Discrete Applied Mathematics 31 (1991) 97–111
North-Holland

Computing external farthest neighbors for a simple polygon*

Pankaj K. Agarwal
Courant Institute of Mathematical Sciences, New York, NY 10012, USA

Alok Aggarwal
IBM Research Division, Yorktown Heights, NY 10598, USA

Boris Aronov
Courant Institute of Mathematical Sciences, New York, NY 10012, USA

S. Rao Kosaraju
The Johns Hopkins University, Baltimore, MD 21218, USA

Baruch Schieber
IBM Research Division, Yorktown Heights, NY 10598, USA

Subhash Suri
Bell Communication Research, Morristown, NJ 07960, USA

Received 21 August 1989
Revised 15 March 1990

Abstract

Agarwal, P.K., A. Aggarwal, B. Aronov, S.R. Kosaraju, B. Schieber and S. Suri, Computing external farthest neighbors for a simple polygon, Discrete Applied Mathematics 31 (1991) 97–111.

Let $\mathscr{P}$ be (the boundary of) a simple polygon with n vertices. For a vertex p of $\mathscr{P}$, let $\phi(p)$ be the set of points on $\mathscr{P}$ that are farthest from p, where the distance between two points is the length of the (Euclidean) shortest path that connects them without intersecting the interior of $\mathscr{P}$. In this paper, we present an $O(n \log n)$ algorithm to compute a member of $\phi(p)$ for every vertex p of $\mathscr{P}$. As a corollary, the external diameter of $\mathscr{P}$ can also be computed in the same time.

* Work on this paper by the first author has been supported by Office of Naval Research Grant N00014-87-K-0129, by National Science Foundation Grant No. NSF-DCR-83-20085, by grants from the Digital Equipment Corporation, and the IBM Corporation. Work on this paper by the third author has been supported by an AT&T Bell Laboratories PhD Scholarship. Work by the fourth author has been supported by NSF Grant CCR-8804284.

1. Introduction

Shortest path planning in an obstacle-cluttered environment is a classical problem in computational geometry. In two dimensions, it is common to model the obstacles by closed simple polygons $\mathscr{P}_1, \ldots, \mathscr{P}_m$ whose interiors represent forbidden regions, and to model a robot by a point. A *feasible* path of the robot in such an environment is a path that does not meet the interior of the obstacles. Given two points in the plane, the distance between them is the length of the shortest feasible path connecting them; if either of the points lies inside an obstacle, this distance is assumed to be infinite.

The set $\bigcup_i \mathscr{P}_i$ is called the *obstacle space.* A useful measure of the obstacle space is the maximum distance between any two points lying on its boundary. This maximum distance is called the *diameter* of the obstacle space. In this paper, we are concerned with the case of a single obstacle.

Given a simple n-gon $\mathscr{P}$ and two points p and q on its boundary, we define the *external distance* between p and q to be the length of a shortest path that joins p and q without intersecting the interior of $\mathscr{P}$. For a vertex p of $\mathscr{P}$, let $\phi(p)$ be the set of points on (the boundary of) $\mathscr{P}$ that are farthest from p with respect to the external distance, and let the *external diameter* of $\mathscr{P}$ be the maximum external distance between any two points of $\mathscr{P}$. The problem of computing the external diameter was first considered by Samuel and Toussaint [7]. They presented an $O(n^2)$ time algorithm to solve the problem, where n denotes the number of vertices of $\mathscr{P}$.

In this paper we provide an $O(n \log n)$ time and $O(n)$ space algorithm for computing a point of $\phi(p)$ for every vertex p of $\mathscr{P}$. A result of Samuel and Toussaint [7] then shows that the external diameter of $\mathscr{P}$ can also be computed in time $O(n \log n)$.

This paper is divided into five sections. Section 2 discusses the basic geometric concepts that we use in this paper. In Section 3 we prove some properties of external shortest paths, which lead us to an efficient algorithm for computing the external farthest neighbors for every vertex of the polygon. Section 4 describes an $O(n \log n)$ algorithm to compute external farthest neighbors. We conclude with some final remarks in Section 5.

2. Geometric preliminaries

Let $\mathscr{P}$ be (the boundary of) a simple polygon with n vertices. Let $p_0, p_1, \ldots, p_{n-1}$ denote the vertices[1] of $\mathscr{P}$ ordered in clockwise direction around it. For distinct points $p, q \in \mathscr{P}$, let $\mathscr{P}[p,q]$ be the section of $\mathscr{P}$ clockwise between p and q inclusive. Define $\mathscr{P}(p,q) = \mathscr{P}[p,q] - \{p,q\}$; define $\mathscr{P}[p,q)$ and $\mathscr{P}(p,q]$ similarly. Let $\mathscr{CH}(\mathscr{P})$

[1] Indices of p_i are modulo n.

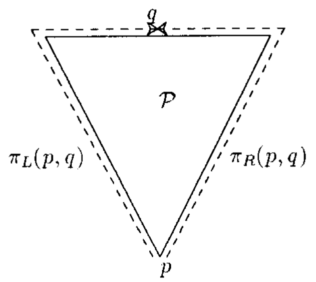

Fig. 1. Left and right shortest paths.

denote the boundary of the convex hull of $\mathscr{P}$, and $q_0, q_1, \ldots, q_{m-1}$ the vertices[2] of $\mathscr{P}$, lying on $\mathscr{CH}(\mathscr{P})$ in clockwise order. Without loss of generality we can assume that $q_0 = p_0$. $\mathscr{P}[q_i, q_{i+1})$ is denoted by $\mathscr{K}_i$ and, if $\mathscr{P}[q_i, q_{i+1}] \neq \overline{q_i q_{i+1}}$, the closed polygonal region bounded by $\mathscr{K}_i$ and $\overline{q_i q_{i+1}}$ is denoted by $\overline{\mathscr{K}_i}$. If $\overline{q_i q_{i+1}} = \mathscr{P}[q_i, q_{i+1}]$, then let $\overline{\mathscr{K}_i} = \overline{q_i q_{i+1}}$. Somewhat abusing the notation, we will refer to either $\mathscr{K}_i$ or $\overline{\mathscr{K}_i}$ as a *pocket*. Note that pockets $\mathscr{K}_0, \mathscr{K}_1, \ldots, \mathscr{K}_{m-1}$ form a partition of $\mathscr{P}$. An edge $\overline{q_i q_{i+1}} \subset \mathscr{CH}(\mathscr{P})$ is called the *lid* of a pocket and the points q_i and q_{i+1}, *lid vertices* of $\mathscr{K}_i$ (or of $\overline{\mathscr{K}_i}$).

For a pair of points p, q not lying in the interior of $\mathscr{P}$, a path from p to q is an *external path* if it avoids the interior of $\mathscr{P}$. The *external distance* $d(p,q)$ between two points p and q is the length of a shortest external path connecting p to q (that such a path always exists can be shown by an easy compactness argument). Since the exterior of $\mathscr{P}$ in the plane is not simply connected, there are infinitely many homotopy classes of paths connecting two points p and q on $\mathscr{P}$. Informally, the homotopy class of such a path determines how many times it winds around $\mathscr{P}$ and the direction in which it winds around $\mathscr{P}$. It is not difficult to show that any path connecting $p \in \mathscr{P}$ to $q \in \mathscr{P}$ that winds around $\mathscr{P}$ more than one complete revolution must self-intersect and thus cannot be a (globally) shortest external path between p and $q \neq p$. We will therefore restrict our attention to paths homotopic to $\mathscr{P}[p,q]$ and $\mathscr{P}[q,p]$, referring to them as *left* and *right* paths from p to q, respectively. If $p = q$, then notion of left and right paths is undefined. For an external path π, let $|\pi|$ denote its length in L_2 metric. For distinct $p, q \in \mathscr{P}$, we define $\pi_L(p,q)$ (respectively $\pi_R(p,q)$) to be the shortest left (respectively right) path from p to q. (Lemma 3.2 below shows that there is a *unique* shortest left (respectively right) path from p to q.) Note that $\pi_L(p,q) = \pi_R(q,p)$, for any pair of distinct points $p, q \in \mathscr{P}$. Let $d_L(p,q) = |\pi_L(p,q)|$ and $d_R(p,q) = |\pi_R(p,q)|$. As previous discussion indicates, there are at most two distinct external shortest paths between two distinct points $p, q \in \mathscr{P}$, namely $\pi_L(p,q)$ and $\pi_R(p,q)$ (see Fig. 1). In particular,

$$d(p,q) = \min\{d_L(p,q), d_R(p,q)\}.$$

[2] Indices of q_j are modulo m.

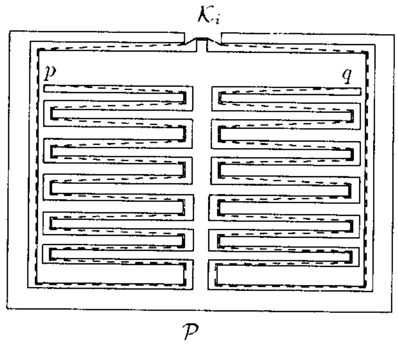

Fig. 2. External diameter is realized by a pair of points p, q lying in the same pocket.

If $d_L(p,q) \le d_R(p,q)$, define $\pi(p,q) = \pi_L(p,q)$, otherwise $\pi(p,q) = \pi_R(p,q)$. Define $\pi(p,p)$ as the "null path" from p to p.

For a given point $p \in \mathscr{P}$, an *external farthest neighbor* of p is a point $q \in \mathscr{P}$ such that

$$d(p,q) = \sup\{d(p,p') \mid p' \in \mathscr{P}\}.$$

Let $\phi(p)$ denote the set of all external farthest neighbors of p. We define a more specialized notion of external farthest neighbors. Let $\phi'(p)$ denote the set of the farthest neighbors of p among points not lying in the same pocket as p. Formally, if $p \in \mathscr{K}_i$, then

$$\phi'(p) = \{x \in \mathscr{P}[q_{i+1}, q_i] \mid \forall y \in \mathscr{P}[q_{i+1}, q_i],\ d(p,x) \ge d(p,y)\}.$$

The *external diameter* of $\mathscr{P}$, denoted by $\mathscr{D}(\mathscr{P})$, is

$$\mathscr{D}(\mathscr{P}) = \sup\{d(p,q) \mid p,q \in \mathscr{P}\}.$$

A pair of points $p,q \in \mathscr{P}$ with $d(p,q) = \mathscr{D}(\mathscr{P})$ is referred to as a *diametric pair*. A result of Samuel and Toussaint [7] shows that

Lemma 2.1 [7]. *For any given simple polygon $\mathscr{P}$, there exists a diametric pair $p,q \in \mathscr{P}$ such that p (but not necessarily q) is a vertex of $\mathscr{P}$.*

Remarks. (i) It is easy to see that there exist simple polygons in which the external farthest point of every vertex is an interior point of an edge, e.g. consider any regular k-gon with an odd number of vertices (see Fig. 1).

(ii) There are simple polygons in which all diametric pairs of points lie within a pocket (see Fig. 2).

Lemma 2.1 implies that a fast algorithm to compute external farthest neighbors immediately yields an efficient procedure for computing the external diameter.

3. Properties of external shortest paths

In this section we study external shortest paths and prove some of their properties which lead us to an efficient algorithm for finding the farthest neighbors. First we state without proof some simple observations:

- The external distance $d(p,r)$ is a metric. In particular, $d(p,r)+d(r,s)\geq d(p,s)$ for any $p,r,s\in\mathscr{P}$.
- For any two points $p,r\in\mathscr{P}$, $\pi(p,r)$ is a polygonal chain which lies entirely within the region enclosed by $\mathscr{CH}(\mathscr{P})$, and whose vertices occur only at vertices of $\mathscr{P}$.
- An external shortest path between two points of $\mathscr{CH}(\mathscr{P})$ is contained in $\mathscr{CH}(\mathscr{P})$.

Lemma 3.1. *If two points $p,r\in\mathscr{P}$ lie in the same pocket $\overline{\mathscr{K}_i}$, then there is a unique external shortest path between p and r and it lies entirely in $\overline{\mathscr{K}_i}$.*

Proof. Suppose, on the contrary, that some shortest path π between p and r contains a point outside $\overline{\mathscr{K}_i}$. Let c_i (respectively c_f) be the first (respectively last) intersection point of π and the lid $\overline{q_i q_{i+1}}$. We can obtain a shorter external path π' by replacing the portion of π between c_i and c_f with the segment $\overline{c_i c_f}$, contradicting the choice of π. Hence $\pi=\pi(p,r)$ lies entirely in $\overline{\mathscr{K}_i}$. But it is known that the shortest path from p to r lying entirely in $\overline{\mathscr{K}_i}$ is unique (see [5]). Thus, $\pi(p,r)$ is unique. □

The above lemma shows that the problem of finding an external shortest path between two points lying in the same pocket is equivalent to finding the shortest path between two points lying inside a simple polygon. The latter problem has been extensively studied and can be solved quite efficiently [5,4].

Now we focus our attention on external shortest paths between points of different pockets of $\mathscr{P}$. In the remainder of this section we assume that $p\in\mathscr{P}$ (respectively $r\in\mathscr{P}$) is a point lying in the pocket $\mathscr{K}_i$ (respectively $\mathscr{K}_j$) and $i\neq j$.

Lemma 3.2. *The shortest left and right paths from p to r are unique. They are given by*

$$\pi_L(p,r)=\pi(p,q_{i+1})\,\|\,\overline{q_{i+1}q_{i+2}}\,\|\cdots\|\,\overline{q_{j-1}q_j}\,\|\,\pi(q_j,r),\tag{3.1}$$

$$\pi_R(p,r)=\pi(p,q_i)\,\|\,\overline{q_i q_{i-1}}\,\|\cdots\|\,\overline{q_{j+2}q_{j+1}}\,\|\,\pi(q_{j+1},r),\tag{3.2}$$

where $\pi(p,q_{i+1})$ and $\pi(p,q_i)$ (respectively $\pi(q_j,r)$ and $\pi(q_{j+1},r)$) are shortest paths within $\overline{\mathscr{K}_i}$ (respectively $\overline{\mathscr{K}_j}$).

Proof. It is sufficient to show that the path given in the right-hand side of (3.1) is indeed strictly shorter than any other left path from p to r. As no shortest path connecting p and q meets the exterior of the region enclosed by $\mathscr{CH}(\mathscr{P})$, $\pi_L(p,r)$ must at least pass through all of $\{q_{i+1},\ldots,q_j\}$ or all of $\{q_{j+1},\ldots,q_i\}$. However, any external path from p to r that stays inside (or on) $\mathcal{CH}(\mathcal{P})$ but does not pass through some $q_{i'}$, $i+1 \le i' \le j$ is a *right* path. Therefore, $\pi_L(p,r)$ passes through each of $\{q_{i+1},\ldots,q_j\}$. (Being a *shortest* left path, it visits each point exactly once.) A similar argument shows that it must visit $q_{i+1},\ldots,q_j$ in clockwise order, hence $\pi_L(p,r)$ contains the portion of $\mathcal{CH}(\mathcal{P})$ clockwise between q_{i+1} and q_j, as asserted. As for the section of $\pi_L(p,r)$ from p to q_{i+1} (respectively from q_j to r), it is unique and must stay completely in $\overline{\mathcal{K}_i}$ (respectively in $\overline{\mathcal{K}_j}$) by Lemma 3.1. The lemma follows. $\square$

To reiterate, $\pi_L(p,r)$ (respectively $\pi_R(p,r)$) consists of three parts:

(i) Shortest path from p to the lid vertex q_{i+1} (respectively q_i) within $\overline{\mathcal{K}_i}$.
(ii) The clockwise (respectively counter-clockwise) portion of $\mathcal{CH}(\mathcal{P})$ between q_{i+1} and q_j (respectively q_i and q_{j+1}).
(iii) Shortest path inside $\overline{\mathcal{K}_j}$ from the lid vertex q_j (respectively q_{j+1}) to r.

Hence, $\pi(p,r)$ also consists of three parts, as above, because it coincides with either $\pi_L(p,r)$ or $\pi_R(p,r)$.

To compute the external distance between two arbitrary points of $\mathcal{P}$ lying in different pockets, we need a fast procedure for determining the external distance between two points on $\mathcal{CH}(\mathcal{P})$, and a procedure for computing the distance from a lid vertex of a pocket $\mathcal{K}_i$ to an arbitrary query point of $\mathcal{K}_i$. The external distance between two points on $\mathcal{CH}(\mathcal{P})$ can be computed quickly in a straightforward manner by precomputing "partial sums" from, say, q_0. As for computing $d(p,q_i)$ (or $d(p,q_{i+1})$) for $p \in \mathcal{K}_i$, in Lemma 3.1 we have shown that the corresponding path lies entirely inside $\mathcal{K}_i$. Therefore we can use the following result of Guibas et al. [4] to compute the distance from a lid vertex of a pocket to a query point of that pocket.

Given a simple polygon $\mathcal{Q}$ and a vertex v of $\mathcal{Q}$, the *anchor* of a point $x \in \mathcal{Q}$ with respect to v is the last vertex on the (geodesic) shortest path in $\mathcal{Q}$ from v to x, and the *shortest path partition* of $\mathcal{Q}$ with respect to v is the partition of the boundary of $\mathcal{Q}$ into maximal segments such that the anchor for all points in a segment remains the same. Guibas et al. [4] have proved the following theorem.

Theorem 3.3 (Guibas et al. [4]). *For a given triangulated polygon $\mathcal{Q}$ with t vertices and a vertex v, its shortest path partition with respect to v can be obtained in linear time. Once this partition has been obtained, the geodesic shortest distance from v to any point $x \in \mathcal{Q}$ can be computed in time* $O(\log t)$. *Moreover, given the anchor of x, the geodesic shortest distance can be computed in* $O(1)$ *time.*

Thus, to compute the external distance between any two points of $\mathcal{P}$, first compute $\mathcal{CH}(\mathcal{P})$ to obtain the pockets of $\mathcal{P}$; this can be done in $O(n)$ time using the algorithm of [2], for example. Then for each pocket $\overline{\mathcal{K}_i}$, obtain the shortest path partition of $\mathcal{K}_i$ from q_i as well as from q_{i+1}. At this point, the three portions of

$d_L(p,r)$ and of $d_R(p,r)$, and hence $d(p,r)$ can be computed quickly. For a point $p \in \mathscr{K}_s$, let the *left anchor* (respectively *right anchor*) denote the anchor of p with respect to q_s (respectively q_{s+1}); recall that q_s and q_{s+1} are the lid vertices of $\mathscr{K}_s$. The preceding discussion establishes the following lemma:

Lemma 3.4. *After linear-time preprocessing of a triangulated polygon $\mathscr{P}$, the external distance between any two points of $\mathscr{P}$ lying in different pockets can be computed in* $O(\log n)$ *time. Given both left and right anchors for $x, y \in \mathscr{P}$, lying in different pockets, $d(x,y)$ can be computed in* $O(1)$ *time.*

Now we prove a crucial property of external shortest paths which we call *monotonicity property.*

Lemma 3.5 (Monotonicity property). *If $r \in \mathscr{P}(p,s)$, then*

$$d_L(p,s) \le d_R(p,s) \quad \Rightarrow \quad d_L(p,r) \le d_R(p,r). \tag{3.3}$$

Proof. By Lemma 3.1 the above inequality trivially holds if r or s lie in the same pocket as p, therefore we only consider the case when both r and s lie outside the pocket containing p. For a contradiction, suppose $d_L(p,s) \le d_R(p,s) = d_L(s,p)$, but $d_L(p,r) > d_R(p,r) = d_L(r,p)$. Thus

$$d_L(s,p) + d_L(p,r) > d_L(r,p) + d_L(p,s). \tag{3.4}$$

By assumption, p, r, s occur in this clockwise order on $\mathscr{P}$. In particular, $\pi_L(p,s)$ intersects $\pi_L(r,p)$. Let x be a point lying on $\pi_L(p,s) \cap \pi_L(r,p)$ but outside the pocket of p; it is easily seen that such a point always exists. Denoting by π^{-1} the reverse of a (directed) path π, $\pi_L(p,s)$ (respectively $\pi_L(r,p)$) can be written as $\pi_1^{-1} \| \pi_2$ (respectively $\pi_3^{-1} \| \pi_4$) with π_1, π_2, π_3, π_4 emanating from x (see Fig. 3).

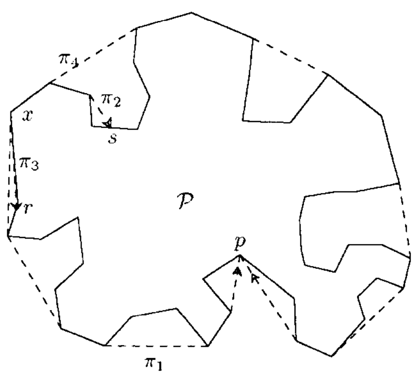

Fig. 3. Illustrating the proof of Lemma 3.5: Four paths emanating from x.

Observe that $\pi_1^{-1} \| \pi_3$ is a left path from p to r, hence $d_L(p,r) \le |\pi_1| + |\pi_3|$. Similarly, $d_L(s,p) \le |\pi_2| + |\pi_4|$, as $\pi_2^{-1} \| \pi_4$ is a left path from s to p. Adding the two inequalities, we obtain

$$d_L(p,r) + d_L(s,p) \le |\pi_1| + |\pi_2| + |\pi_3| + |\pi_4| = d_L(p,s) + d_L(r,p),$$

contradicting (3.4). □

Observe that the inequalities in Lemma 3.5 can also be made "strict" on both sides if desired.

Corollary 3.6. *For a point $p \in \mathscr{P}$, there exists a pair of points $p', p'' \in \mathscr{P}$ (not necessarily distinct) such that*

(i) $\forall x \in \mathscr{P}[p,p'),\ d_L(p,x) < d_R(p,x)$,
(ii) $\forall x \in \mathscr{P}[p',p''],\ d_L(p,x) = d_R(p,x)$,
(iii) $\forall x \in \mathscr{P}(p'',p],\ d_L(p,x) > d_R(p,x)$.

Proof. Let $p \in \mathscr{K}_i$. By Lemma 3.1, $d_R(p,q_{i+1}) > d_L(p,q_{i+1})$ and $d_L(p,q_i) > d_R(p,q_i)$. Since the function $\Delta(x) = d_R(p,x) - d_L(p,x)$ is continuous, the Intermediate Value Theorem guarantees the existence of a point $r \in \mathscr{P}[q_{i+1},q_i]$ such that $\Delta(r) = 0$, i.e., $d_L(p,r) = d_R(p,r)$. Let p' (respectively p'') be the first (respectively last) point on $\mathscr{P}[q_{i+1},q_i]$ for which $\Delta(x) = 0$. Now, Corollary 3.6 follows immediately from the monotonicity property. □

Thus, for every point $p \in \mathscr{P}$, there exists a unique point p'' (namely, the last point of $\mathscr{P}[q_{i+1},q_i]$ for which $\Delta(x) = 0$) such that for all $x \in \mathscr{P}[p,p'']$, we have $\pi(p,x) = \pi_L(p,x)$, and for all $x \in \mathscr{P}(p'',p]$, we have $\pi(p,x) = \pi_R(p,x)$. We call p'' the *bifurcation point* of p, and denote it by $\beta(p)$.

Corollary 3.7. *If $y \in \mathscr{P}[p,\beta(p)]$, then for all $x \in \mathscr{P}[p,y]$, $d_L(x,y) \le d_R(x,y)$.*

Proof. Immediate from Corollary 3.6 and the monotonicity property. □

Corollary 3.8. *Let p, r be two points on an edge of $\mathscr{P}$ such that r lies after p in clockwise direction. Then, p, r, $\beta(p)$, $\beta(r)$ occur in this clockwise order on $\mathscr{P}$ (possibly with $\beta(p) = \beta(r)$).*

Proof. Since p and r lie in the same pocket $\mathscr{K}_i$, $d_L(p,r) < d_R(p,r)$ (and $d_L(r,p) > d_R(r,p)$). Thus $\beta(p)$, $\beta(r) \in \mathscr{P}[r,p]$. Ordering p, r, $\beta(r)$, $\beta(p)$ contradicts the monotonicity property unless $\beta(p) = \beta(r)$. The claim follows. □

4. Computing external farthest neighbors

In this section we describe a two-phase algorithm for computing external farthest neighbors for all vertices of $\mathscr{P}$. First we determine a farthest neighbor of p among the points lying in the same pocket as p, and then we compute a member of $\phi'(p)$. The farther of the two will be an external farthest neighbor of p. At the end of this section we discuss how to modify the algorithm so that it computes *all* external farthest neighbors for every vertex of p.

In Lemma 3.1 we have shown that the external shortest path between two points lying in the same pocket does not leave the pocket; this implies that for every vertex $p \in \mathscr{K}_i$, the problem of computing a farthest neighbor among the points lying in the same pocket as p is equivalent to finding its farthest neighbor inside the simple polygon $\overline{\mathscr{K}_i}$. Suri [9] and Guibas and Hershberger [3] have independently given an $O(t \log t)$ algorithm to compute one internal geodesic farthest neighbor for every vertex in a simple polygon with t vertices. Repeated application of either of these algorithms to each pocket of $\mathscr{P}$ in turn yields the following lemma:

Lemma 4.1. *For every vertex p of $\mathscr{P}$, a farthest neighbor of p among the points lying in its pocket can be computed in time* $O(n \log n)$.

In the remainder of this section we develop an efficient algorithm to compute a representative of $\phi'(p)$ for each vertex $p \in \mathscr{P}$. For a point $p \in \mathscr{K}_i$, let $l(p) = q_{i+1}$ and $r(p) = q_i$. Let $\mathscr{L}(p)$ (respectively $\mathscr{R}(p)$) denote the vertices of $\mathscr{P}$ lying in $\mathscr{P}[l(p), \beta(p)]$ (respectively $\mathscr{P}[\beta(p), r(p)]$) ordered in clockwise direction. Let $\delta_L(p)$ denote the *last* vertex $p' \in \mathscr{L}(p)$ such that

$$d_L(p, p') = \max\{d_L(p, x) \mid x \in \mathscr{L}(p)\}.$$

Similarly, let $\delta_R(p)$ denote the *first* vertex $p'' \in \mathscr{R}(p)$ such that

$$d_R(p, p'') = \max\{d_R(p, x) \mid x \in \mathscr{R}(p)\}.$$

Intuitively, $\delta_L(p)$ (respectively $\delta_R(p)$) represents a vertex of $\mathscr{P}$ farthest from p among the vertices lying in $\mathscr{P}[p, \beta(p)]$ (respectively $\mathscr{P}[\beta(p), p]$).

Lemma 4.2. *For any point $p \in \mathscr{P}$, at least one of $\delta_L(p)$, $\beta(p)$, $\delta_R(p)$ is in $\phi'(p)$.*

Proof. Suppose $x \in \phi'(p) \cap \mathscr{P}[l(p), \beta(p)]$. By definitions of $\phi'(p)$ and $\beta(p)$, we must have

$$d_L(p, x) = \sup\{d_L(p, y) \mid y \in \mathscr{P}[l(p), \beta(p)]\}.$$

But it follows from [6, Lemma 1] that $d(p, y) = d_L(p, y)$ is a convex function of y on any straight-line segment of $\mathscr{P}[l(p), \beta(p)]$, and hence $d(p, y)$ attains its maximum at either of the endpoints. Thus $x \in \mathscr{L}(p) \cup \{\beta(p)\}$, as asserted. The case $x \in \phi'(p) \cap \mathscr{P}[\beta(p), r(p)]$ is similar. $\square$

Observe that the proof of the previous lemma implies that the point(s) of $\{\delta_L(p), \beta(p), \delta_R(p)\}$ farthest from p must lie in $\phi'(p)$. Therefore, a representative of $\phi'(p)$ can be located by computing $\delta_L(p)$, $\beta(p)$, and $\delta_R(p)$.

Assume that $\mathscr{P}$ is given, $\mathscr{CH}(\mathscr{P})$ has been computed, and the preprocessing for external distance queries has been performed. We will first describe how to compute $\beta(p)$ for all vertices $p \in \mathscr{P}$ and then present an algorithm that computes $\delta_L(p)$ for each vertex p. The procedure for computing $\delta_R(p)$ is similar.

4.1. Locating bifurcation points

Let Π_i^l (respectively Π_i^r) denote the set of endpoints of the segments of the shortest-path partition of $\mathscr{K}_i$ from q_i (respectively q_{i+1}). Let $u_1, u_2, \ldots, u_s$ be the points of $\Pi = \bigcup_i (\Pi_i^l \cup \Pi_i^r)$, sorted along $\mathscr{P}$ in the clockwise direction, such that u_1 is a vertex of $\mathscr{CH}(\mathscr{P})$. (Note that the set V of vertices of $\mathscr{P}$ is contained in Π by construction.) Π partitions $\mathscr{P}$ into $O(n)$ segments such that the left and right anchors of a point x remain the same as it varies over a single segment (cf. [4]). If v is the right anchor of x, then $d_R(p,x) = d_R(p,v) + d(v,x)$, where $d_R(p,v)$ is fixed as x ranges over an atomic segment $\overline{u_k u_{k+1}}$, and $(d(v,x))^2$ depends quadratically on the position of x on $\overline{u_k u_{k+1}}$. The function $d_L(x,y)$ has a similar form.

Lemma 4.3. *It is possible to compute Π and determine left and right anchors for each point of Π and for each segment of $\mathscr{P} - \Pi$ in triangulation time.*

Proof. Π_i^l and Π_i^r can be computed in desired time as in Lemma 3.4. Π is determined by merging Π_i^l and Π_i^r for each i and concatenating the resulting sequences. As the left (respectively right) anchor of a point $p \in \Pi_i^r$ (respectively Π_i^l) depends only on the relative position of p among points of Π_i^l (respectively Π_i^r), for each point on Π, both anchors can be easily determined at merge time. Segments of $\mathscr{P} - \Pi$ can be handled similarly. $\square$

For $x, y \in \mathscr{P}$, let $\Delta(x,y) = d_R(x,y) - d_L(x,y)$. By Corollary 3.6 the sign of $\Delta(p,z)$ changes monotonically as z moves along $\mathscr{P}[l(p), r(p)]$, therefore there is a unique $u_i \in \Pi$ such that $\Delta(p, u_i) \geq 0$ and $\Delta(p, u_{i+1}) < 0$. Since $\beta(p)$ is the last point $z \in \mathscr{P}[l(p), r(p)]$ such that $\Delta(p,z) = 0$, $\beta(p) \in \overline{u_i u_{i+1}}$. Moreover, given i, $\beta(p)$ can be determined in constant time, as the precise analytic form of $\Delta(p,z)$ on $\overline{u_i u_{i+1}}$ is known (refer to the discussion above). Thus we can compute bifurcation points for all vertices of $\mathscr{P}$ as follows: We first compute $\beta(p_0)$ by scanning the list Π starting from $u_{i_0} = l(p_0)$ until the first i such that $\Delta(p_0, u_i) < 0$ and then locating $\beta(p_0)$ in the segment $\overline{u_{i-1} u_i}$. Now inductively assume that we have computed $\beta(p_{j-1})$ and it lies in $\overline{u_l u_{l+1}}$. By Corollary 3.8, $\beta(p_j) \in \mathscr{P}[\beta(p_{j-1}), p_{j-1}]$, therefore we scan Π from u_l, find the first i such that $\delta(p_j, u_i) < 0$, and locate $\beta(p_j)$ in $\overline{u_{i-1} u_i}$.

Lemma 4.4. *Given Π together with the anchor information, the above algorithm computes $\beta(p_i)$ for all vertices $p_i \in V$ in linear time.*

Proof. Correctness of the algorithm follows immediately from Corollary 3.8 and from the previous discussion. Observe that each external distance query used in the algorithm is between two points in different pockets and requires O(1) time; this is because the points lie in Π and we have assumed that left and right anchors for all such points have been precomputed. Therefore the running time of the algorithm is proportional to the number of times Δ is evaluated. Corollary 3.8 implies that every evaluation of Δ is associated with advancing either in the list of vertices or in the list Π of candidate points. Since we traverse the list Π at most twice (once to compute $\beta(p_0)$ and at most once to compute $\beta(p_1), \ldots, \beta(p_{n-1})$), Δ is evaluated $O(n)$ times. Hence the total time spent in computing $\beta(p_i)$ for all vertices of $\mathscr{P}$ is $O(n)$. $\square$

4.2. *Computing δ_L and δ_R*

In this section we describe how to compute δ_L for all vertices of $\mathscr{P}$; δ_R is computed similarly. For $p, q \in \mathscr{P}$, let $V[p,q]$ denote the sequence of vertices lying on $P[p,q]$, in clockwise direction.

Recall that $\delta_L(p)$ is the last vertex of $\mathscr{L}(p)$ farthest from p. A brute-force algorithm for computing $\delta_L(p_i)$ for a vertex p_i of $\mathscr{P}$ would determine the external distance from p_i to every point in $\mathscr{L}(p_i)$, and would choose the last point farthest from p_i. However, the worst-case time complexity of computing $\delta_L(p_i)$ for all vertices of $\mathscr{P}$ using this approach is quadratic. Consequently, we now describe the intuition behind the changes that will transform this straightforward procedure into an efficient algorithm. We regard $\mathscr{L}(p_i)$ as a queue of points ordered along $\mathscr{P}$, with its front at $l(p_i)$ and rear towards $\beta(p_i)$. Observe that, as one advances from p_i to p_{i+1}, $\mathscr{L}(p_{i+1})$ can be obtained from $\mathscr{L}(p_i)$ by deleting $V[l(p_i), l(p_{i+1}))$ from its front and appending $V(\beta(p_i), \beta(p_{i+1})]$ to the rear. (The former step is nonvacuous only if p_{i+1} is a lid vertex because otherwise $l(p_i) = l(p_{i+1})$.) However, this modification alone does not improve the performance of the algorithm, since for each p_i it still appears to be necessary to examine every point in $\mathscr{L}(p_i)$ when looking for a maximum of the function $d_L(p_i, \cdot)$. The next improvement follows from the observation that $d_L(p_i, x) > d_L(p_i, y)$ if and only if $d_L(p_{i+1}, x) > d_L(p_{i+1}, y)$, for $x, y \in V[l(p_{i+1}), \beta(p_i)]$, as shown in Lemma 4.5 below. This allows us to maintain $\mathscr{L}(p_i)$ as a *max_queue* $\mathscr{Q}$; max_queue is a data structure that, in addition to the standard constant-time queue operations *delete_front* ($\mathscr{Q}$) and *insert_rear* ($\mathscr{Q}$, *value*), supports a constant-time *find_max* ($\mathscr{Q}$) operation which returns the maximum value found in $\mathscr{Q}$, without otherwise disturbing its state. In what follows we will assume that find_max locates the *last* such element in $\mathscr{Q}$; for possible implementations of max_queues, see [1, 8].

More formally, let $V[q_i, p_j]$ be the sequence of consecutive vertices of $\mathscr{P}$, start-

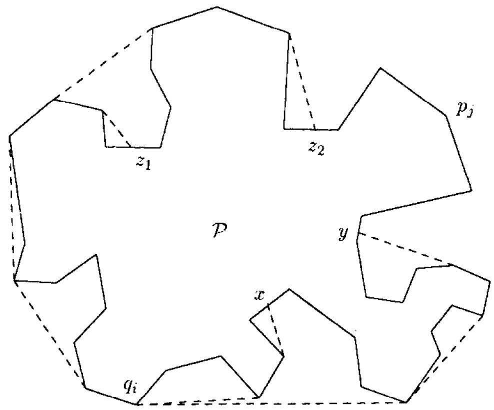

Fig. 4. Illustration of Lemma 4.5.

ing at a vertex q_i lying on $\mathscr{CH}(\mathscr{P})$, and ending at $p_j \neq q_i$. For $z_1, z_2 \in V[q_i, p_j]$, we say that z_2 is *larger* than z_1 if $d_L(x, z_2) > d_L(x, z_1)$, where x is an arbitrary point of $\mathscr{P}(p_j, q_i]$. This notion is independent of the choice of $x \in \mathscr{P}[p_j, q_i]$, as the following lemma shows (taking y to be an alternate choice of x):

Lemma 4.5. *Let $z_1, z_2 \in \mathscr{P}[q_i, p_j]$ and $x, y \in \mathscr{P}(p_j, q_i]$, with p_j and q_i as above (see Fig. 4). Then*

$$d_L(x, z_2) > d_L(x, z_1) \quad \Leftrightarrow \quad d_L(y, z_2) > d_L(y, z_1).$$

Proof. Since $q_i \in \mathscr{P}[x, z_2] \cap \mathscr{CH}(\mathscr{P})$, q_i lies on $\pi_L(x, z_2)$, and so $d_L(x, z_2) = d_L(x, q_i) + d_L(q_i, z_2)$. Similarly, $d_L(y, z_2) = d_L(y, q_i) + d_L(q_i, z_2)$, $d_L(x, z_1) = d_L(x, q_i) + d_L(q_i, z_1)$, and $d_L(y, z_1) = d_L(y, q_i) + d_L(q_i, z_1)$. Therefore

$$\begin{aligned} d_L(x, z_2) > d_L(x, z_1) \quad &\Leftrightarrow \quad d_L(x, q_i) + d_L(q_i, z_2) > d_L(x, q_i) + d_L(q_i, z_1) \\ &\Leftrightarrow \quad d_L(y, q_i) + d_L(q_i, z_2) > d_L(y, q_i) + d_L(q_i, z_1) \\ &\Leftrightarrow \quad d_L(y, z_2) > d_L(y, z_1). \qquad \square \end{aligned}$$

Hence $\delta_L(p_i)$ can be determined by maintaining $\mathscr{L}(p_i)$ as a max_queue. Recall that p_0 is assumed to be a vertex of $\mathscr{CH}(\mathscr{P})$. $\mathscr{L}(p_0)$ is built from an empty queue by a clockwise scan of $\mathscr{P}$. The second scan computes $\mathscr{L}(p_i)$ and determines $\delta_L(p_i)$ as its largest element, for all successive values of i. A more formal description of the algorithm is given in Fig. 5. Correctness of the algorithm follows from the above discussion. As for the running time, we have:

Lemma 4.6. *The time required to compute $\delta_L(p_i)$, for all vertices $p_i \in \mathscr{P}$, after preprocessing $\mathscr{P}$ for external distance queries, is at most* $O(n)$.

Input: $\mathcal{P}$ *preprocessed for external distance queries, and* $\beta(p_i)$ *for all* $p_i \in V$.
Output: $\delta_L(p_i)$ *for each* $p_i \in V$.

```
Q := (p_0);  χ := p_0
for j = 0 ... n − 1 do
    while front(Q) ≠ l(p_j) do
        delete_front(Q)
    end while
    for p_k ∈ V(χ, β(p_j)] do
        insert_rear(Q, p_k)
    end for
    δ_L(p_j) := find_max(Q);  χ = β(p_j)
end for
```

Fig. 5. Computing $\delta_L(p_i)$ for vertices of $\mathscr{P}$.

Proof. By Lemma 4.4, the bifurcation points of all vertices of $\mathscr{P}$ can be calculated in O(n) time. As for updating the queue $\mathscr{Q}$, V is scanned at most twice (once for computing $\delta_L(p_0)$, and once for computing δ_L for the remaining vertices), which implies that only O(n) points are inserted into $\mathscr{Q}$. Since each element is deleted from the queue only once, only O(n) queue insertions, deletions, and find_max queries are performed; each requires O(1) time. Maintenance of the max_queue involves comparing the values of its elements, i.e., making external distance queries. However, only a constant number of such queries is made per max_queue operation and each query takes O(1) time. Hence all δ_L can be computed in O(n) time after the initial preprocessing. □

Using the same method we can compute δ_R for all vertices in O(n) time. Therefore, we conclude that

Theorem 4.7. *One point of* $\phi'(p_i)$ *for each vertex* $p_i \in V$ *can be computed in triangulation time.*

Remark. The above algorithm computes only a representative of $\phi'(p_i)$. However, it can be easily extended so that it returns all elements of $\phi'(p_i)$ in time O($n+k'$), where $k' = \sum_{p_i \in V} |\phi'(p_i)|$. First we run the above algorithm to determine which of $\{\delta_L(p_i), \beta(p_i), \delta_R(p_i)\}$ is farthest from p_i. If $\delta_L(p_i)$ (respectively $\delta_R(p_i)$) is farthest, then we compute all vertices in $\mathscr{P}[l(p_i), \beta(p_i)]$ (respectively $\mathscr{P}[\beta(p_i), r(p_i)]$) which are farthest from p_i by running a slightly modified version of our algorithm. In this variation of the algorithm, the implementation of the max_queue data structure is altered to allow the access to the list of *all* largest elements contained

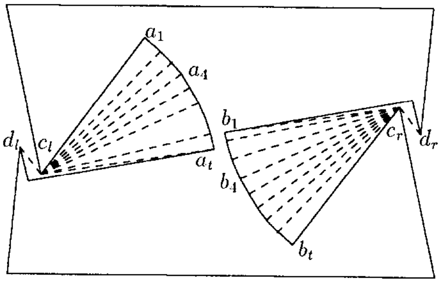

Fig. 6. A simple polygon $\mathscr{P}$ with $k'=\Omega(n^2)$.

in $\mathscr{Q}$, so that the set of all vertices of $\mathscr{L}(p_i)$ farthest from p_i can be extracted from it (if necessary) in time proportional to the number of such points.

It can be easily shown that in the worst case $k'=\Omega(n^2)$. For the lower bound, consider the simple polygon $\mathscr{P}$ shown in Fig. 6. The points $a_1, \ldots, a_t$, for $t=\frac{1}{2}n-5$ lie on a circular arc with center c_l and radius ϱ, and the points $b_1, \ldots, b_t$ lie on another circular arc with center c_r and radius ϱ. If we choose d_l and d_r so that $\overline{c_l d_l}=\overline{c_r d_r}$, $d_L(c_l, d_r)=d_R(d_l, c_r)$, and d_l (respectively d_r) is not visible from a_t (respectively b_1), then it is easy to see that for any $i \le t$, $\phi(a_i)=\{b_1, \ldots, b_t\}$ (respectively $\phi(b_i)=\{a_1, \ldots, a_t\}$). Hence $\sum_{p_i \in \mathscr{P}} |\phi'(p_i)|=\Omega(n^2)$.

Corollary 4.8. *The total time required to compute one external farthest neighbor $\phi(p_i)$ for each vertex $p_i \in \mathscr{P}$ is at most* $O(n \log n)$.

Remark. Note that the only part of the algorithm that required $O(n \log n)$ time is the computation of a farthest neighbor of p_i among the points lying in the same pocket as p_i, for each $p_i \in V$, as the preprocessing of our algorithm requires only triangulation plus linear time and after preprocessing a representative of $\phi'(p_i)$ for each $p_i \in \mathscr{P}$ can be computed in $O(n)$ time. Therefore, a more efficient algorithm to compute the geodesic farthest neighbors would immediately yield a faster algorithm for the external farthest neighbors. Moreover, we can modify the algorithm as discussed above to yield, together with a variant of the algorithm by Suri [9] or that given by Guibas and Hershberger [3], an $O(k+n \log n)$ algorithm for obtaining *all* external farthest neighbors for each vertex $p \in \mathscr{P}$, where $k=\sum_{p \in V} |\phi(p)|$.

As for computing the external diameter of $\mathscr{P}$, it follows from Lemma 2.1 that

Theorem 4.9. *The external diameter of a simple n-gon can be computed in time* $O(n \log n)$ *using* $O(n)$ *space.*

5. Conclusions

In this paper, we provided an $O(n \log n)$ algorithm for finding an external geodesic farthest neighbor for every vertex of a given simple n-gon. We also showed that this algorithm can be modified to output all external geodesic farthest-neighbors for all the vertices of the polygon, and that it can be used to compute the external diameter of the polygon in $O(n \log n)$ time. For finding the external diameter, the time complexity of our algorithm improves upon that given by Samuel and Toussaint [7] by an $O(n/\log n)$ factor. However, the best known lower bound for computing the external geodesic diameter is the trivial $O(n)$, and obtaining an optimal bound remains open.

A careful analysis of our algorithm shows that if there is an algorithm that computes the internal geodesic farthest neighbors of the vertices of a simple n-gon in $\tau(n)$ time, then our algorithm can use it as a subroutine to compute one external farthest neighbor of every vertex of a triangulated simple n-gon in time $O(\tau(n)+n) = O(\tau(n))$. Consequently, an improvement in the time complexity of finding the internal geodesic farthest neighbors will yield the corresponding improvement in the running time of finding the external farthest neighbors.

References

[1] H. Gajewska and R.E. Tarjan, Deques with heap order, Inform. Process. Lett. 22 (1986) 197–200.

[2] R.L. Graham and F.F. Yao, Finding the convex hull of a simple polygon, J. Algorithms 4 (1983) 324–331.

[3] L. Guibas and J. Hershberger, Optimal shortest path queries in a simple polygon, J. Comput. Systems Sci. 39 (1989) 126–152.

[4] L. Guibas, J. Hershberger, D. Leven, M. Sharir and R. Tarjan, Linear time algorithms for shortest path and visibility problems, Algorithmica 2 (1987) 209–233.

[5] D.T. Lee and F.P. Preparata, Euclidean shortest paths in presence of rectilinear barriers, Networks 14 (1984) 393–410.

[6] R. Pollack, M. Sharir and G. Rote, Computing the geodesic center of a polygon, Discrete Comput. Geom. 4 (1989) 611–626.

[7] D. Samuel and G. Toussaint, Computing the external geodesic diameter of a simple polygon, Computing 44 (1990) 1–19.

[8] R. Sundar, Worst case data structures for the priority queue with attrition, Inform. Process. Lett. 31 (1989) 69–75.

[9] S. Suri, The all-geodesic-furthest neighbors problem for simple polygons, J. Comput. Systems Sci. 39 (1989) 220–235.