in $\mathcal{S}$. We use γ_j to denote $g(\mathcal{S}_j \cup \mathcal{S}'_j)$.

Partition-Segments Algorithm

Given: A sequence of open line segments $\mathcal{S} = (s_1, s_2, \ldots, s_m)$ in $\mathbb{R}^1$.

Output: A partitioning of $\mathcal{S}$ into contiguous subsequences $\mathcal{S}_1 = (s_1, s_2, \ldots, s_{i_1})$, $\mathcal{S}'_1 = (s_{i_1+1}, s_{i_1+2}, \ldots, s_{i_2})$,, $\mathcal{S}_2 = (s_{i_2+1}, s_{i_2+2}, \ldots, s_{i_3}), \ldots, \mathcal{S}'_h = (s_{i_{2h-1}+1}, s_{i_{2h-1}+2}, \ldots, s_m)$, such that

Partition-Segments($\mathcal{S}$)
/* $\mathcal{S}$ = a sequence of line segments $(s_1, s_2, \ldots, s_m)$ */
/* Returns a linked list of contiguous subsequences of $\mathcal{S}$ */
1. Initialize linked list $\mathcal{A}$ to $\emptyset$;
2. FOR $i = 1$ TO m DO
3. Create new node a where $a.seq = (s_i)$ and $a.size = 1$;
4. Add a to the end of linked list $\mathcal{A}$;
5. WHILE $\exists a \in \mathcal{A}$ such that $g(a.seq) > g(a.next.seq)$ DO
6. Merge a and $a.next$ to form a new node a' in $\mathcal{A}$;
7. BALANCE-NEXT(a');
8. BALANCE-PREV(a');
/* Now, evenly split the j^{th} node to create $\mathcal{S}_j$ and $\mathcal{S}'_j$ (without changing $\mathcal{A}$). */
9. $a = \mathcal{A}.first$;
10. $j = 0$;
11. REPEAT
12. IF $(j \neq 0)$ $a = a.next$;
13. $j = j + 1$;
14. Partition $a.seq = (s_i, \ldots, s_{i'})$ to create two new approximately equal sized sequences $\mathcal{S}_j = (s_i, \ldots, s_{\lceil (i+i')/2 \rceil})$ and $\mathcal{S}'_j = (s_{\lceil (i+i')/2 \rceil + 1}, \ldots, s_{i'})$.
15. UNTIL $a = \mathcal{A}.last$;
16. $h = j$;
17. Return$((\mathcal{S}_1, \mathcal{S}'_1, \mathcal{S}_2, \ldots, \mathcal{S}_h, \mathcal{S}'_h))$.

Fig. 20. Partition-Segments Algorithm.

(1) $|\mathcal{S}_j| = |\mathcal{S}'_j|$ or $|\mathcal{S}_j| = |\mathcal{S}'_j| + 1$ for $1 \leq j \leq h$;
(2) $\gamma_1 \leq \gamma_2 \leq \cdots \ldots \gamma_h$, where $\gamma_j = g(\mathcal{S}_1 \cup \mathcal{S}'_1)$;
(3) $\sum_{j=1..h} \mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq m/40$.

Basic Idea. The general idea of our algorithm is to start with a total partition of one element per set, and merge adjacent sets S_i and S_{i+1} whenever $g(S_i)$ and $g(S_{i+1})$ are in the wrong order. If S_i and S_{i+1} are approximately the same size, then $\mathcal{F}(g(S_i \cup S_{i+1}), S_i, S_{i+1})$ can be shown to be large (see Lemma 10). Thus, we ensure that any two adjacent sets are of approximately the same size, by appropriately rebalancing neighboring sets after each merge. Sets created in the rebalancing process (after the merge of S_i and S_{i+1}) may not generate large values of $\mathcal{F}$, but we can bound their total size by the size of S_i. Once all the $g(S_i)$ are in the proper order, each set S_i is evenly split into two, $\mathcal{S}_i$ and $\mathcal{S}'_i$, for the desired partition. A detailed description of the algorithm follows.

Algorithm Description. We start with splitting $\mathcal{S}$ into m distinct subsequences, (s_i), consisting of one element each. Store the m sets in a linked list $\mathcal{A}$ in the order they appear in $\mathcal{S}$. Each set is stored in a separate node a in the linked list where $a.seq$ contains the subsequence stored at a, $a.size = |a|$ is the number of elements

```
BALANCE-NEXT(a)
/* a = node in linked list. Ensure all nodes a' after a are balanced. */
 1. If a = A.last THEN RETURN;
 2. a' ← a;
 3. WHILE a'.next ≠ A.last AND |a'| > 3|a'.next| DO
 4.         REPEAT
 5.                 Merge a'.next and a'.next.next;
 6.         UNTIL |a'| ≤ 3|a'.next| OR a'.next = A.last;
 7.         IF a'.next = A.last THEN RETURN;
 8.         IF |a'.next| ≤ 3|a'.next.next| THEN RETURN;
 9.         IF |a'.next| > (2/3)|a'| THEN
10.                 Split a'.next into two equal sized sets;
11.                 RETURN;
12.         a' ← a'.next;

BALANCE-PREV(a)
/* a = node in linked list. Ensure all nodes a' before a are balanced. */
 1. If a = A.first THEN RETURN;
 2. a' ← a;
 3. WHILE a'.prev ≠ A.first AND |a'| > 3|a'.prev| DO
 4.         REPEAT
 5.                 Merge a'.prev and a'.prev.prev;
 6.         UNTIL |a'| ≤ 3|a'.prev| OR a'.prev = A.first;
 7.         IF a'.prev = A.first THEN RETURN;
 8.         IF |a'.prev| ≤ 3|a'.prev.prev| THEN RETURN;
 9.         IF |a'.prev| > (2/3)|a'| THEN
10.                 Split a'.prev into two equal sized sets;
11.                 RETURN;
12.         a' ← a'.prev;
```

Fig. 21. BALANCE-NEXT and BALANCE-PREV Procedures.

in the $a.seq$, $a.next$ is the next node in the linked list, and $a.prev$ is the previous node in the linked list. Let $\mathcal{A}.first$ and $\mathcal{A}.last$ be the first and last nodes in $\mathcal{A}$.

DEFINITION 11. (Balanced node/list) We call a node a in $\mathcal{A}$ *balanced* if $3|a| \geq |a.next|$ and $3|a| \geq |a.prev|$. The linked list $\mathcal{A}$ is *balanced* if all nodes a in $\mathcal{A} - \{\mathcal{A}.first, \mathcal{A}.last\}$ are balanced. □

Clearly, $\mathcal{A}$ is initially balanced. While $\mathcal{A}$ contains some node a such that $g(a.seq) > g(a.next.seq)$, repeat the following. First, merge node a with $a.next$, storing the result in a new node a'. Next, call procedures BALANCE-NEXT and BALANCE-PREV to rebalance the linked list by merging and splitting nodes near a'. Repeat this while loop until $g(a.seq) \leq g(a.next.seq)$ for each a in $\mathcal{A}$. See Figure 20.

Procedure BALANCE-NEXT rebalances nodes after a' as follows. Note that a' is balanced, since it was formed by merging balanced nodes. However, $a'.next$ may have become unbalanced due to $3|a'.next|$ becoming less than $|a'|$. If $3|a'.next|$

is less than $|a'|$, then merge nodes $a'.next$ and $a'.next.next$ storing the result in $a'.next$. Continue merging $a'.next$ and $a'.next.next$ until $3|a'.next|$ is greater than or equal to $|a'|$. Now, $a'.next$ is balanced, but $a'.next.next$ may have become unbalanced. If $a'.next.next$ is balanced, i.e., $3|a'.next.next| \geq |a'next|$, then stop. If $|a'.next|$ is greater than $(2/3)|a'|$, then split $a'.next$ into two nodes of approximately equal size and stop (Lemma 8 shows that this splitting balances the nodes). Otherwise, let a' equal $a'.next$ and start again. In a similar manner, BALANCE-PREV rebalances nodes before a. See Figure 21. This completes the description of our Partition-Segments Algorithm.

Proof Plan. In the following lemmas, we prove the correctness and time complexity of the Partition-Segments algorithm. As a general outline, we follow the following plan.

(1) Lemma 8 shows that the Partition-Segments Algorithms keeps the list balanced.
(2) For any two sets of segments $\mathcal{S}_j$ and $\mathcal{S}'_j$ such that $|\mathcal{S}_j|$ is $|\mathcal{S}'_j|$ or $|\mathcal{S}'_j| + 1$, we show that

$$\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq \min(f(\gamma_j, \mathcal{S}_j) + f^+(\gamma_j, \mathcal{S}_j), f(\gamma_j, \mathcal{S}'_j) + f^-(\gamma_j, cS'_j)).$$

The above follows from the definition of $\mathcal{F}$, and is shown in Lemma 9.
(3) Let a_j be the j^{th} node in the final $\mathcal{A}$ used to create the subsequences $\mathcal{S}_j$ and $\mathcal{S}'_j$. If a_j was created by merging two nodes a and $a.next$ such that $g(a.seq) > g(a.next.seq)$, then $\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq (1/8)|a_j|$. Intuitively, the above is true due to Lemma 9 and the approximately equal sizes of a and $a.next$. See Lemma 10.
(4) Lemma 11 shows that the total size of all the nodes a_j that are created in the BALANCE-NEXT or BALANCE-PREV procedures is at most $4/5m$.
(5) Now, a node a_j in the final list $\mathcal{A}$ is created either by merging two nodes (as considered in (3) above), or in the BALANCE-NEXT or BALANCE-PREV procedures. The rest of the nodes a_j consist of a single element (s_i), and satisfy $\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq (1/8)|a_j|$. Thus, using (3) and (4) above, there are at least $m/5$ line segments in nodes a_j with the property that $\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq (1/8)$, and hence, the final subsequences satisfy the property $\sum_{j=1..h} \mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq m/40$. It is easy to see that the final subsequences also satisfy the other desired properties. The above is formally proven in Lemma 12.

LEMMA 8. *The Partition-Segments Algorithm maintains the balance of the linked list $\mathcal{A}$, except in steps 6-8 (step 6 creates a new node, and step 7-8 are calls to the BALANCE-PREV/NEXT procedures).*

Proof: We show that completion of Step 10 in BALANCE-NEXT ensures that all nodes after a' are balanced.

Let b_1, b_2, b_3 be the three nodes after a' at the completion of Step 10 of BALANCE-NEXT, where $b_1 = a'.next$, $b_2 = b_1.next$ and $b_3 = b_2.next$. Nodes b_1 and b_2 were created by splitting a node whose size is greater than $(2/3)|a'|$ and $3|b_3|$. It is easy to see that nodes a', b_1, and b_2 are balanced after the split.[8] Since

[8]Since, $3|b_1| \geq |a'|$ and $3|b_2| \geq |b_3|$. Also, nodes b_1 and b_2 have approximately the same size, so $3|b_2| \geq |b_1|$ and $3|b_1| \geq |b_2|$.

nodes b_3 and beyond have not been touched, we need to only ensure that b_3 is balanced due to $|b_2|$. In particular, we need to show that $3|b_3| \geq |b_2|$,

Proving $|b_3| \geq |b_2|/3$. Let b' be the node split to create b_1 and b_2. Node b' was created by merging a sequence of nodes $b'_1, b'_2, \ldots, b'_k$, where $a'.next = b'_1$ and $b'_i.next = b'_{i+1}$. The total number of elements in $b'_1, b'_2, \ldots, b'_{k-1}$ is less than $|a'|/3$ or b'_k would not have been added to b'. The value of $|b'_k|$ must be greater than $|a'|/3$ or else $|b'|$ would not be greater than $(2/3)|a'|$. Thus, we have

$$|b'_k| > |a'|/3, \tag{1}$$

$$|a'|/3 > \sum_{i=1..k-1} |b'_i|. \tag{2}$$

Since the list was initially balanced, we have

$$\begin{aligned} |b_3| &\geq |b'_k|/3 = |b'_k|/6 + |b'_k|/6 \\ &\geq (|b'_k| + \sum_{i=1..k-1} |b'_i|)/6 \qquad \text{From equations (1), (2)} \\ &\geq (2|b_2|)/6 = |b_2|/3 \end{aligned}$$

The last inequality follows because the node b_2 is formed by splitting the node b', which was creating by merging the sequence of nodes $b'_1, b'_2, \ldots, b'_k$.

The above shows that new list is balanced after the completion of of Step 10 in BALANCE-NEXT. Similarly, it can be shown that after completion of Step 10 in BALANCE-PREV, all nodes before a' are balanced. Following the description of the balance procedures, it is now easy to see that the BALANCE-NEXT and BALANCE-PREV procedures are successful in balancing the linked list $\mathcal{A}$, that might have become unbalanced due to step 6 of Partition-Segments Algorithm. Thus, the linked list $\mathcal{A}$ is balanced at each step (except steps 6 to 8) of the Partition-Segments Algorithm. ∎

LEMMA 9. *For any two sets of segments $\mathcal{S}_j$ and $\mathcal{S}'_j$ such that $|\mathcal{S}_j|$ is $|\mathcal{S}'_j|$ or $|\mathcal{S}'_j|+1$, we have $\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq \min(f(\gamma_j, \mathcal{S}_j)+f^+(\gamma_j, \mathcal{S}_j), f(\gamma_j, \mathcal{S}'_j)+f^-(\gamma_j, c\mathcal{S}'_j))$.*

Proof: For simplicity, let m_0, m_-, m_+ denote $f(\gamma_j, \mathcal{S}_j)$, $f^-(\gamma_j, \mathcal{S}_j)$, and $f^+(\gamma_j, \mathcal{S}_j)$ respectively, while m'_0, m'_-, m'_+ denote $f(\gamma_j, \mathcal{S}'_j)$, $f^-(\gamma_j, \mathcal{S}'_j)$, and $f^+(\gamma_j, \mathcal{S}'_j)$ respectively. By the choice of point γ_j,

$$m_0 + m_- + m'_0 + m'_- \geq |\mathcal{S}_j \cup \mathcal{S}'_j|/2.$$

Since $|\mathcal{S}_j|$ equals $|\mathcal{S}'_j|$ or $|\mathcal{S}'_j| + 1$,

$$m_0 + m_- + m'_0 + m'_- \geq |\mathcal{S}_j|. \tag{3}$$

On the other hand,

$$m_0 + m_- + m_+ = |\mathcal{S}_j|. \tag{4}$$

Subtracting (4) from (3), we get

$$m'_0 + m'_- \geq m_+. \tag{5}$$

Thus,

$$\begin{aligned} m_0 + m'_0 + \min(m_+, m'_-) &= \min(m_0 + m'_0 + m_+, m_0 + m'_0 + m'_-) \\ &\geq \min(m_0 + m'_0 + m_+, m_0 + m_+) \quad \text{From 5} \\ &= m_0 + m_+ \end{aligned} \tag{6}$$

Similarly, we can show that $m_0 + m_+ \geq m'_-$, and

$$m_0 + m'_0 + \min(m_+, m'_-) \geq m'_0 + m'_-. \tag{7}$$

Recalling the definition of $\mathcal{F}$, we have

$$\begin{aligned} \mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) &= m_0 + m'_0 + \min(m_+, m'_-) \\ &\geq \min(m_0 + m_+, m'_0 + m'_-) \qquad \text{From (6), (7)} \end{aligned}$$

■

We now present two lemmas that are used in Lemma 12 to prove the correctness of the Partition-Segments Algorithm. Let a_j be the j^{th} node in $\mathcal{A}$ at the completion of Partition-Segments Algorithm used to create the subsequences $\mathcal{S}_j$ and $\mathcal{S}'_j$.

LEMMA 10. *If node a_j was created in step 6 of Partition-Segments Algorithm (by merging two nodes a and $a.next$ such that $g(a.seq) > g(a.next.seq)$), then $\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq (1/8)|a_j|$.*

Proof: Let us assume that $|a.next| \geq |a|$. Since, the sequence $a.seq$ is a subsequence of $\mathcal{S}_j$, we get

$$f(\gamma_j, \mathcal{S}_j) \geq f(\gamma_j, a.seq), \tag{8}$$

$$f^+(\gamma_j, \mathcal{S}_j) \geq f^+(\gamma_j, a.seq). \tag{9}$$

Since, $a_j.seq = (a.seq \cup a.next.seq)$, the point $\gamma_j = g(a_j.seq)$ must lie between $g(a.seq)$ and $g(a.next.seq)$. So, we get

$$\begin{aligned} f(\gamma_j, a.seq) + f^+(\gamma_j, a.seq) &\geq f(g(a.seq), a.seq) + f^+(g(a.seq), a.seq) \\ &\geq |a|/2. \end{aligned} \tag{10}$$

Since $|a.next| \leq 3|a|$, we have $|a_j| \leq 4|a|$. Thus, from above equations (8, 9, 10), we get

$$f(\gamma_j, \mathcal{S}_j) + f^+(\gamma_j, \mathcal{S}_j) \geq (1/8)|a_j|. \tag{11}$$

In the case that $|a.next| < |a|$, similar reasoning gives

$$f(\gamma_j, \mathcal{S}'_j) + f^-(\gamma_j, \mathcal{S}'_j) \geq (1/8)|a_j|. \tag{12}$$

From Lemma 9 and equations (11), (12), we get

$$\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq (1/8)|a_j|.$$

■

LEMMA 11. *At most $(4/5)m$ line segments lie in the nodes a_j that are created in the BALANCE-NEXT or BALANCE-PREV procedures.*

Proof: Recall that a_j is the j^{th} node in $\mathcal{A}$ and the size of $\mathcal{A}$ is equal to h at the completion of Partition-Segments Algorithm.

Let $b_1, b_2, \ldots, b_k$ be nodes created in a call BALANCE-NEXT(a) where $a.next = b_1$ and $b_j.next = b_{j+1}$ for $j \leq k$. Since BALANCE-NEXT stops whenever $|a'.next| > (2/3)|a'|$, we have the following:

— $|b_1| \leq (2/3)|a|$,
— $|b_j| \leq (2/3)|b_{j-1}|$ for all $2 \leq j \leq k-1$, and
— $|b_k| \leq (4/3)|b_{k-1}|$. We prove this in the next paragraph.

Thus, we get

$$\sum_{i=1..k} |b_j| \leq \left(\sum_{i=1..k-1} (2/3)^i |a| \right) + (4/3)(2/3)^{k-1}|a| \leq 2|a|.$$

Similarly, the total size of the nodes created in a call BALANCE-PREV(a) is at most $2|a|$. Since each line segment in a corresponds to at most 4 line segments in nodes created in BALANCE-NEXT(a) and BALANCE-PREV(a), there are at most $(4/5)m$ line segments in nodes created in the BALANCE-NEXT and BALANCE-PREV procedures.

Proof of $|b_k| \leq (4/3)|b_{k-1}|$. Let $b'_1, b'_2, \ldots, b'_h$ be the nodes merged to form b_k, where $b'_j.next = b'_{j+1}$, $j \leq h$. The total number of elements in $b'_1, b'_2, \ldots, b'_{h-1}$ is less than $|b_{k-1}|/3$ or b'_h would not have been added to b_k. In particular, $|b'_{h-1}|$ must be less than $|b_{k-1}|/3$. Since the list was initially balanced, we have

$$|b'_h| \leq 3|b'_{h-1}| \leq 3(|b_{k-1}|/3) = |b_{k-1}|.$$

Thus,

$$|b_k| = |b'_h| + \sum_{j=1..h-1} |b'_j| \leq (4/3)|b_{k-1}|$$

. ∎

LEMMA 12. *Partition-Segments Algorithm is correct, i.e., the set of subsequences* $\{\mathcal{S}_1, \mathcal{S}'_1, \mathcal{S}_2, \ldots, \mathcal{S}'_h\}$ *returned by Partition-Segments Algorithm have the desired properties.*

Proof: Initially, the line segments s_j are stored in $\mathcal{A}$ in sorted order of their subscripts. The merging and splitting steps in the main algorithm and in the subroutines BALANCE-NEXT and BALANCE-PREV preserve the order of the s_i, so the subsequences $\mathcal{S}_j, \mathcal{S}'_j$ returned properly partition $\mathcal{S}$ into contiguous subsequences. Sets $\mathcal{S}_j$ and $\mathcal{S}'_j$ are created by partitioning $a_j.seq$ into two equal sized sequences, so property 1 is satisfied. Recall that γ_j is $g(\mathcal{S}_j \cup \mathcal{S}'_j) = g(a_j.seq)$. The while loop of Partition-Segments only terminates when $\gamma_1 \leq \gamma_2 \leq \cdots \leq \gamma_h$, so property 2 is clearly satisfied.

To show that property 3 holds, note that a node a_j is either an initial node or it is created in step 6 of Partition-Segments or it is created in the BALANCE-NEXT or BALANCE-PREV procedures. If a_j is an initial node, then $\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq 1 \geq (1/8)|a_j|$ (as $a_j.seq = \{s\}$ for some $s \in \mathcal{S}$, $\mathcal{S}_j = \{s\}, \mathcal{S}'_j = \emptyset$, and point γ_j is the midpoint of s). Now, using Lemma 10 and Lemma 11, there are at least $m/5$ line

segments in nodes a_j with the property that $\mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq (1/8)|a_j|$. Thus, we get $\sum_{j=1..h} \mathcal{F}(\gamma_j, \mathcal{S}_j, \mathcal{S}'_j) \geq m/40$. ∎

LEMMA 13. *The Partition-Segments Algorithm runs in $O(m \log m)$ time.*

Proof: Steps 1-4 of Partition-Segments Algorithm take $O(m)$ time. Step 6 can be implemented in constant time.[9] Steps 9-17 take $O(m)$ time. Below, we show that the total running time of step 5 and the balance procedures (steps 7 and 8) is $O(m \log m)$. First, we observe that the number of iterations of the WHILE loop of Partition-Segments Algorithm is at most $(m-1)$.

Merging two nodes decreases the number of nodes in $\mathcal{A}$ by one. Procedures BALANCE-NEXT and BALANCE-PREV never increase the number of such nodes.[10] Since list $\mathcal{A}$ starts with m nodes, step 6 is executed at most $m-1$ times. Similarly, procedures BALANCE-NEXT and BALANCE-PREV are called at most $m-1$ times.

Running Time of Balance Procedures. The running time of BALANCE-NEXT is dominated by the inner loop in step 5 and the time to execute step 10. Each execution of step 5 reduces the number of nodes in $\mathcal{A}$ by one. Step 10 increases the number of nodes in $\mathcal{A}$ by one, but it is only executed once per call to BALANCE-NEXT. Since BALANCE-NEXT is only called $m-1$ times and there are at most m nodes in $\mathcal{A}$, step 5 is executed at most $2m-2$ times.

The time to split node $a'.next$ in step 10 of BALANCE-NEXT is proportional to $|a'.next|$. As argued in Lemma 11, the total size of the nodes created in the call BALANCE-NEXT(a') is at most $2|a'|$. Thus, the total running time of step 10 is proportional to the sum of the sizes of nodes created in step 6 of Partition-Segments. Step 6 merges nodes a and $a.next$ to create a new node a'. The size of a' is at least $4/3|a|$ and $4/3|a.next|$. Since each time a node is merged in step 6, its size increases by $4/3$, any given line segment s is in $O(\log m)$ merge steps. Thus, the sum of the sizes of nodes created in step 6 of Partition-Segments and the total running time of step 10 of BALANCE-NEXT is $O(m \log m)$. A similar analysis applies to BALANCE-PREV.

Running Time of Step 5 of Partition-Segments. We are left with analyzing the running time of step 5 of Partition-Segments. Step 5 checks if $a.next.g < a.g$ for some $a \in \mathcal{A}$. We can accomplish this by keeping a set $\mathcal{A}^*$ of nodes in $\mathcal{A}$ to be checked. For each node $a \in \mathcal{A}^*$, we check that $g(a.seq) \leq g(a.next.seq)$ and that $g(a.seq) \geq g(a.prev.seq)$. The points $g(a.seq)$, $g(a.next.seq)$ and $g(a.prev.seq)$ can be found in time proportional to $|a|$, $|a.next|$ and $|a.prev|$, respectively, using a linear median find algorithm. Since $|a.next| \leq 3|a|$ and $|a.prev| \leq 3|a|$, computing $g(a.seq)$, $g(a.next.seq)$ and $g(a.prev.seq)$ takes $O(|a|)$ time. Node a can be created as a merge of two nodes in step 6 of Partition-Segments or in BALANCE-NEXT or BALANCE-PREV. As previously argued, the sum of the sizes of the nodes created in step 6 of Partition-Segments is $O(m \log m)$. Each node a' created in step 6 can cause the creation of nodes in BALANCE-NEXT(a') and BALANCE-PREV(a') with a total of $4|a'|$ elements. Thus, the sum of the sizes of created nodes is $O(m \log m)$ and the running time of step 5 is $O(m \log m)$. ∎

[9] The sequences $a.seq$ can be stored as linked lists with first and last pointers.

[10] Node $a'.next$ is split only in step 10 when it is the result of multiple merges.

6. CONSTRUCTING ISOMORPHIC TRIANGULATIONS

In this section, we use our approximation algorithm for the PDLP problem to improve our earlier result [Gupta and Wenger 1997] on construction of isomorphic triangulations between simple polygons.

A triangulation T_P of P (possibly with interior vertices) is *isomorphic* to a triangulation T_Q of Q if there is a one-to-one, onto mapping f between the vertices of T_P and the vertices of T_Q such that p, p', p'' are vertices of a triangle in T_P if and only if $f(p), f(p'), f(p'')$ are vertices of a triangle in T_Q. An isomorphic triangulation of P and Q also defines a piecewise linear homeomorphism between P and Q. The size of a triangulation is the total number of vertices, edges and triangles in the triangulation.

Algorithms for constructing isomorphic triangulations between labeled point sets are described in [Saalfeld 1987] and [Souvaine and Wenger 1994]. Algorithms for isomorphic triangulations between simple polygons are given in [Aronov et al. 1993], [Kranakis and Urrutia 1995], and [Gupta and Wenger 1997]. In [Gupta and Wenger 1997], we designed an algorithm to construct isomorphic triangulations of simple polygon P and Q consisting of $O(M_1 \log n + n \log^2 n)$ triangles in $O(M_1 \log n + n \log^2 n)$ time, where M_1 is the size of an optimal (minimum size) solution and n is the number of vertices of the polygons. The result in Theorem 2 leads directly to an improvement in the above result of [Gupta and Wenger 1997] when $M_1 = o(n \log n)$. We outline the improved algorithm here. Readers are referred to [Gupta and Wenger 1997] for more complete details and figures of the original algorithm.

THEOREM 3. *Let P be a simple polygon with vertices $\{p_1, p_2, \ldots, p_n\}$ and Q be a simple polygon with vertices $\{q_1, q_2, \ldots, q_n\}$. Let M_1 be the minimum number of triangles in any isomorphic triangulation of P and Q mapping p_i to q_i. Isomorphic triangulations mapping p_i to q_i using $O(M_1 \log n)$ edges can be constructed in $O(M_1 \log n)$ time. Moreover, no new vertices are added to the boundary of P and Q in these triangulations.*

Proof: We start with a description of the algorithm.

Algorithm Description. As in [Gupta and Wenger 1997], construct a triangulation T_P of P using no vertices other than those of P and choose a set of edges $\mathcal{C}_P$ from T_P as follows. Arbitrarily, pick an edge e^* of P. Let Ψ_P be the union of the set of triangles of T_P that intersect $\widehat{\text{Vis}}(e^*)$. The boundary of Ψ_P is composed of edges and chords of the original polygon P. Add all the chords bounding Ψ_P to $\mathcal{C}_P$. Each such chord c divides P into two polygons. Let $P(c)$ be one not containing e^*. Recursively apply this procedure to each such polygon $P(c)$ starting at its edge c, to choose a set of edges from $P(c)$ and add them to $\mathcal{C}_P$. Let k be the size of $\mathcal{C}_P$.

For each diagonal (p_i, p_j) in P, there is a corresponding pair of vertices (q_i, q_j) in Q. Let Π_Q be the set of k pairs of distinct vertices of Q corresponding to the diagonals in $\mathcal{C}_P$. Note that the pairs of vertices in Π_Q are untangled. By Theorem 2, a set of k pairwise disjoint interior paths connecting each pair in $(q_i, q_j) \in \Pi_Q$ can be constructed in $O(n \log k + M_2 \log k)$ time using at most $120M_2 + 127k$ total links, where M_2 is the minimum total number of line segments necessary to connect all pairs $(q_i, q_j) \in \Pi_Q$ by pairwise disjoint paths. For each new vertex q' on the path

from q_i to q_j, create a corresponding new vertex p' on the diagonal (p_i, p_j).

The k diagonals in $\mathcal{C}_P$ split polygon P into $k+1$ subpolygons $P_1, P_2, \ldots, P_{k+1}$. The interior paths split polygon Q into $k+1$ corresponding subpolygons $Q_1, Q_2, \ldots, Q_{k+1}$. Each subpolygon P_i has an edge e that lies within link distance two of every point in P_i. Thus, each P_i has link diameter at most five. By Lemma 3.4 in [Gupta and Wenger 1997], isomorphic triangulations of P_i and Q_i can be constructed in $O(n_i \log n_i)$ time using $40n_i\lfloor \log_2 n_i \rfloor + O(n_i)$ edges, where n_i is the number of vertices of P_i and Q_i. Since these isomorphic triangulations do not contain any new boundary vertices, their union is an isomorphic triangulation of P and Q.

Output Size Analysis. The total number of edges in the isomorphic triangulations of P and Q is at most $\sum_{i=1}^{k+1}(40n_i\lfloor \log_2 n_i \rfloor + O(n_i))$. Since each edge of Q_i is either an edge of Q or an edge on one of the k disjoint interior paths, $\sum_{i=1}^{k+1} n_i < n + 2(120M_2 + 127k)$. Bounding $\log_2 n_i$ by $\log_2 n$ and k by n gives a bound of

$$40(240M_2 + 254n)\lfloor \log_2 n \rfloor + O(M_2 + n) = O(M_2 \log n + n \log n).$$

Let T_P^* and T_Q^* be an optimal pair of isomorphic triangulations of P and Q mapping p_i to q_i and let M_1 be the number of triangles in T_P^* (which equals the number of triangles in T_Q^*). In the next paragraph, we show that $M_2 \leq 3M_1$. Now since $M_1 \geq n-2$, we conclude that the total number of edges in the constructed isomorphic triangulations of P and Q is $M_2 \log n + n \log n = O(M_1 \log n)$.

Proving $M_2 \leq 3M_1$. As isomorphic triangulations define a piecewise linear mapping between P and Q [Gupta and Wenger 1997], each diagonal $(p_i, p_j) \in \mathcal{C}_P$ has a mapping, induced by the isomorphic triangulations T_P^* and T_Q^*, in Q. In particular, the set of diagonals $\mathcal{C}_P$ get mapped to a set of disjoint link paths ζ in Q. By the choice of $\mathcal{C}_P$, no triangle of T_P^* intersects more than three diagonals of $\mathcal{C}_P$. Hence, no triangle of T_Q^* intersects more than three link paths from ζ. Since, each path in ζ uses at most one line segment within a triangle of T_Q^*, the set of paths ζ use at most $3M_1$ total line segments. As M_2 is the number of line segments in an optimal construction of link paths, we get $M_2 \leq 3M_1$.

Time Complexity. As in [Gupta and Wenger 1997], it can be shown that the running time of the above algorithm is $O(M_1 \log n)$, since the additional time is only $O(n \log k + M_2 \log k) = O(M_1 \log n)$ for constructing disjoint paths for Π_Q. ∎

7. CONCLUSIONS

Let P be a simple polygon on n vertices, let Π be an untangled set of m pairs of distinct vertices of P, and let L be the sum of the interior link distances between pairs in Π. In [Gupta and Wenger 1997], we proved that pairwise disjoint interior paths connecting the pairs can be constructed using $L + O(m \log m)$ line segments. In this paper we showed that $L + \Omega(m \log m)$ line segments are sometimes required. We gave an approximation algorithm for constructing pairwise disjoint paths in $O(n + M + m \log m)$ time using $O(M)$ line segments where M is the minimum number required. We applied this approximation algorithm to construct isomorphic triangulations of two polygons in $O(M_1 \log n)$ time using $O(M_1 \log n)$ edges where M_1 is the size of the optimal solution. This improved our result in [Gupta and Wenger 1997].

We still do not know if constructing pairwise disjoint paths using the fewest line segments is an NP-complete problem or whether it can be solved in polynomial time. We also do not know of an ϵ-approximation scheme for constructing such paths. Possibly the techniques of Arora [1996] for the Euclidean Traveling Salesman Problem could apply.

Kahan and Snoeyink [1996] show that a polygon on n vertices with bit complexity $n \log n$ may have a minimal link path with $\Theta(n^2 \log n)$ bit complexity. Similar problems plague pairwise disjoint paths with the minimum number of line segments. Because the algorithms in this paper are approximation algorithms, the bit complexity of coordinates in the output is a constant factor times the bit complexity of input coordinates.

REFERENCES

Aggarwal, A., Schieber, B., and Tokuyama, T. 1993. Finding a minimum weight k-link path in graphs with Monge property and applications. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.* 189–197.

Aggarwal, A., Schieber, B., and Tokuyama, T. 1994. Finding a minimum-weight k-link path in graphs with the concave Monge property and applications. *Discrete Comput. Geom. 12*, 263–280.

Alsuwaiyel, M. H. and Lee, D. T. 1993. Minimal link visibility paths inside a simple polygon. *Comput. Geom. Theory Appl. 3,* 1, 1–25.

Arkin, E. M., Mitchell, J. S. B., and Suri, S. 1992. Optimal link path queries in a simple polygon. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms.* 269–279.

Arkin, E. M., Mitchell, J. S. B., and Suri, S. 1995. Logarithmic-time link path queries in a simple polygon. *Internat. J. Comput. Geom. Appl. 5,* 4, 369–395.

Aronov, B., Seidel, R., and Souvaine, D. 1993. On compatible triangulations of simple polygons. *Comput. Geom. Theory Appl. 3,* 1, 27–35.

Arora, S. 1996. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.* 2–11.

Bender, M. A. and Farach-Colton, M. 2000. The lca problem revisited. In *LATIN '00: Proceedings of the 4th Latin American Symposium on Theoretical Informatics.* Springer-Verlag, 88–94.

Chazelle, B. 1991. Triangulating a simple polygon in linear time. *Discrete Comput. Geom. 6,* 5, 485–524.

Garey, M. R. and Johnson, D. S. 1979. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.

Goodman, J. E. and O'Rourke, J., Eds. 1997. *Handbook of Discrete and Computational Geometry.* CRC Press LLC, Boca Raton, FL.

Guibas, L. J., Hershberger, J., Leven, D., Sharir, M., and Tarjan, R. E. 1987. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica 2*, 209–233.

Gupta, H. and Wenger, R. 1997. Constructing piecewise linear homeomorphisms of simple polygons. *J. Algorithms 22*, 142–157.

Harel, D. and Tarjan, R. E. 1984. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput. 13,* 2, 338–355.

Kahan, S. and Snoeyink, J. 1996. On the bit complexity of minimum link paths: Superquadratic algorithms for problems solvable in linear time. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.* 151–158.

Kranakis, E. and Urrutia, J. 1995. Isomorphic triangulations with small number of Steiner points. In *Proc. 7th Canad. Conf. Comput. Geom.* 291–296.

Lee, D. T. and Yang, C. D. 1996. Rectilinear paths among rectilinear obstacles. *Discrete Applied Mathematics 70*, 185–215.

Papadopoulou, E. 1999. k-pairs non-crossing shortest paths in a simple polygon. *International Journal of Computational Geometry and Applications 9,* 6.

Saalfeld, A. 1987. Joint triangulations and triangulation maps. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.* 195–204.

Shiloach, Y. 1980. A polynomial solution to the undirected two paths problem. *J. ACM 27,* 3, 445–456.

Souvaine, D. and Wenger, R. 1994. Constructing piecewise linear homeomorphisms. Technical Report 94–52, DIMACS, New Brunswick, New Jersey.

Suri, S. 1986a. A linear time algorithm for minimum link paths inside a simple polygon. *Comput. Vision Graph. Image Process. 35*, 99–110.

Suri, S. 1986b. A linear time algorithm with minimum link paths inside a simple polygon. *Computer Vision Graphics and Image Processing 35,* 1, 99–110.

Takahashi, J., Suzuki, H., and Nishizeki, T. 1992. Algorithms for finding non-crossing paths with minimum total length in plane graphs. In *International Symposium on Algorithms and Computation (ISAAC).*

Takahashi, J., Suzuki, H., and Nishizeki, T. 1993. Finding shortest non-crossing rectilinear paths in plane regions. In *International Symposium on Algorithms and Computation (ISAAC).* 98–107.

Takahashi, J., Suzuki, H., and Nishizeki, T. 1996. Shortest noncrossing paths in plane graphs. *Algorithmica 16,* 3, 339–357.

Yang, C. D., Lee, D. T., and Wong, C. K. 1997. The smallest pair of noncrossing paths in a rectilinear polygon. *IEEE Transactions on Computers 46,* 8, 930–941.