On Solving Geometric Optimization Problems Using Shortest Paths *

Elefterios A. Melissaratos and Diane L. Souvaine
Department of Computer Science, Rutgers University, New Brunswick, NJ 08903

1 Introduction

We have developed techniques which contribute to efficient algorithms for certain geometric optimization problems involving simple polygons: computing minimum separators, maximum inscribed triangles, a minimum circumscribed concave quadrilateral, or a maximum contained triangle. The structure for our algorithms is as follows: a) decompose the initial problem into a low-degree polynomial number of easy optimization problems; b) solve each individual subproblem in constant time using the methods of calculus, standard methods of numerical analysis, or linear programming. The decomposition step uses shortest path trees inside simple polygons (Guibas et. al., 1987) and, in the case of inscribed triangles, produces a new class of polygons, the fan-shaped polygon. By extending the shortest-path algorithm to splinegons, we also generate splinegon-versions of the algorithms for some of the optimization problems.

The problems we discuss fall into four subgroups:

Separators: If two points x and y lie on the boundary of simple polygon P and define a directed line segment $xy \subseteq P$ that separates P into two sets P_L and P_R, then xy is called a *separator*.

> **Minimum length:** The areas of P_L and P_R are defined by constants K_L and K_R. Find a separator of minimum length.

*This research was supported in part by NSF grant CCR-88-03549 and by the NSF DIMACS Science and Technology Center STC88-09648.

> **Minimum sum of ratios:** Find a separator that minimizes the sum of the ratio of the area of P_L and the square of its perimeter and the ratio of the area of P_R and the square of its perimeter.

Inscribed Triangles: Given a simple polygon P, a triangle T such that $T \subseteq P$ and the vertices of T lie on the boundary of P is an *inscribed triangle.*

> **Maximum area/perimeter:** Find the inscribed triangle of maximum area/perimeter.

> **Constrained maximum area/perimeter:** Find a maximum area/perimeter inscribed triangle with one edge of given length.

Circumscribed Quadrilateral: Given a simple polygon P, a quadrilateral Q such that $P \subseteq Q$ and the all four sides of Q intersect the boundary of P is a *circumscribed quadrilateral.*

> **Minimum area concave:** Find the circumscribed concave quadrilateral of minimum area.

Contained Triangle: Given a simple polygon P, a triangle T such that $T \subseteq P$ is a *contained triangle.*

> **Maximum Area:** Find the contained triangle of maximum area.

We solve the separator problems in $O(n^2)$ time, the inscribed triangle problems in $O(n^3)$ time, and the contained triangle problem in $O(n^4)$ time, all in linear space. The quadrilateral algorithm uses either $O(n_c^2 n_p^2)$ time and $O(n)$ space or $O(n_c^2(n_p + k))$ time and $O(n+k)$ space where n_c is the number of vertices of the convex hull of P, $n_p = n - n_c$ and k is an instance-dependent parameter which ranges between $O(n_p)$ and $O(n_p^2)$.

Lisper [23] posed the first separator problem, citing applications in solid modeling and graph cutting. A linear algorithm exists when P is convex [24]. Chang posed the second separator problem [7], claiming applications in finite element analysis. Aggarwal [1] posed the contained triangle problem and the concave circumscribed quadrilateral problem. There exist many related results (e.g. [21],[2],[3],[10]), including a linear-time algorithm for finding the minimum area triangle containing a convex n-gon [25], an $O(nk + n\log n)$ time algorithm for computing the minimum area circumscribing k-gon of a convex n-gon [4],and an $O(n\log n)$ algorithm for finding the minimum perimeter triangle circumscribing a convex n-gon [4]. Note that any convex k-gon circumscribing a simple polygon P also circumscribes the convex hull of P.

Many researchers have studied inclusion problems (e.g. [5],[11],[11],[9],[15]). Some results include a linear-time algorithm for computing the minimum area triangle inscribed in a convex polygon of n vertices [12], an $O(kn + n\log n)$ algorithm for computing the the maximum area or perimeter convex k-gon inside a convex n-gon [3], and $O(n^7)$ time (resp. $O(n^6)$ time) and $O(n^5)$ space algorithms for finding the maximum area (perimeter) convex polygon contained within a given simple polygon P ([6], [7]).

2 Shortest Paths

In [17], the authors describe a linear time and space algorithm for finding the shortest paths from a vertex of a simple polygon to all the other vertices. The union of these paths form a tree called as *the shortest path tree.* In the next few paragraphs we review necessary facts and definitions from that paper and establish our notation. We use these conventions: a simple polygon P has n edges represented by the integers $1, 2, ..., n$ in clockwise order, and edge j has endpoints p_j and p_{j+1}; whenever a subset of polygon vertices (edges) are identified by uppercase (lowercase) letters, alphabetic order implies clockwise order around P; and a line l is tangent to a polygonal chain C if it intersects the chain in one or more points and C lies entirely in one of the halfplanes defined by l.

A point x in a simple polygon P is *visible* from an edge i of P if there exists a point y on i such that $xy \subseteq P$. Two edges i, j of P are *visible* from each other if and only if there exist at least two points x, y on i, j respectively such that $xy \subseteq P$. The set of points $x \in P$ which are visible from an edge i form the *visibility polygon* of P from edge i. If two edges i and j of P are visible from each other, the set of points of j which are visible from i form the *visible part of j with respect to i.* One can compute the visible parts of a given edge i from every other edge of the polygon, as well as the visible parts of every edge from i, in $O(n)$ time and space using the shortest path algorithm [17].

If edges i, j of polygon P are visible from each other then the shortest paths from p_{j+1} to p_i ($SP_{p_{j+1}}(p_i)$) and from p_{i+1} to p_j ($SP_{p_{i+1}}(p_j)$) are inward disjoint convex chains. The region bounded by the above chains and i and j is called an *hourglass* and denoted $H_{i,j}$. Given a point x inside or on the boundary of P which is visible from edge i, the region bounded by i, $SP_{p_{i+1}}(x)$ and $SP_{p_i}(x)$ defines a *funnel* on *base i.* The closest vertex to p_i (p_{i+1} resp.) on the shortest path from x to p_i (p_{i+1} resp.) is called the *anchor* of p_i (p_{i+1} resp.) with respect to x and is denoted by $anchor^x(p_i)$ ($anchor^x(p_{i+1})$ resp.).

The shortest path algorithm [17] applied to a triangulated simple polygon P at a designated start vertex v produces a subdivision where each region corresponds to a *funnel* based on some polygon edge i, denoted $F_v(i)$ (fig. 1). The computation corresponds

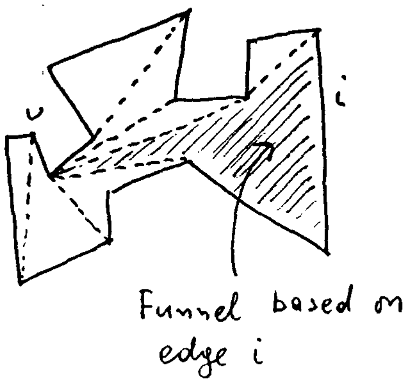

Figure 1: the shortest path tree

to a preorder traversal of the binary tree with one node for each triangle, with an edge joining two nodes whose triangles share an edge, and with the triangle containing s as the root, maintaining the invariant that the funnels have been computed for all edges of frontier nodes of the traversed portion of the tree. A parent funnel is split to form the funnel for its children. Extending the edges of each funnel up to their intersection with the funnel's base produces a refined subdivision of P called the *shortest path map* [17],[20] (fig. 2). The shortest path map from a vertex v subdivides the edges of P. for its children. $S_v(i)$ is the subdivision of a specific edge i and its size is denoted by s_i^v.

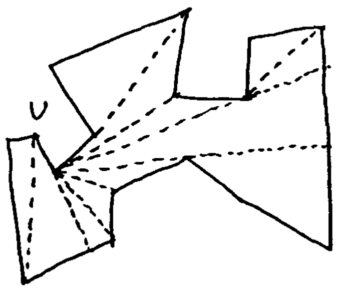

Figure 2: a shortest path map

Minor revisions allow the polygonal shortest path algorithm [17] to work for curved polygons, also known as *splinegons* in comparable time and space bounds. The original algorithm runs on a triangulated polygon. A splinegon cannot necessarily be triangulated without either adding vertices in its interior or adding an unlimited number on its boundary. Fournier and Montuno [22], however, show that triangulation of a simple polygon is linear-time equivalent to solving the all vertex-edge horizontal visibility problem. We prove:

Theorem 2.1 *The shortest path tree algorithm for a splinegon runs in $O(n)$ time given its horizontal visibility decomposition.*

Proof: Preprocess the edges of the splinegon such that each edge is monotone with respect to both the x and the y axes. Decompose the splinegon into horizontal (curved) trapezoids ([28], [14]). Refine the trapezoids so that every component is either a triangle or a quadrilateral with one edge on the splinegon boundary. The dual of the decomposition is now a binary tree. Despite revisions for splitting funnels with curved edges, the recursive formula used in [17] to prove the linearity of the entire algorithm still applies.

3 Separators

If x, y delimit a separator for simple polygon P, then x and y are visible in P. Thus, if x lies on edge i, y on edge j, then i and j are visible in P. Furthermore, the line segment xy lies in the hourglass $H_{i,j}$ defined by the shortest paths from p_{j+1} to p_i ($SP_{p_{j+1}}(p_i)$) and from p_{i+1} to p_j ($SP_{p_{i+1}}(p_j)$). Thus our goal is to reduce the optimum separator problem for a simple polygon to a series of optimum separator problems on hourglasses which are simpler to solve. We focus on the minimum-length separator problem.

Define as a_L (a_R resp.) the area of P to the left (right resp.) of xy (fig. 3). Not every hourglass will admit a separator satisfying the constraints that $a_L = K_L$ and $a_R = K_R$ where $K_L + K_R = A$. Note, however, that $SP_{p_i}(p_{j+1})$ (resp. $SP_{p_j}(p_{i+1})$) cuts the polygon into two or more pieces, those to the left with (combined) area $AL_{p_{j+1}}(p_i)$, and those to the right with area $AL_{p_{i+1}}(p_j)$). A necessary and sufficient condition for H_{ij} to contain a separator xy is $AL_{p_{j+1}}(p_i) \leq K_L$ and $AL_{p_{i+1}}(p_j) \leq K_R$. It is easy to compute $AL_{p_{j+1}}(p_i)$, since the shortest path map from p_{j+1} divides P into a linear number of triangles where the funnel $F_{p_{j+1}}(i)$ on edge i is the disjoint union of some subset of these triangles. Below we present a high level description of our algorithm for computing the minimum-length separator, where *shortestpath(v)* represents the procedure which computes the shortest path tree (map) from vertex v, $a(F)$ denotes the area of funnel F, and *hourglass(i, j, locmin)* computes the minimum-length separator for $H_{i,j}$.

```
For $j = 1$ to $n$ do begin
    $globmin = \infty$;
    $shortestpath(p_j)$;
    $shortestpath(p_{j+1})$;
    $AL_{p_{j+1}}(p_{j+2}) = 0$;
    for $i = 3$ to $n$ do
        $A_{p_{j+1}}(p_{j+i}) = A_{p_{j+1}}(p_{j+i-1}) + a(F_{p_{j+1}}(j+i))$;
    for $i = 1$ to $n$ do
        if $i$, $j$ are visible and $AL_{p_{j+1}}(p_i) \leq K_L$
        then do begin
            $hourglass(i,j,locmin)$
            $globmin = MIN(globmin, locmin)$;
        end;
end;
```

Next, we need a procedure *hourglass(i,j,locmin)* to solve the following problem:

Problem: Given an hourglass $H_{i,j}$ find x and y on i and j respectively such that

a) $xy \subseteq H_{i,j}$, b) the area bounded by $SP_{p_{j+1}}(p_i)$, p_ix, xy, and yp_{j+1} equals $K_L - AL_{p_{j+1}}(p_i)$, and c) the length of xy is minimum.

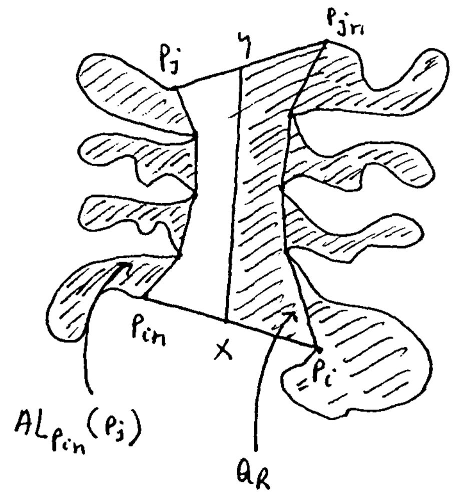

Figure 3: an hourglass divides the polygon

First, we simplify the test of condition b). For each hourglass $H_{i,j}$, define $C_{p_{j+1}}(p_i)$as the area of the region bounded by $SP_{p_{j+1}}(p_i)$ and the segment p_ip_{j+1}. Depth-first-search traversal of the shortest path tree of P from vertex p_{j+1} produces all of the $C_{p_{j+1}}(p_i)$ in linear time. Then the test of condition b) reduces to determining whether the quadrilateral p_ixyp_{j+1} has area $K_L - AL_{p_{j+1}}(p_i) + C_{p_{j+1}}(p_i)$, from now on referred to as K.

Condition a) is satisfied if and only if the closed halfplane to the left of the $\overrightarrow{xy}$ contains all the vertices of $SP_{p_{j+1}}(p_i)$ and the one to the right contains all the vertices of $SP_{p_{i+1}}(p_j)$. To reduce the number of constraints, we exploit the fact that shortest paths are convex to decompose the problem further. First, trim edge i to create an edge i' so that all of i' is visible from j. Next, subdivide i' by merging $S_{p_{j+1}}(i)$ and $S_{p_j}(i)$. As x moves from p'_i to p'_{i+1}, $anchor^{p_{j+1}}(x)$ and/or $anchor^{p_j}(x)$ changes only when x moves from one subinterval to the next. Consequently, we can reduce an arbitrary hourglass problem to a series of problems defined on elementary hourglasses:

> **Problem:** For each interval I of i, find points $x = (x_1, y_1)$ and $y = (x_2, y_2)$ on I and j respectively such that a) the $anchor^{p_j}(x)$ and $anchor^{p_{j+1}}(x)$ do not lie in the same open halfspace defined by $\overrightarrow{xy}$, b) the area of the quadrilateral p_ixyp_{j+1} equals K, and c) length of xy is minimum.

This is a continuous optimization problem, rather than a combinatorial one. Algebraic manipulation of the constraints involved leads to an expression for the length of xy as the square root of a rational a function of x_1 where the domain of x_1 is restricted by the facts that x must lie within a specific subinterval I, the point y lies on segment j, and the line through x and y must keep the $anchor^{p_j}(x)$ and $anchor^{p_{j+1}}(x)$ on opposite sides. The degrees of the numerator and denominator of the rational function permit analytical solution for finding the optimum in constant time. The global minimum for the original hourglass is the minimum of all the minima obtained from the continuous problems on elementary hourglasses. Therefore:

Theorem 3.1 *The hourglass problem can be solved in $O(n)$ time and space where n is the number of hourglass vertices.*

Since the hourglass algorithm takes linear time and since we call the hourglass algorithm at most $O(n^2)$ times then our algorithm has at most $O(n^3)$ complexity. We exploit the linearity of the shortest path to improve the bound.

Theorem 3.2 *The minimum length area separator of a simple polygon can be computed in $O(n^2)$ time and $O(n)$ space.*

Proof: The test in the if statement can be done in $O(1)$ time since the area parameters of P as well as the visibility of i, j have been computed during the shortest path computation. The hourglass algorithm takes $O(s_j^{p_i} + s_j^{p_{i+1}})$. Thus the whole algorithm takes $O(\sum_{i=1}^n \sum_{j=1}^n (s_j^{p_i} + s_j^{p_{i+1}}))$ time which is $O(n^2)$ since $\sum_{k=1}^n s_k^v = O(n)$ for all v. □

Corollary 3.3 *The minimum length area separator of a simple splinegon can be computed in $O(n^2)$ time and $O(n)$ space.* □

Theorem 3.4 *The minimum sum of ratios separator of a simple polygon can be computed in $O(n^2)$ time and $O(n)$ space.* □

Remark: Although the results in this section were explained using hourglasses, the same results can be achieved with an alternate formulation the details of which are described in the full paper. The idea is to fix a particular edge i, find the shortest path maps from its endpoints. These maps subdivide the boundary of P into a set of segments. As a point x moves on a particular segment $anchor^{p_i}(x)$ and $anchor^{p_{i+1}}(x)$ remain constant. Let $a = anchor^{p_i}(x)$

and $b = anchor^{p_{i+1}}(x)$. Consider only segments for which $SP_{p_i}(a)$ and $SP_{p_{i+1}}(b)$ are inward convex (these are the visible ones) and for each of them solve a continuous optimization problem. Repeat the same process for all edges i.

4 Inscribed Triangles

For three points x, y, z on the boundary of a simple polygon P to define an inscribed triangle, it is necessary and sufficient that they are pairwise visible. If x, y, z lie on edges k, j, i, respectively, then the points are pairwise visible if and only if xy, yz and zx lie inside $H_{k,j}$, $H_{j,i}$ and $H_{k,i}$, respectively. Thus the boundary of the triangle xyz is interior both to P and to the union of the three hourglasses. Since P is simple, the entire triangle must be interior to P. It is also contained in the polygon $F_{i,j,k} \subseteq P$ bounded by i, $SP_{p_{i+1}}(p_j)$, j, $SP_{p_{j+1}}(p_k)$, k and $SP_{p_{k+1}}(p_i)$.

$F_{i,j,k}$ is called a *fan-shaped polygon* with *bases* i, j, k (fig.4). $F_{i,j,k}$ is *legal* iff the visible parts of each of its bases with respect to the two others have non-empty intersection. A triangle T is *inscribed* in $F_{i,j,k}$ if and only if if $T \subseteq F_{i,j,k}$ and the vertices of T lie on the bases. Every triangle inscribed in a simple polygon P is also inscribed in a fan-shaped polygon $F_{i,j,k} \subseteq P$. Below we present a high level description of our algorithm for computing the maximum area/perimeter inscribed triangle, where *shortestpath(v)* represents the procedure which computes the shortest path tree (map) from vertex v, and $fan(i, j, k, locmax)$ computes the maximum triangle inscribed in a legal $F_{i,j,k}$:

```
globmax = 0;
for $i = 1$ to $n$ do begin
    shortestpath($p_i$); shortestpath($p_{i+1}$);
    for $j = 1$ to $n$ do begin
        shortestpath($p_j$); shortestpath($p_{j+1}$);
        for $k = 1$ to $n$ do begin
            if $F_{i,j,k}$ is legal
            then fan(i,j,k, locmax);
            globmax = MAX(globmax, locmax);
        end;
    end;
end;
```

It remains to develop procedure $fan(i, j, k, locmax)$.

Lemma 4.1 *For a fan-shaped polygon $F_{i,j,k}$ the maximum-area inscribed triangle with vertices x, y, z on the bases must have at least two sides tangent to the convex chains of the fan (fig. 4).*

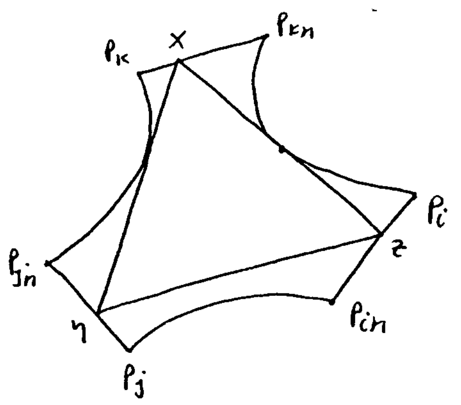

Figure 4: fan-shaped polygon

Proof: By contradiction. In specific, assume that neither xy nor xz is tangent to the boundary of $F_{i,j,k}$. Tangents from y and z to the chains $SP_{p_{j+1}}(p_k)$ and $SP_{p_i}(p_{k+1})$, respectively, intersect k at points v and w such that x must lie between them. Then it is clear that one of vyz or wyz must have area greater than or equal to the area of xyz (the equality happens when k is parallel to yz). □

Lemma 4.2 *The maximum-perimeter triangle inscribed in a fan-shaped polygon must have at least two sides tangent to the convex chains of the fan.* □

Assume that the optimum triangle xyz has xy tangent to $SP_{p_{j+1}}(p_k)$, and xz tangent to $SP_{p_i}(p_{k+1})$. To find the optimum, trim edge k to create an edge k' which is visible from both i and j. If k' is empty then inscribed triangle does not exist. Perform the comparable operation on edges i and j, creating i' and j' respectively. Next, subdivide k' by merging $S_{p_{j+1}}(k)$ and $S_{p_i}(k)$. As x moves from p'_k to p'_{k+1}, $anchor^{p_{j+1}}(x)$ and/or $anchor^{p_i}(x)$ changes only when x moves from one subinterval to the next.

Problem: For each interval K of k', find points $x \in K$, $y \in i$, $z \in j$ such that a) $anchor^{p_i}(x) \in xy$ and $anchor^{p_{j+1}}(x) \in xz$ b) y and z are mutually visible, and c) area/perimeter of xyz is maximum.

Testing condition b) is still complicated, so we decompose the problem further. Subdivide i' according to $S_{p_j}(i)$ and subdivide j' according to $S_{p_{i+1}}(j)$. As y (z resp.) moves from p'_i to p'_{i+1} (p'_j to p'_{j+1} resp.) $anchor^{p_j}(y)$ ($anchor^{p_{i+1}}(z)$ resp.) changes only when y (z resp.) moves from one subinterval to the next.

Let W be the number of vertices of either $S_{p_j}(i)$ or $S_{p_{i+1}}(j)$. To each interval of i' (resp. j') assign a number called its rank which corresponds to the position of its anchor in $SP_{p_i}(p_j)$ (resp. $SP_{p_j}(p_i)$), assuming that the first position is 0.

Refine the subdivision of k' so that whenever x belongs to some interval K, y and z each have constant rank. The rank of K equals the sum of those ranks. If $rank(K) < W$, do nothing; if $rank(K) > W$, solve Problem I; if $rank(K) = W$ solve Problem II.

> **Problem I:** Given three non intersecting line segments AB, CD, and EF and two points p, q such that p (q resp.) lies on AE and BF (AC and BD resp.), find s,t,u on AB,EF,CD with p (q resp.) on st (su resp.), such that the area of stu is maximum.

> **Problem II:** Add the constraint that the line through t and u should be always above a constant point (x_0, y_0).

Solutions to both problems can be computed analytically. Solving the perimeter optimization versions of these problems requires numerical methods.

Lemma 4.3 *The maximum inscribed triangle in a fan-shaped polygon $F_{i,j,k}$ can be found in $O(s_k^{p_i} + s_k^{p_{j+1}} + s_j^{p_{i+1}} + s_i^{p_j})$ which is $O(n)$ where n is the number of vertices of the fan-shaped polygon.* □

Since we decompose the simple polygon into at most $O(n^3)$ fan-shaped polygons, the global algorithm uses at most $O(n^4)$ time. Careful analysis produces a better bound.

Theorem 4.4 *The maximum triangle inscribed in a simple polygon P can be found in $O(n^3)$ arithmetic operations where n is the number of vertices of P. The space required is $O(n)$.*

Proof: The total time spent in the shortest path computation is $O(n^3)$ since the shortest path procedure is called $O(n^2)$ times. The *if* statement takes $O(1)$ time since whether $F_{i,j,k}$ is legal can be decided from the shortest path information. According to Lemma 4.3, the procedure *fan(i,j,k)* takes $O(s_k^{p_i} + s_k^{p_{j+1}} + s_j^{p_{i+1}} + s_i^{p_j})$. Thus the total time spent in the fan-shaped polygons corresponding to all triples (i, j, k) of edges of P is:

$O(\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} (s_k^{p_i} + s_k^{p_{j+1}} + s_j^{p_{i+1}} + s_i^{p_j}))$ which is $O(n^3)$ since $\sum_{k=1}^{n} s_k^v = O(n)$ for any vertex v of P according to [17]. □

Unfortunately, the maximum area or perimeter triangle inscribed in a simple splinegon might not have two sides tangent to the chains of the fan-shaped polygon. Consequently, Lemmas 4.1 and 4.2 do not hold, and a new algorithm is necessary:

Theorem 4.5 *The maximum area or perimeter triangle inscribed in a simple splinegon can be found in $O(n^4)$ time and $O(n)$ space.* □

The maximum inscribed triangle with one of its sides constrained to have given length does have at least one of the non-given length sides tangent to a fan-shaped polygon:

Theorem 4.6 *The Maximum Inscribed Triangle with one of its sides having given length can be solved in $O(n^3)$ time and $O(n)$ space.* □

Remark: Although the results in this section were explained using fan-shaped polygons, the same results can be achieved with an alternate formulation described in the full paper.

5 Minimum Area Concave Quadrilateral

$ABCD$ always represents a concave quadrilateral where C is the reflex vertex. $CH(P)$ represents the convex hull of a simple polygon P. Each simple polygonal region Q interior to $CH(P)$ but exterior to P is called a *pocket* of P.

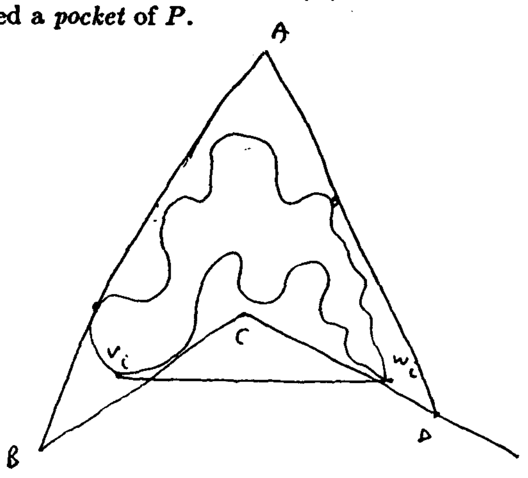

Figure 5: Optimum quadrilateral characterization

Lemma 5.1 *The minimum-area concave quadrilateral $ABCD$ containing a simple polygon P satisfies* *the following conditions: a) A, B, D lie outside of the $CH(P)$ with AB and AD tangent to $CH(P)$ at some points k and l, respectively; b) C is inside the visibility polygon VQ_i of some pocket Q_i of P with respect to pseudoedge v_iw_i; c) BC, DC are tangent to the boundary of Q_i at points a, b where a and b are the anchors of the shortest paths from v_i to C and w_i to C, respectively, inside the pocket. Note that C need not lie on the boundary of the pocket and that the minimum-area concave quadrilateral may be degenerate (i.e. a triangle) (figs. 5 and 6).* □

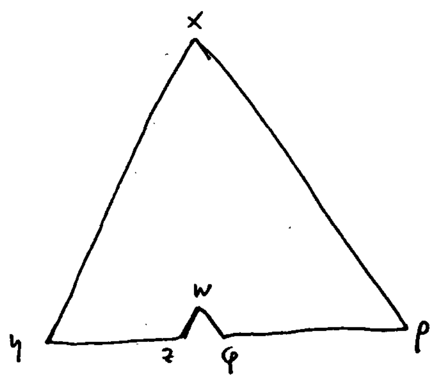

Figure 6: Optimum quadrilateral has C collinear with B and D

A description of the algorithm follows:

```
Compute $CH(P)$ and let $p =$ the number of pockets.
Triangulate $P$ and all its pockets $Q_i$, for $i = 1...p$.
For all pairs $a$, $b$ of vertices of $CH(P)$ do
    for $i = 1$ to $p$ do begin
        Compute $VQ_i$ and the shortest path maps
          inside $VQ_i$ from both $v_i$ and $w_i$ [17].
        Merge those maps [18] and label each region
          ($\leq 6$ sides) with its anchors w.r.t. $v_i$ and $w_i$.
        For every region $R$, call pocket($a, b, k, l, i, R$) .
    end.
Report the optimum.
```

Note that instead of explicitly merging the two maps, we can take each pair (r_1, r_2) where r_1 (resp. r_2) is a region of the shortest path map from v_i (resp. w_i), and calculate the intersection region explicitly. For each such intersection region call *pocket*(a, b, k, l, i, R). Since every region of the shortest path map is a triangle the intersection of two such regions has constant number of sides. The space required is the space to keep the two shortest path maps, i.e linear. The procedure *pocket*(a, b, k, l, i, R) must solve the following optimization problem:

> **Problem:** Construct the minimum area concave quadrilateral $ABCD$ such that AB and AD pass through the constant points a and b respectively, and have slopes in the intervals defined by the adjacent convex hull edges, CD and CB pass through k and l, and $C \in R$.

These constraints generate a linear program in eight variables which is subject to a constant number of linear constraints. That problem can be solved in $O(1)$ time. Thus, each pass through the loop above takes $O(n_i + k_i)$ time where n_i is the size of Q_i and k_i is the size of the subdivision created by merging the shortest path maps from v and w. Thus, if n_c are the number of vertices of $CH(P)$ and n_p the total number of vertices of all pockets of P and k be the sum of the merged subdivisions over all pockets, then we have the following theorem:

Theorem 5.2 *The minimum concave quadrilateral that contains a simple polygon P can be found either in $O(n_c^2 n_p^2)$ time and $O(n)$ space or in $O(n_c^2(n_p + k))$ time and $O(n + k)$ space.* □

6 Contained Triangles

The maximum area triangle $T = xyz$ contained in a simple polygon P may have 0, 1, 2 or 3 vertices on the boundary of P. The case of 3 vertices of the triangle on the boundary of P corresponds to the maximum area inscribed triangle problem solved in $O(n^3)$ time and $O(n)$ space earlier. We focus here on what we call 0-case, 1-case, and 2-case.

Lemma 6.1 *Let A and C (B and D resp.) be two points on $\overrightarrow{Ox}$ ($\overrightarrow{Oy}$ resp.) such that segments AB and CD intersect inside the wedge defined by Ox and Oy at point E. Let FG be a line segment through E with F (G resp.) between A and C (D and B resp.). The area of the triangle OFG is maximized when one of the following holds: $F = A$ and $G = B$; or $F = C$ and $G = D$ (fig. 7 a).* □

Corollary 6.2 *Let $\overrightarrow{Ox}$ and $\overrightarrow{Oy}$ be two rays with common origin O. Let also A and B be two points on $\overrightarrow{Ox}$ and $\overrightarrow{Oy}$ respectively and let C_{AB} be an inward convex chain. Let D (E resp.) lie on OA (OB resp.)*

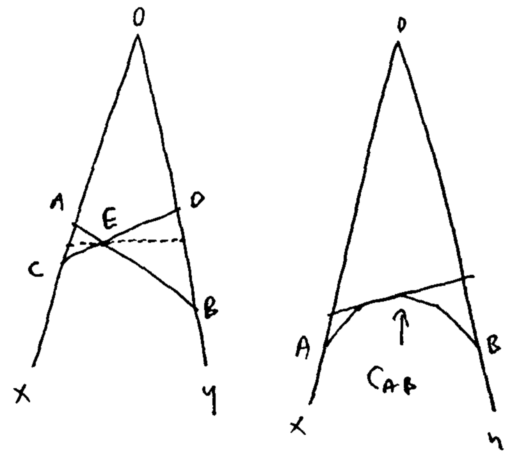

Figure 7: Examples for lemma and corollary

such that O and C_{AB} do not lie on the same side of the line through D and E. Then the area of triangle ODE becomes maximum if DE contains an edge of the convex chain C_{AB} (fig. 7 b).

Proof: Assume that DE does not contain any edge of C_{AB}. Then either a) the intersection of DE and C_{AB} is one vertex of C_{AB} or b) is the empty set. In case of a) we have an instance of Lemma 6.1. In case of b) we can translate DE in a direction perpedicular to itself until it intersects C_{AB} and then apply Lemma 6.1. □.

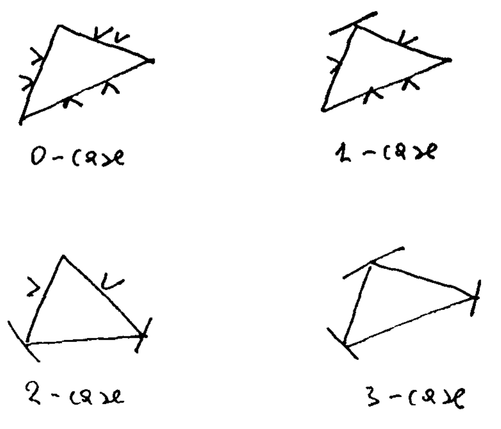

Figure 8: a) 0-case; b) 1-case; c) 2-case; d) 3-case

Lemma 6.3 *Let T be a maximum area triangle. Then each edge of T contains at least two points of the boundary of P. Specifically the following hold (fig. 8): If T is 0-case then each edge of T contains at least two reflex vertices of polygon P; if T is 1-case with x on the boundary of P, then yz touches at least two reflex vertices of P and xy and xz at least one. if T is 2-case with y and z on edges i and j, there exists at least one edge k of P such that i, j, k define a fan-shaped polygon $F_{i,j,k} \supseteq T$.*

0-Case Triangle Algorithm. One algorithm would consider all triples of pairs of reflex vertices ($O(n^6)$ objects), check whether the corresponding triangle is contained in P in $O(\log n)$ time using ray-shooting [8] [17], and choose the largest one, a total of $O(n^6 \log n)$ arithmetic operations. An alternate algorithm would fix two sides of the candidate triangle by choosing a pair of pairs of reflex vertices (A, B) and (E, F), assuming that xy contains AB and xz contains EF, and spending linear time to find the optimum position of yz, for a total of $O(n^5)$ arithmetic operations.

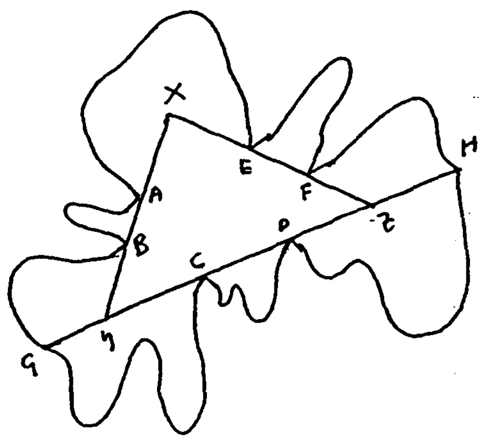

Figure 9: 0-case

Using the linearity of shortest path trees inside simple polygons, we can invert this algorithm and reduce the complexity by an order of magnitude (fig. 9). Fix a pair of reflex vertices C and D with the characteristic that $\overline{CD} \subset P$ and all edges incident to C and D lie on the same side of the line containing $\overline{CD}$ and assume that side yz contains these vertices. Determine in $O(\log n)$ time [8], [17] the points G and H closest to C and D, respectively, where the extension of $\overline{CD}$ intersects the boundary of P. Let P' represent the subpolygon of P which lies at the opposite side of GH from the edges incident to C and D. Since x must be visible from GH, then the shortest paths from G to x and from H to x inside P must be inward convex chains containing segments AB and EF, respectively, where A, B, E, F are reflex vertices and $\overrightarrow{AB}$ and $\overrightarrow{EF}$

intersect within P at point x. Thus, we need consider only pairs of the $O(n)$ edges of the shortest path tree from G and the $O(n)$ edges of the shortest path tree from H, a total of $O(n^2)$ objects.

How can we test efficiently whether the chosen pair AB and EF of shortest path tree edges forms a legal triangle with GH? Choose only those AB such that a) the shortest paths from G to A and G to B are inward convex chains, b) points y and z do not lie in segment CD, and c) segments Ax and Ex lie inside P. Conditions a) and b) clearly can be checked in $O(1)$ time. One way to test condition c) is to apply ray shooting inside P in $O(\log n)$ time. Since AB and CD are shortest-path tree edges, however, constant time suffices. Define e_1 (resp. e_2) as the edge of the shortest path map from G which is adjacent to vertex A (resp. E) and collinear to AB (resp. EF). If e_1 and e_2 both exist and intersect then the intersection point is a valid vertex x. Repeating the above procedure for all the $O(n^2)$ pairs of reflex vertices C and D yields:

Lemma 6.4 *The 0-case maximum triangle can be found in $O(n^4)$ time and $O(n)$ space.*

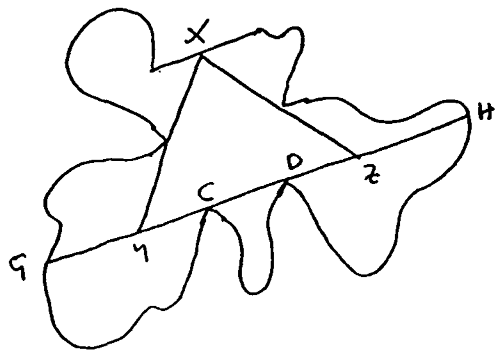

Figure 10: 1-case

1-Case Triangle Algorithm. As in the 0-case algorithm we fix a pair of reflex vertices C and D (fig. 10). We find again points G and H as defined previously and then we have to walk on the shortest path maps of G and H on the boundary of P as we did in the inscribed triangle case of Sect. 4. Thus for a fixed pair of reflex vertices we spend, using similar arguments, $O(n)$ time and therefore a total $O(n^3)$ for the whole problem. Thus:

Lemma 6.5 *The 1-case maximum triangle can be found in $O(n^3)$ time and $O(n)$ space.* □

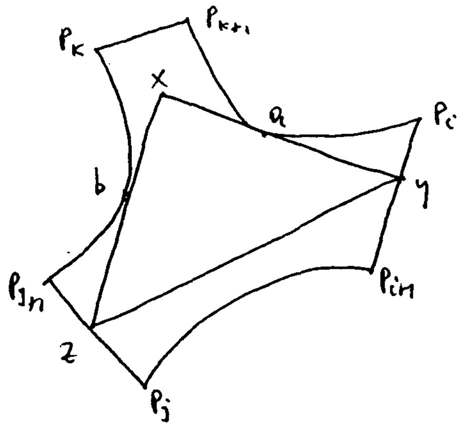

Figure 11: 2-case

2-Case Triangle Algorithm. According to *Lemma 6.3*, triangle xyz lies in a fan-shaped polygon (see fig. 11). Subdivide j (i resp.) according to the shortest path maps from both p_{i+1} and p_k (p_j and p_{k+1} resp.). For each interval on the subdivision of edge i and each interval of the subdivision of edge j, a) check whether points y and z are visible, using techniques developed in Sect. 4; b) let $a = anchor^{p_{k+1}}(y)$ and $b = anchor^{p_k}(z)$ and check whether $SP_{p_{k+1}}(a)$ and $SP_{p_k}(b)$ are inward convex; c) check whether the intersection x of the lines through segments ya and zb lies "below" the line through edge k. That guarantees that triangle xyz lies inside $F_{i,j,k}$. d) Solve the appropriate continuous optimization problem. It should be clear that steps a) through d) take $O(1)$ time.

According to the above discussion the time complexity per fan-shaped polygon is $O((s_j^{p_{i+1}}+s_j^{p_k})(s_i^{p_j}+s_i^{p_{k+1}}))$. Then summing over all fan-shaped polygons we get

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} O((s_j^{p_{i+1}} + s_j^{p_k})(s_i^{p_j} + s_i^{p_{k+1}})) = O(n^4)$$

□

Lemma 6.6 *The 2-case maximum triangle can be found in $O(n^4)$ time and $O(n)$ space.*

Theorem 6.7 *The maximum area triangle contained in a simple polygon can be found in $O(n^4)$ time and $O(n)$ space.*

References

[1] A. Aggarwal, "Lecture notes in Computational Geometry." *MIT Research Seminar Series MIT/LCS/RSS 3*, August, 1988.

[2] A. Aggarwal, J. S. Chang and C. K. Yap, "Minimum Area Circumscribing Polygons." *Visual Computer*, 1 (1985), 112-117.

[3] A. Aggarwal, M. Klawe, S. Moran, P. Shor, R. Wilber, "Geometric Applications to a Matrix Searching Algorithm." *Algorithmica*, 2 (1987), 209-233.

[4] A. Aggarwal, J. Park, "Notes on Searching in Multidimensional Monotone Arrays." *Proc. 29th IEEE Symp. on Found. of Comp. Sci.* (1988), 497-512.

[5] J.E. Boyce, D.P. Dobkin, R.L. Drysdale, L.J. Guibas, "Finding Extremal Polygons." *SIAM J. of Computing*, 14 (1985), 134-147.

[6] J. S. Chang, C. K. Yap, "A Polynomial Solution for Potato-Peeling and Other Polygon Inclusion and Enclosure Problems." *Discrete and Comp. Geom.*, 1 (1986), 155-182.

[7] J. S. Chang, "Polygon Optimization Problems." Ph.D. Thesis, New York University, 1986.

[8] B. Chazelle, L. Guibas, "Visibility and Intersection Problems in Plane Geometry." *Proc. of ACM Symp. Comp. Geom.* (1985), 135-146.

[9] P. Chew, K. Kedem, "Placing the Largest Similar Copy of a Convex Polygon among Polygonal Obstacles." *Proc. of ACM Symp. Comp. Geom.* (1989), 167-74.

[10] N.A.A. DePano, "Polygon Approximation with Optimized Polygonal Enclosures: Applications and Algorithms." Ph.D. thesis, Dept of Computer Science, Johns Hopkins University, April 1988.

[11] N.A. DePano, Yan Ke, J. O'Rourke, "Finding Largest Inscribed Equilateral Triangles and Squares." *Proc. of the Allerton Conference*, 1987.

[12] D. Dobkin, L. Snyder, "On a General Method for Maximizing and Minimizing among Certain Geometric Problems." *Proc. 20th IEEE Symp. on Found. of Comp. Sci.* (1979), 9-17.

[13] D. Dobkin, D. Souvaine, "Computational Geometry in a Curved World." *Algorithmica*, 5 (1990).

[14] D. Dobkin, D. Souvaine, C. Van Wyk, "Decomposition and Intersection of Splinegons." *Algorithmica*, 3 (1988), 473-485.

[15] S. Fortune, "A Fast Algorithm For Polygon Containment By Translation." *Proc. 13th ICALP* (1985), 189-198.

[16] M. Garey, D. Johnson, F. Preparata, R. Tarjan, "Triangulation of a simple polygon." *Information Processing Letters*, 7 (1978), 175-179.

[17] L. Guibas, J. Hershberger, D. Leven, M. Sharir, R. Tarjan, "Linear Time Algorithms for Visibility and Shortest Path Problems inside Triangulated Simple Polygons." *Algorithmica*, 2 (1987), 209-233.

[18] L. Guibas, R. Seidel. "Computing Convolutions via Reciprocal Search." *Discrete and Computational Geometry*, 2 (1988), 175-193.

[19] L. Guibas, J. Stolfi, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams, " *ACM Trans. Graphics*, 4 (1985), 74-123.

[20] J. Hershberger, "An Optimal Visibility Graph Algorithm for Triangulated Simple polygons." *Algorithmica*, 4 (1989), 141-155.

[21] V. Klee and M. Laskowski, "Finding the Smallest Triangles Containing a Given Convex Polygon." *J. Algorithms*, 6 (1985), 359-375.

[22] A. Fournier and D. Y. Montuno, "Triangulating Simple Polygons and Equivalent Problems." *ACM Transactions on Graphics*, 3 (1984), 153-74.

[23] B. Lisper, TheoryNet posting and followup communication, July, 1988.

[24] J. D. Mittleman, D. L. Souvaine, "Shortest Area-Bisector of a Convex Polygon." Rutgers University Technical Report LCSR-TR-139, November 1989.

[25] J. O' Rourke, A. Aggarwal, S. Maddila, M. Baldwin, "An Optimal Algorithm for Finding Minimal Enclosing Triangles." *J. Algorithms*, 7 (1986), 258-269.

[26] M. H. Overmars, J. van Leeuwen, "Maintenance of Configurations in the Plane." *J. Comput. System Sci.*, 23 (1981), 166-204.

[27] D. L. Souvaine, "Computational Geometry in a Curved World." Ph.D. Thesis, Princeton University, October, 1986.

[28] R.E.Tarjan, C. Van Wyk, "Triangulation of a Simple Polygon." *SIAM Journal of Computing*, 17 (1988), 143-178.