International Journal of Computational Geometry & Applications

AN $O(n \log n)$ ALGORITHM FOR FINDING A SHORTEST CENTRAL LINK SEGMENT *

LYUDMIL G. ALEKSANDROV
Bulgarian Academy of Sciences, CICT
Acad. G. Bonchev Str., Bl. 25-A, 1113 Sofia, Bulgaria

HRISTO N. DJIDJEV
Department of Computer Science, University of Warwick
Coventry, CV4 7AL, England

and

JÖRG-RÜDIGER SACK
School of Computer Science, Carleton University
Ottawa, Ontario K1S 5B6, Canada

Received received date
Revised revised date
Communicated by Editor's name

ABSTRACT

A central link segment of a simple n-vertex polygon P is a segment s inside P that minimizes the quantity $\max_{x \in P} \min_{y \in s} d_L(x, y)$, where $d_L(x, y)$ is the link distance between points x and y of P. In this paper we present an $O(n \log n)$ algorithm for finding a central link segment of P. This generalizes previous results for finding an edge or a segment of P from which P is visible. Moreover, in the same time bound, our algorithm finds a central link segment of minimum length. Constructing a central link segment has applications to the problems of finding an optimal robot placement in a simply connected polygonal region and determining the minimum value k for which a given polygon is k-visible from some segment.

Keywords: Algorithm design and analysis, Computational geometry, Link distance, Simple polygon, Shortest segment

1. Introduction

1.1. Motivation and Applications

The link distance, defined with respect to a planar region P, sets the distance between a pair of points (x, y) in P to be the minimum number of line segments

*Research supported in part by the Natural Sciences and Engineering Research Council of Canada and the National Science Foundation.

in any path in P that connects x to y. A path connecting x and y whose length is equal to the link distance between x and y is called a *minimum link path*. See Fig. 1 below.

The study of link distance problems is motivated by applications in several areas in computer science, such as Computer Graphics, Geographical Information Systems, Robotics, Image Processing, Cartography, VLSI, Computer Vision, and Solid Modeling. Frequently in these application areas it is important to produce paths of low combinatorial complexity. In the following we list some applications: the details can be found in [3,6,10,15,20,21,22,24,26,27]. For a survey on link distance problems and their solutions we refer the reader to the Handbook of Computational Geometry [7]).

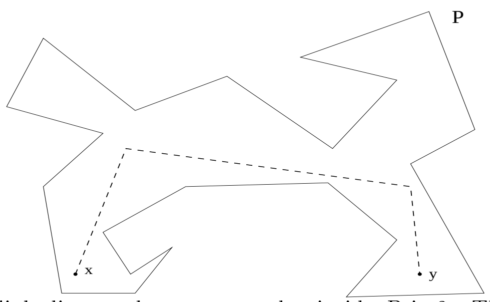

Fig. 1. The link distance between x and y inside P is 3. The dashed line segments illustrate a link path between x and y.

1. *Motion Planning:* Consider a robot that can move forward and rotate, and that is given the task to navigate collision-free in a constrained planar region. Suppose that straight line motion is “free”, but rotation is “expensive”. One motion planning task is to move the robot collision-free between two specified positions inside the region, such that the total number of turns is minimized. For point robots, the task corresponds to computing a minimum link path. In the *optimal robot placement problem* the goal is to place a robot inside P so that the maximum number of rotations needed for the robot to reach any point in P is minimum.

2. *Communication Systems Design:* In the design of communication systems using microwaves or lights as a communication media, direct communication is possible only if the transmitter and the receiver are in direct line-of-sight, a condition rarely met. Therefore, special devices, such as repeaters, are installed. Finding a minimum link path between two points inside a polygonal region means minimizing the number of repeaters required for the communication inside that region. (For details, see [24].)

3. *Placement of telescoping manipulators:* Suppose that an environment for a manipulator with telescoping links is described by a polygonal region. A telescoping link is a link of flexible size. The task here is to determine the minimum number of telescoping links that will allow the manipulator to reach every point in the environment. (For details, see [15].)

4. *Curve compression:* Minimum link paths are also used for curve compression in solid modeling. A typical problem in solid modeling is to compute the intersection of surfaces. The intersection of two surfaces can be quite complex even if the surfaces themselves are of relative low degree. To avoid the combinatorial explosion inherent in the closed form solution, it is convenient to compute a piecewise linear approximation to the curve of intersections. Natarajan [22] proposes to replace an intersection curve Γ by its minimum link analog, which is computed inside an ϵ-fattening of Γ. Related work on approximating simple polygons and polygon maps is discussed in [10].

5. *VLSI Layouts:* In graph layout and VLSI layout problems, it is desirable to minimize the number of bends. Moreover, in VLSI layout problems, wires typically run along orthogonal axes, and this motivates the study of link path problems in rectilinear settings. (For details, see [27].)

An interesting aspect of the study of link distance problems is that often algorithms are more complex and/or have higher time complexities than those solving the corresponding problems for the Euclidean distance. One of the reasons for this is that typically the Euclidean path connecting two points is unique whereas this is usually not the case for a minimum link path. In our consideration and time complexity evaluations we assume the algebraic computational model [23]. When bit complexity and robustness issues are taken into account link distance problems become harder as shown in [16].

1.2. Central Link Segment

The problem we solve in this paper is related to both the notion of visibility of a polygon from a segment and to optimal robot placement.

Recall that a polygon P is called *visible* (or *weakly visible*) from a line segment s inside P if each point in P can be seen from some point on s. Avis and Toussaint [2] gave a linear-time algorithm for deciding whether a simple polygon P is visible from a given edge of P. Sack and Suri [25] presented a linear-time algorithm for finding the set of all polygonal edges from which the input polygon is visible. Ke [14] subsequently extended their results to finding diagonals from which P is visible. (A *diagonal* is a segment inside P joining two vertices of P.) Doh and Chwa [8] reported an $O(n \log n)$ algorithm that determines whether a given polygon of size n is visible from any inner segment and if yes the algorithm finds such a segment. A linear-time algorithm for finding the shortest such line segment was claimed in [5]. Generalizing the notion of segment visibility, a polygon P is defined to be k-*visible* from some segment s in P if the link distance from each point x in P to some point on s (depending on x) is at most k. This implies that if P is k-visible from s, then the number of mirrors required to illuminate any point in P from a (segment) light source s is at most $k-1$. The k-*visibility problem* is to find the minimum value k^* for which a given polygon P is k^*-visible from some segment in P and to determine such a segment s of minimum length.

This problem can be formulated as a link distance problem as follows. The *link eccentricity* of a given segment s in P is defined as

$$c(s,P) = \max_{x \in P} \min_{y \in s} d_L(x,y),$$

where $d_L(x,y)$ is the link distance between x and y. A *central link segment* of P is defined as a segment in P with minimum link eccentricity among all segments in P. The *covering radius* r_c of P is the eccentricity of any central link segment. Therefore, an algorithm for finding a shortest central link segment and the covering radius of a polygon solves the k-visibility problem.

Consider now the optimal robot placement problem as defined above. A natural approach for solving that problem is to locate the robot anywhere inside the link center of the polygon, where the *link center* of P is the set of all points x in P such that the maximum link distance between x and any other point of P is a minimum. An $O(n^2)$ algorithm for computing the link center of a simple n-vertex polygon has been described in [18]; the time complexity was reduced to $O(n \log n)$ by [6] and independently by [14].

However, as illustrated in Fig. 2(a), such an approach may result in a suboptimal solution for the robot placement problem. By placing the robot at point a (an endpoint of a central link segment s) with initial orientation in the direction of the other endpoint b of s, any point of P can be reached after at most two rotations. Placing the robot at x (still inside the link center) results in a suboptimal location; no initial robot orientation is possible so that both v and w can be reached after at most two rotations. In general, the optimal robot placement is at one of the endpoints of a central link segment with initial orientation in the direction of the other endpoint. Fig. 2(b) illustrates a shortest central link segment s^* for P and provides a solution to the robot placement problem. The segment is also a solution to the k-visibility problem, where the covering radius of P equals two.

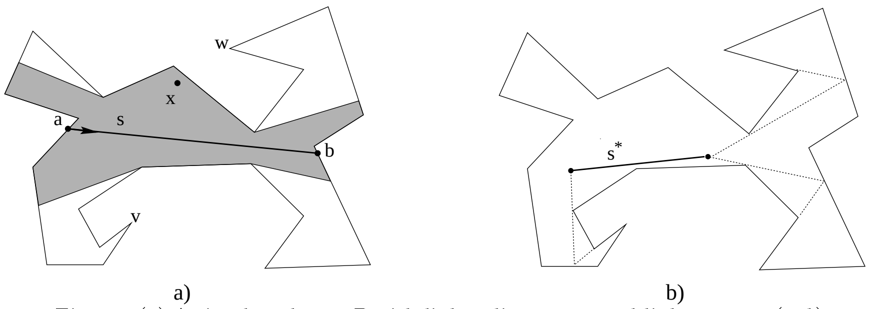

Fig. 2. (a) A simple polygon P with link radius 3 a central link segment (a,b) and shaded link center. By placing a robot at a oriented toward b any point in P can be reached in at most two rotations. (b) A shortest central link segment s^* of P. The eccentricity of s^* and the covering radius of P equal two.

1.3. Overview of Algorithm

In this paper we present an $O(n \log n)$ algorithm for finding a central link segment of minimum length (a *shortest central link segment*) of a simple n-vertex polygon P. This provides $O(n \log n)$ solutions to the k-visibility problem and the optimal robot placement problem. In our preliminary paper [1] we described an algorithm that finds a central link segment, but not necessarily of minimum length. In [14] Ke claims the same time bound established here; no proof is given. His claim is based on his work described in [13]. The key reduction of [13], i.e. replacement of a set of segments by a simpler such set, which would make the problem significantly easier, turns out to be incorrect (a counter example can be obtained from the authors upon request).

Chords are segments interior to P whose two endpoints lie on the boundary of P. Diagonals are chords whose endpoints are vertices of P. Any chord s separates P into two subpolygons, which we refer to as *half-polygons* determined by s.

We first reduce the problem of finding a shortest central link segment to a problem of finding a shortest segment in P that intersects a collection of "half-polygons".

We apply the link center algorithm [6] and compute a set S of $O(n)$ "relevant" chords in $O(n \log n)$ time. To each of the chords s in S one of the two half-polygons defined by s is assigned. Let Π denote the set of those half-polygons. We show that a segment is a central link segment if and only if it intersects all half-polygons of Π.

Next, using the set Π, we define and compute a set of $O(n)$ pairs of convex polygonal chains inside P and show that a shortest central link segment must be a segment joining one of these pairs. Thereby for each pair of convex chains we find a shortest segment in P joining these chains. A shortest segment over all pairs is a shortest central link segments of P and will provide a solution to our problem.

The algorithm for finding shortest segments for all pairs of chains is technically fairly complicated, since, although the union of all chains is of size $O(n)$ only, they may overlap and in result any of the $\Theta(n)$ chains may have size $\Theta(n)$. We apply a careful sweep technique based on the order of chains and combine it with an algorithm of Kirkpatrick and Snoeyink [17] which, finds a shortest segment in an X-shaped polygon. More precisely, given an X-shaped polygon, i.e. a polygon whose boundary consists of four outward convex chains X_1, X_2, X_3, X_4 listed in clockwise order, the algorithm finds a shortest segment inside X with endpoints on X_1 and X_3 in $O(\log n)$ time (assuming the representation of X_1, X_2, X_3, and X_4 allows binary search).

The paper is organized as follows. In Section 2, we give additional definitions, present some preliminary results and reduce the problem of finding a central link segment to the half-polygon intersection problem. In Section 3, we study some properties of central link segments and identify a collection of pairs of regions inside P that contain endpoints of central link segments. In Section 4, we describe the structure of the entire algorithm. Two key steps require algorithmic development and a detailed explanation; these are given in Sections 5 and 6. In Section 5, we study the convex boundaries of the pairs of regions identified in Section 3. We efficiently compute these pairs of boundary chains and store them appropriately in order to enable efficient finding of the shortest segments that join those pairs. In Section 6, shortest segments inside P that join pairs of chains from Section 5 are found. Finally, in Section 7, we consider a special case where one of our assumptions is not satisfied and show how our algorithm handles that case.

2. Preliminaries and Reduction to an Intersection Problem

In this section we recall some terminology and results from the literature as necessary for this paper. Let P be an n-vertex simple polygon. A vertex v of P is called *reflex* if the inner angle of P at v is larger than $180°$, otherwise, v is called *convex*. The segments of the boundary of P are called *edges* of P. Throughout the paper we deal with segments that lie inside P; amongst them we distinguish between *chords* and *diagonals*. Chords are segments interior to P whose two endpoints lie on the boundary of P. Diagonals are chords whose endpoints are vertices of P. Any chord s separates P into two subpolygons, which we refer to as *half-polygons* determined by s.

Recall that the link center of P consists of all points in P whose maximum link distance to any other point of P is minimized. The *link radius* r of P is the maximum link distance between a point of the link center and any other point of P. The link radius of P can be found in $O(n \log n)$ time using the above mentioned algorithms [6,14] for finding the link center. The following proposition follows from the definitions of covering radius and link radius.

Proposition 2.1 *The covering radius of a polygon with link radius r equals either $r-1$ or r.*

Corollary 2.2 *If the covering radius r_c of P equals r, then any segment intersecting the link center is a central link segment and any point inside the link center is a shortest central link segment.*

In our algorithm we first use the algorithm from [6] and find the link radius r of the input polygon as well as some additional information, as described below. Next, our algorithm determines whether there exist segments with eccentricity $r-1$. If yes, then we have $r_c = r-1$, and the algorithm proceeds by finding a shortest central link segment. If not, then $r_c = r$ and by Corollary 2.2 it follows that finding a shortest central link segment reduces to finding a point in the link center and can be done by running the algorithm from [6]. Except for the determination problem, that is whether r_c equals r or not, the interesting case arises when r_c equals $r-1$.

For our solution we will make use of several results from the related literature discussed next. A *k-visibility region* $N_k(x)$ of a point $x \in P$ is defined by

$$N_k(x) = \{y \in P : d_L(x,y) \leq k\}.$$

Informally, $N_k(x)$ is the set of points of P that are k-visible from x. The notion of k-visibility regions was introduced in [18] and the following theorem concerning their structure has been proved.

Theorem 2.3 [18] *Let $k \geq 1$ and let x be a point of P. The visibility region $N_k(x)$ is a subpolygon of P all of whose vertices lie on the boundary of P and each of whose edges is one of the following types:*

(i) *a (portion of an) edge of P;*

(ii) *a chord connecting a reflex vertex of P to a point on the boundary of P.*

Theorem 2.3 and the structure of visibility regions are illustrated in Fig. 3.

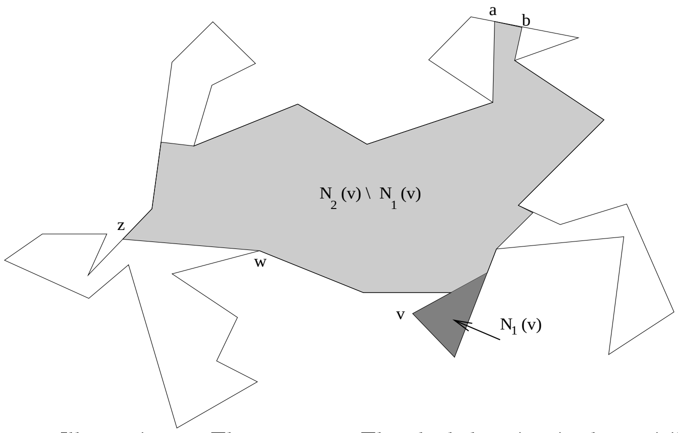

Fig. 3. Illustration to Theorem 2.3. The shaded region is the 2-visibility region $N_2(v)$ of the vertex v. The edge (a, b) is of type (i), whereas (w, z) is of type (ii). Regions $N_1(v)$ and $N_2(v) \setminus N_1(v)$ are shaded differently.

The visibility regions are relevant in our context as stated in Proposition 2.4. The proposition follows easily by using a straightforward generalization of the argument given in [2] that an edge sees all vertices of a polygon iff it sees the entire polygon.

Proposition 2.4 *Let P be a polygon with link radius r. A segment s^* is a central link segment of P with link eccentricity $r-1$ iff s^* intersects $N_{r-1}(v)$ for any vertex v of P.*

Proposition 2.4 suggests a straightforward approach for finding a central link segment. We may construct the regions $N_{r-1}(v)$, for all vertices of P, and then find a segment s^* that intersects all these regions. But, as it can easily be observed, the total number of vertices on $N_k(v)$ for all v and $k \leq r-1$ may be quadratic in n and therefore this straightforward approach cannot lead to an $O(n \log n)$ algorithm for finding a central link segment. Instead, we adopt the approach from [6], where the algorithm computes only parts of the boundaries of $N_k(v)$ regions for $v \in P$ and $k = 1, \ldots, r-1$.

We sketch the approach from [6]. A *central diagonal* d is a diagonal of P that has eccentricities at least r in both half-polygons determined by d, i.e., if $c(d, P_L) \geq r$ and $c(d, P_R) \geq r$, where P_L and P_R are the two half-polygons of P determined by d (Fig. 4). A *central triangle* of P is a triangle $\triangle$ inside P whose edges t_1, t_2, t_3 are diagonals of P such that $c(t_i, P_i) \leq r-1$, where P_i is the half-polygon determined by t_i that does not contain $\triangle$, for $i = 1, 2, 3$.

Theorem 2.5 [6] *Let P be a simple n-vertex polygon with link radius r. In $O(n \log n)$ time, either*

(i) *a central diagonal d of P is found, or*

(ii) *it is detected that P does not possess such a diagonal, in which case a central triangle is found.*

The central diagonal or central triangle allows us to direct our search for a possible location of a central link segment. We shall prove that it is sufficient to find only those of the edges of the regions $N_{r-1}(v)$ that are visible from d or $\triangle$.

Let Q be the visibility region of the central diagonal d or the central triangle $\triangle$, as the case may be, i.e., the set of points which are seen from some point on the diagonal (triangle, respectively). Let E_k, for $0 < k \leq r-1$, be the set of chords that are edges of the regions $N_k(v)$, for vertices v of P. By definition any chord $s \in E_k$ is an edge of $N_k(v)$ for some vertex v. Let $P(s,v)$ (or simply $P(s)$) denote the half-polygon of P determined by s and containing $N_k(v)$. Clearly, a segment intersects a region $N_k(v)$ iff it intersects all half-polygons determined by the edges of $N_k(v)$. Thus, by Proposition 2.4, a segment s^* is a central link segment with eccentricity $r-1$ iff s^* intersects all half-polygons $P(s)$ for $s \in E_{r-1}$. Using the region Q, we are able to define and compute a subset of E_{r-1} that still characterizes segments with eccentricity $r-1$.

Let $E_{r-1}(Q)$ be the subset of chords in E_{r-1} that intersect Q. A subset $S \subset E_{r-1}(Q)$ is called an *essential subset* for $E_{r-1}(Q)$, if for any chord s' in $E_{r-1}(Q)$ there exists a chord $s \in S$ so that $P(s) \subset P(s')$. Note that if a segment intersects all half-polygons determined by an essential subset, then it intersects all half-polygons determined by $E_{r-1}(Q)$. An essential subset is called *minimal* if it does not contain smaller essential subset.

The next lemma shows that, knowing an essential subset of $E_{r-1}(Q)$, the computation of a central link segment can be reduced to finding a segment intersecting the half-polygons determined by that set.

Lemma 2.6 *Let P be a polygon with covering radius $r_c = r-1$. Let S be an essential subset of chords for $E_{r-1}(Q)$. A segment s^* in P is a central link segment of P iff s^* intersects all half-polygons $P(s)$ for all $s \in S$.*

Proof. Proving the "$\Rightarrow$" direction is straightforward: namely, by Proposition 2.4, s^* intersects all half-polygons $P(s)$ for $s \in E_{r-1}$ and therefore it intersects all $P(s)$ for $s \in S$, since $S \subset E_{r-1}$.

Consider now the "$\Leftarrow$" direction. Let s^* intersect all half-polygons $P(s)$ for $s \in S$. The set S is an essential subset and thus s^* intersects all $P(s)$ for $s \in E_{r-1}(Q)$. We shall prove that s^* has eccentricity $r-1$. We also need to argue that $E_{r-1}(Q)$ is not empty.

By Proposition 2.4, a segment has eccentricity $r-1$ if it intersects all half-polygons $P(s)$ for $s \in E_{r-1}$ and therefore we need to show that s^* intersects the half-polygons $P(s)$ for those chords in E_{r-1} that do not intersect Q. To prove this, we show that s^* intersects Q and that $Q \subset P(s)$ for any $s \in E_{r-1} \setminus E_{r-1}(Q)$.

Consider first the case where the polygon P has a central diagonal d. By definition, d has eccentricity at least r in each of the two half-polygons P_L and P_R determined by d. Therefore, there must be two chords $s_L \in E_{r-1}$ and $s_R \in E_{r-1}$ such that $P(s_L) \subset P_L$ and $P(s_R) \subset P_R$. Our polygon P has a covering radius $r-1$ and therefore (by Proposition 2.4) there is a segment that intersects all half-polygons determined by chords in E_{r-1} and, in particular, it intersects both $P(s_L)$ and $P(s_R)$. Since these two polygons lie on opposite sides of d, it follows that a central link segment of P must intersect s_L, s_R and d. Thus s_L and s_R are visible from d, or otherwise they intersect Q and belong to $E_{r-1}(Q)$. We proved that $E_{r-1}(Q)$ is not empty. Note that, according to the definition of an essential subset of $E_{r-1}(Q)$, there are two segments with the same property in S, and w.l.o.g. we may assume that s_L and s_R are in S.

The segment s^* intersects $P(s_L)$ and $P(s_R)$ and thus it must intersect d. Therefore s^* intersects Q (in fact, in this case s^* lies entirely in Q).

Next we show that if a chord in E_{r-1} does not intersect Q, then its corresponding half-polygon contains Q. Let s be any chord in $E_{r-1} \setminus E_{r-1}(Q)$. Since s does not intersect d, s lies entirely either in P_L or in P_R – assume that $s \in P_L$. The chord s does not intersect Q and therefore its half-polygon $P(s)$ either contains Q, or does not intersect Q at all. The latter is impossible since in that case no segment can intersect both $P(s)$ and $P(s_R)$ and thus no segment would have eccentricity $r-1$. We proved that $Q \subset P(s)$ for each $s \in E_{r-1} \setminus E_{r-1}(Q)$, which establishes the correctness of the proposition for the case where a central diagonal exists. Besides, we have made an observation that we shall use in our further discussion; we formulate it as a Corollary 2.7 after the proof.

Consider now the case where P has no central diagonal. According to Theorem 2.5, in this case, a central triangle Δ exists. By the definition of a central triangle, the edges t_i of Δ, for $i = 1, 2, 3$, that are diagonals of P, have eccentricities in their corresponding half-polygons P_i not exceeding $r-1$. Therefore, t_i is intersected by $P(s, v)$ for each $s \in E_{r-1}$ such that $v \in P_i$. This implies that $Q \subset P(s)$ for each chord $s \in E_{r-1}$ that does not intersect Q.

It remains to prove that s^* intersects Q. The intersection of all half-polygons $P(s)$ for $s \in E_{r-1}$ is empty, since otherwise the radius of P would have been less than r. By Helly's theorem, there exists a triple of chords s_1, s_2, and s_3 in E_{r-1} so that $P(s_1) \cap P(s_2) \cap P(s_3)$ is empty. Since for each $s \in E_{r-1}$ either s intersects $\triangle$ or $\triangle \subset P(s)$, then at least two of the chords s_1, s_2, and s_3 must intersect $\triangle$ and therefore must lie entirely in Q. Moreover, these two chords must intersect the third one since otherwise the intersection of the half-polygons $P(s_1)$, $P(s_2)$, and $P(s_3)$ could not be empty. This means that $s_i \in E_{r-1}(Q)$, for $i = 1, 2, 3$, and thus $E_{r-1}(Q)$ is not empty. Finally, the segment s^* intersects $P(s_i)$, for $i = 1, 2, 3$, and since $\cap_{i=1}^{3} P(s_i) = \emptyset$, then s^* must intersect at least two of the segments s_i. Two of these three segments lie entirely in Q and thus s^* intersects Q. $\square$

Corollary 2.7 *Let P be a polygon with a central diagonal d and covering link radius $r-1$. Then any essential subset S of $E_{r-1}(Q)$ contains a pair of chords s_L and s_R so that $P(s_L) \subset P_L$ and $P(s_R) \subset P_R$.*

By using the algorithm from [6] we are able to compute a minimal essential subset S of $E_{r-1}(Q)$. Below we give an idea about the efficient computation of such a chord set S. For the details and time complexity arguments we refer to [6].

First, the visibility region Q is determined. By Theorem 2.3, the region Q is a subpolygon of P whose edges are either portions of the edges of P or chords incident to reflex vertices of P. Each chord on the boundary of Q determines a half-polygon that does not contain Q. These half-polygons are called pockets. See Fig. 4 where pockets are shaded and Q is not. The algorithm processes each pocket separately.

Let P' be any of the pockets defined by Q and let c' be the chord on the boundary of Q determining P'.

The algorithm constructs a data structure for P' with respect to c' called a *window tree*, which was introduced by Suri in [26]. Each level of the window tree represents the visibility region (not necessarily connected) of all points in P' that are at equal link distance from c'. The nodes of the window tree correspond to the connected components of these regions. Two nodes in the window tree are connected if their regions share a common chord. So the root of the window tree contains c' itself. The first level contains the visibility region of c' in P' and so on. Note that, by our assumptions, the depth of the window tree can not exceed $r-1$.

The algorithm processes P' following the window tree upward and iteratively for $k = 1, \ldots, r-1$ constructs sets of chords E'_k. Sets E'_k are subsets of the set $\cup E_k(v)$, for those convex vertices v of P' whose link distance to c' are at least $r-3$ ($r-2$ in the case where central diagonal exists). At each step of iteration the algorithm discards as many chords that are provably "irrelevant" for the consequent computation of the set S, as possible.

For example, if it is established that $Q \subset P(s)$ for some chord s, then s is irrelevant and the algorithm discards it. Similarly, if it is established that $P(s) \subset P(s_1)$ for two chords s and s_1, then s_1 is irrelevant and it is discarded too. The first rule ensures that no more than one chord per a convex vertex is contained in each of the sets E'_k for $k < r-1$.The second rule ensures that no more than one chord incident to a reflex vertex is contained in each of the sets E'_k for $k < r-1$.

At the end the algorithm computes a minimal essential subset of $E_{r-1}(Q)$, where $E_{r-1}(Q)$ is the set of chords in E_{r-1} that intersect Q. The set S has $O(n)$ size and is constructed in $O(n \log n)$ time. See [6] for further details. Formally, we use the following result.

Theorem 2.8 [6] *Let P be a simple n-vertex polygon with link radius r. Let $E_{r-1}(Q)$ be the subset of chords in E_{r-1} intersecting Q. A minimal essential subset S of $E_{r-1}(Q)$ of $O(n)$ size can be determined in $O(n \log n)$ time.*

An illustration of the theorem for the case where a central diagonal exists and $r = 3$ is given in Fig. 4.

In summary, in this section the problem of finding a shortest central link segment has been reduced to the problem of finding a shortest segment that intersects a collection of half-polygons $P(s)$ defined by a set of chords S. The set S has linear size and is determined in $O(n \log n)$ time.

3. Characterization of a Central Link Segment

Recall that P denotes a simple, n-vertex polygon with link radius r. According to Theorem 2.5, the following two cases are possible: either P has a central diagonal (a diagonal whose link eccentricities in both directions are not less than r), or P has no central diagonal, but has a central triangle $\triangle$ (a triangle whose edges are diagonals with link eccentricities not greater than $r-1$ toward the direction not-containing $\triangle$). In Sections 3, 4, 5 and 6 we will present an algorithm for finding a shortest central link segment assuming the existence of a central diagonal. The

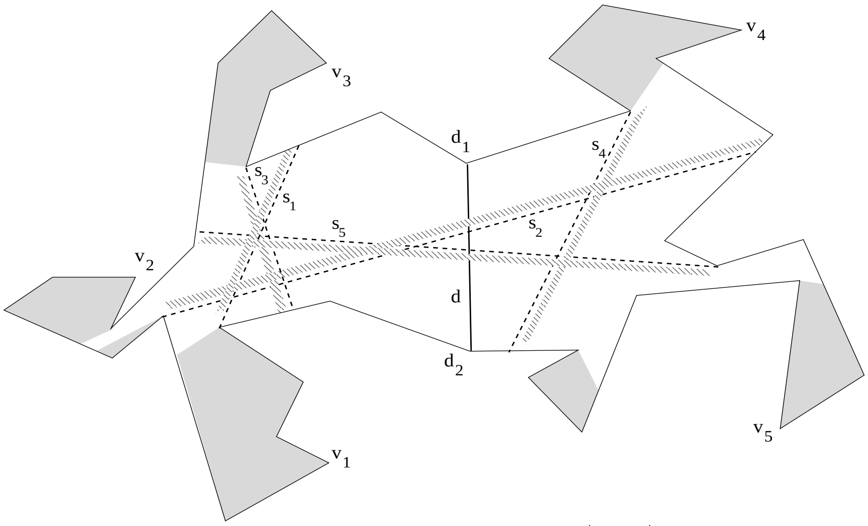

Fig. 4. A polygon with a central diagonal $d = (d_1, d_2)$. The region Q (i.e. visibility region of d) is not shaded. The minimal essential chord set S is $\{s_1, s_2, s_3, s_4, s_5\}$. Half-polygons $P(s_i)$ (illustrated by patterns along s_i) are half-polygons determined by s_i and containing v_i. The sets S_+, S_-, S_L, S_R, and S defined in Section 3 are as follows: $S_+ = \{s_2\}$, $S_- = \{s_5\}$, $S_L = \{s_1, s_3\}$, $S_R = \{s_4\}$.

case where no central diagonal exists requires minor changes to the algorithm and will be discussed in Section 7.

Let $d = (d_1, d_2)$ be a central diagonal in P. We denote by P_L and P_R the half-polygons determined by d. Let S be the minimal essential set of chords determined by Theorem 2.8. By the definition, d has eccentricity at least r in each of P_L and P_R. According to Corollary 2.7, S contains chords s_L and s_R so that $P(s_L) \subset P_L$ and $P(s_R) \subset P_R$. By Lemma 2.6, any segment with eccentricity $r-1$ must intersect all half-polygons $P(s)$ for $s \in S$. Since $P(s_L)$ and $P(s_R)$ are contained in P_L and P_R, correspondingly, it follows that any segment with eccentricity $r-1$ must intersect d. Hence the following holds.

Proposition 3.1 *Let P be an n-vertex simple polygon with a central diagonal d and covering radius $r_c = r - 1$. Then any central link segment in P intersects d.*

Assume now that the set S contains m chords that are denoted by $s_1, \ldots, s_m$. Let $\Pi = \{P(s_1), \ldots, P(s_m)\}$ be the set of their corresponding half-polygons. From minimality of S it follows that no half-polygon in Π contains another half-polygon in Π. Corollary 2.7 ensures the existence of chords s_L and s_R so that $P(s_L) \subset P_L$ and $P(s_R) \subset P_R$. Therefore no half-polygon in Π contains the central diagonal d entirely. Using that observation we partition the chords in S into three subsets S_L, S_R, and S_X depending on their position with respect to the central diagonal $d = (d_1, d_2)$. The subsets S_L and S_R contain all chords in S which do not intersect d and lie inside P_L and P_R, respectively. The set S_X contains all chords in S that intersect d. The set S_X is partitioned into two subsets S_+ and S_-, where S_+ contains the chords $s \in S_X$ for which $d_1 \in P(s)$ and S_- contains those for which

$d_2 \in P(s)$. Formally, sets S_L, S_R, S_X, and S_+, are defined as follows:

$$S_L = \{s \in S : s \cap d = \emptyset,\ P(s) \subset P_L\}; \quad S_R = \{s \in S : s \cap d = \emptyset,\ P(s) \subset P_R\};$$
$$S_X = \{s \in S : s \cap d \neq \emptyset\};$$
$$S_+ = \{s \in S_X : d_1 \in P(s)\}; \quad S_- = \{s \in S_X : d_2 \in P(s)\}.$$

See the example in Fig. 4, where sets S, S_L, S_R, S_X, S_+, and S_- are illustrated. In the next proposition we will establish a more accurate estimate on the number of segments in S than that given in Section 2.

Proposition 3.2 *The set S contains no more than $2n$ chords.*

Proof. Recall that each of the chords $s_1, \ldots, s_m$ is an edge of a visibility region $N_{r-1}(v)$ for some vertex v. Let us denote by $s_1(v), \ldots, s_{m(v)}(v)$ the chords in S that are edges of $N_{r-1}(v)$ for a fixed vertex v. Consider the complimentary half-polygons $\bar{P}(s_i(v)) = P \setminus P(s_i(v))$ for $i = 1, \ldots, m(v)$. By the definition of visibility region, no two of the half-polygons $\bar{P}(s_i(v))$ intersect. Furthermore, $d \cap \bar{P}(s_i(v))$ is non-empty, since d is not contained entirely in any of the half-polygons $P(s_i(v))$. This implies that at least one of the endpoints d_1 and d_2 of d belongs to each of the half-polygons $\bar{P}(s_i(v))$ for $i = 1, \ldots, m(v)$. Therefore there are no more than two such half-polygons per vertex v and the proposition is proved. $\square$

In what follows we will use the notion of *P-convex polygons* (or regions) introduced next. Half-polygons are simplest P-convex polygons. In general, a P-convex polygon is a polygon that is an intersection of half-polygons. Equivalently, a P-convex polygon is subpolygon of P whose vertices in the interior of P are convex. One can prove properties of P-convex polygons that are similar to the properties of convex point sets, e.g., that if C is a P-convex polygon and x and y are points in C, then the shortest link path between x and y in P is contained in C and that the intersection of two P-convex polygons is a P-convex polygon. Let us now define the following two particular P-convex polygons

$$C_L = \bigcap_{s \in S_L} P(s), \qquad C_R = \bigcap_{s \in S_R} P(s).$$

The polygons C_L and C_R can be readily constructed in $O(n \log n)$ time by intersecting half-polygons. Since the central diagonal separates C_L and C_R, the next proposition holds (Fig. 5).

Proposition 3.3 *Let (a, b) be a segment in P such that $a \in P_L$ and $b \in P_R$. The segment (a, b) intersects all half-polygons $P(s)$, for $s \in S_L \cup S_R$, iff $a \in C_L$ and $b \in C_R$.*

Assuming that C_L and C_R are not empty and $r_c = r - 1$, by Proposition 3.3 the problem of finding a shortest central link segment is reduced to finding a shortest segment with endpoints in C_L and C_R that intersects all $P(s)$ for $s \in S_X$. If one of the regions C_L and C_R is empty, Proposition 3.3 and Lemma 2.6 imply that $r_c = r$. In this case, by Corollary 2.1, any point in the link center is a shortest central link segment.

Now we show that, after the construction of C_L and C_R, the set S_X could be, possibly, further reduced without violating the validity of Lemma 2.6 and Proposition 3.3. The idea is to delete from S_X any chord whose left-endpoint is not in C_L

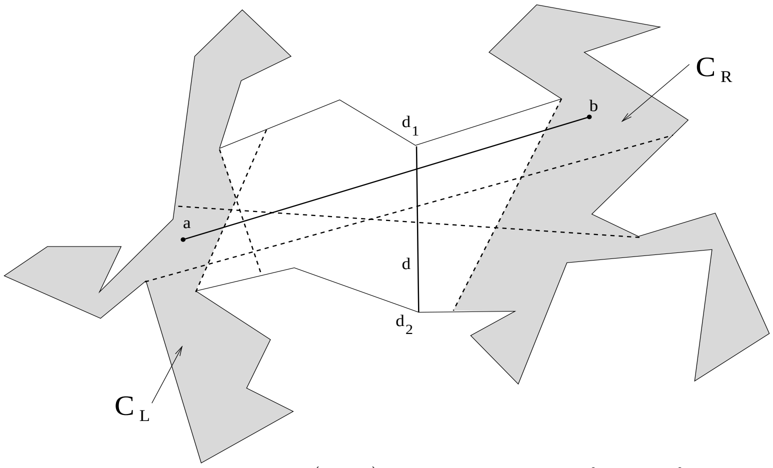

Fig. 5. P-convex region C_L (C_R) is the intersection of the half-polygons corresponding to the segments from S_L (S_R). A segment (a, b) intersects all half-polygons $P(s)$, for $s \in S_L \cup S_R$, iff $a \in C_L$ and $b \in C_R$.

or whose right-endpoint is not in C_R. Such a reduction will simplify the algorithm for finding a shortest central link segment.

Suppose that $s \cap C_L = \emptyset$ for some $s \in S_X$. Then either $P(s) \cap C_L = \emptyset$, or $P(s) \supset C_L$. If $P(s) \supset C_L$, then any segment intersecting C_L intersects $P(s)$ and hence s can be deleted from S_X. If $P(s) \cap C_L = \emptyset$, then we can move s from S_X to S_R and replace C_R by $C_R \cap P(s)$. Note that Proposition 3.3 still holds for the (possibly) reduced C_R . In both cases s is deleted from S_X.

The above reduction can be implemented in $O(n \log n)$ time as follows. We first sort the left-endpoints of the chords in each of S_+ and S_- along the boundary of P. Note that the right-endpoints inherit the order of the left ones since any pair of chords in S_+ and any pair in S_- intersect (recall that S is minimal). Next, while there is a chord $s \in S_X$ with no endpoint in C_L (C_R), we either reduce C_R (C_L) by intersecting it with $P(s)$ and move s from S_X to S_R (S_L), or just delete s. The procedure takes $O(n \log n)$ time since the number of steps does not exceed the number of the segments in S_X and each step is executed in at most $O(\log n)$ time. Thus, for the reduced P-convex polygons C_L and C_R, Proposition 3.3 holds and any of the remaining chords $s \in S_X$ intersects both the reduced C_L and the reduced C_R.

For the remainder let λ denote the number of left-endpoints of chords from S_X; let $x_1, \ldots, x_\lambda$ be these left-endpoints sorted along the boundary of P, i.e., $x_0 < x_1 < \ldots < x_\lambda < x_{\lambda+1}$, where x_0 and $x_{\lambda+1}$ are the endpoints of the boundary chain that belongs to C_L. The notation $z_1 < z_2 < z_3$ means that z_2 is between z_1 and z_3 on the clockwise polygonal chain from z_1 to z_3. See Fig. 6 for an example where $\lambda = 2$ and points $x_0, \ldots, x_3$ appear clockwise along the boundary of P.

By Proposition 3.3, the necessary and sufficient condition for a segment to intersect all half-polygons $P(s)$, for $s \in S_L \cup S_R$, is that its endpoints belong to C_L and C_R. Now we will study the problem of how to identify the chords with endpoints belonging to C_L and C_R that intersect all half-polygons $P(s)$, for $s \in S_X$.

To gain some insight into the problem, let us look at an example. Let us divide the boundary of P_L into subchains using the points $x_1, \ldots, x_\lambda$. Consider a segment (a, b) inside P such that the ray from b in the direction of a intersects the boundary of P at a point z such that $x_i \le z < x_{i+1}$ (Fig. 6). Let s be a chord from S_X such that (a, b) intersects $P(s)$. Note that s should have one endpoint in $x_1, \ldots, x_\lambda$. Then in the case where $x_i, x_{i+1} \in P(s)$ the point a must lie in $P(s)$ and in the case where $x_i, x_{i+1} \notin P(s)$ the point b must be in $P(s)$. Hence, if we denote by $C_{L,i}$ the intersection of all half-polygons $P(s)$ that contain x_i, x_{i+1}, and if (a, b) is a central link segment with its left endpoint a in $C_{L,i}$, then the right endpoint b must belong to all $P(s)$ that do not contain x_i, x_{i+1}. This observation leads to the following definition and lemma. For $i = 0, \ldots, \lambda$ we define the following pairs of regions illustrated in Fig. 6:

$$C_{L,i} := C_L \cap \{\cap P(s) : s \in S_X \text{ and } x_i, x_{i+1} \in P(s)\},$$

$$C_{R,i} := C_R \cap \{\cap P(s) : s \in S_X \text{ and } x_i, x_{i+1} \notin P(s)\}.$$

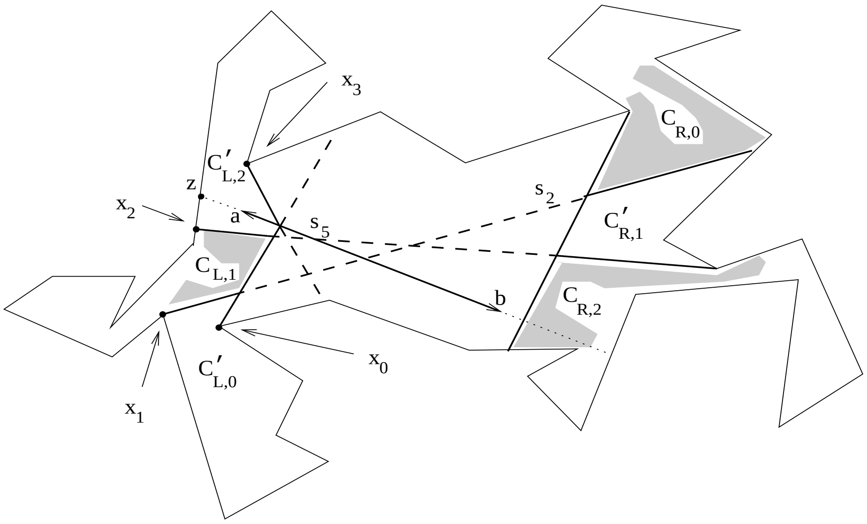

Fig. 6. In our example polygon the ray b toward a hits the boundary of P at z that is between x_2 and x_3 on the boundary of P. Regions $C_{L,1}$, $C_{R,0}$, $C_{R,2}$ are presented in the figure. Regions $C_{L,0}$, $C_{L,2}$, $C_{R,1}$ are not presented explicitly – instead we presented subregions $C'_{L,0}$, $C'_{L,2}$, $C'_{R,1}$ which are used only in this figure. Regions $C_{L,0}$, $C_{L,2}$, $C_{R,1}$ are obtained from those in the figure as follows $C_{L,0} = C'_{L,0} \cup C_{L,1}$, $C_{L,2} = C'_{L,2} \cup C_{L,1}$, and $C_{R,1} = C_R = C_{R,0} \cup C'_{R,1} \cup C_{R,2}$. If (a, b) is a central link segment then a must be in $C_{L,2}$ and b must be in $C_{R,2}$.

Regions $C_{L,i}$ and $C_{R,i}$ are P-convex polygons. It is possible that $C_{L,i}$ and $C_{L,j}$ (as well as $C_{R,i}$ and $C_{R,j}$) have nonempty intersection and share common parts on their boundaries for $i \ne j$. These regions will be studied in more detail in Section 5 (see, e.g., Fig. 7). Regions $C_{L,i}$ and $C_{R,i}$ are used in the following key lemma, which gives a characterization of a central link segment.

Lemma 3.4 *Let (a, b) be a segment in P for which $a \in P_L$ and $b \in P_R$. The segment (a, b) is a central link segment with link eccentricity $r-1$, iff there exists an index i, $0 \leq i \leq \lambda$, such that $a \in C_{L,i}$ and $b \in C_{R,i}$.*

Proof. "$\Leftarrow$" Assume that for some i, $0 \leq i \leq \lambda$, $a \in C_{L,i}$ and $b \in C_{R,i}$. By the definitions of $C_{L,i}$ and $C_{R,i}$, we have $a \in C_L$ and $b \in C_R$. Let s be any chord of S_X. If $x_i, x_{i+1} \in P(s)$, then from $a \in C_{L,i}$ and from the definition of $C_{L,i}$ it follows that $a \in P(s)$ and thus (a, b) intersects $P(s)$. If $x_i, x_{i+1} \notin P(s)$, then from $b \in C_{R,i}$ and from the definition of $C_{R,i}$ it follows that $b \in P(s)$ and again (a, b) intersects $P(s)$. "$\Rightarrow$" Let (a, b) be a segment in P with link eccentricity $r-1$ and with endpoints $a \in P_L$, $b \in P_R$, where r is the link radius of P. Then (a, b) intersects all subpolygons $P(s)$ for $s \in S$ and by Proposition 3.3 $a \in C_L$ and $b \in C_R$.

In order to prove that $a \in C_{L,i}$ for some i, let us consider the ray from b in the direction of a. Denote by z the point at which that ray meets the boundary of P (Fig. 6). Observe that $b \notin C_L$, since $b \in C_R$ and $C_R \cap C_L = \emptyset$. Then $z \in C_L$, since any of the half-polygons $P(s)$ whose intersection gives C_L contains a and at least one of them does not contain b.

We claim that $a \in C_{L,i}$, where i is such that $x_i \leq z < x_{i+1}$. Assume that $a \notin C_{L,i}$. This means that there exists a chord $s \in S_X$ such that $x_i, x_{i+1} \in P(s)$ and $a \notin P(s)$. Since z belongs to the polygon chain between x_i and x_{i+1}, we have $z \in P(s)$ and $a \notin P(s)$, implying $b \notin P(s)$ (Fig. 6). Then $(a, b) \cap P(s) = \emptyset$, which is a contradiction. The proof that b belongs to $C_{R,i}$ is analogous and thus omitted here. □

Hereafter we will denote by ∂Z the part of the boundary of a subpolygon Z of P that is inside P. Since $C_{L,i}$ and $C_{R,i}$ are closed point sets, the next claim follows.

Corollary 3.5 *If the covering radius of P is $r-1$, then there exists an index i, $0 \leq i \leq \lambda$, such that a shortest central link segment of P has endpoints on $\partial C_{L,i}$ and $\partial C_{R,i}$.*

4. Algorithm Outline

By Corollary 3.5, the problem of determining a shortest central link segment is reduced to determining a shortest segment restricted to lie in P and having endpoints on $\partial C_{L,i}$ and $\partial C_{R,i}$, for $i = 0, 1, \ldots, \lambda$. We can find such a segment by the following simple procedure: for $i = 0, 1, \ldots, \lambda$ and for each segment $q' \in \partial C_{L,i}$ and each segment $q'' \in \partial C_{R,i}$ find a shortest segment in P joining q' and q''. However, the number of pairs of segments in $\partial C_{L,i}$ and $\partial C_{R,i}$, for $i = 0, 1, \ldots, \lambda$, might be quadratic in n, in the worst case. Thus, in order to obtain an $O(n \log n)$ algorithm, we have to develop a more efficient approach.

Our idea is the following. Let $\mathcal{A}$ be the set of all segments that belong to the boundaries $\partial C_{L,i}$, for $i = 0, 1, \ldots, \lambda$. We shall prove later that $\mathcal{A}$ contains $O(n)$ segments. For any segment $q \in \mathcal{A}$, let $s(q)$ and $P(s(q))$ denote the chord from S containing q and the half-polygon corresponding to $s(q)$, respectively. We define $C_R(q)$ as the region with the following property: a segment σ with one endpoint on q and that lies in P has link eccentricity $r-1$ iff its other endpoint belongs to $C_R(q)$. Then our goal is, for each $q \in \mathcal{A}$, to determine the shortest segment joining

q and $C_R(q)$ in $O(\log n)$ time, and then to determine the shortest segment of the resulting $|\mathcal{A}| = O(n)$ candidates.

In order to implement this idea, we are going to obtain the set $\mathcal{A}$ appropriately sorted (as discussed below). Then we process segments in $\mathcal{A}$ in that sorted order and for each segment $q \in \mathcal{A}$ we construct its region $C_R(q)$ and find the shortest segment $\sigma(q)$ in P that joins q and $C_R(q)$. The key for the efficient generation of regions $C_R(q)$ and finding of segments $\sigma(q)$ is the order in $\mathcal{A}$ and an appropriate representation of the regions $C_R(q)$ based on pairs of indices $(i_1(q), i_2(q))$ assigned to each $q \in \mathcal{A}$.

The indices $i_1(q)$ and $i_2(q)$ will be computed, depending on the position of q with respect to the chords in S_+ and S_- so that they will determine completely the region $C_R(q)$ as follows. Let λ' and λ'' be the sizes of S_+ and S_-, respectively. Denote by $u_1 < \ldots < u_{\lambda'}$ the sorted sequence of the left-endpoints of the chords in S_+ and by $v_{\lambda''} < \ldots < v_1$ the sorted sequence of the left-endpoints of the chords in S_-. Note that no two segments of S_+ (or S_-) share the same endpoint, since otherwise S will not be minimal. Let $s(u_j)$ denote the chord in S_+ with endpoint u_j, for $j = 1, \ldots, \lambda'$, and $s(v_j)$ denote the chord in S_- with endpoint v_j, for $j = 1, \ldots, \lambda''$. The following formula, which will be proven in Proposition 5.2, gives a description of the regions $C_R(q)$ using the indices $i_1(q)$ and $i_2(q)$ (Fig. 10)

$$C_R(q) = C_R \cap \left\{ \bigcap_{j > i_1(q)} P(s(u_j)) \right\} \cap \left\{ \bigcap_{j > i_2(q)} P(s(v_j)) \right\}. \tag{1}$$

Our approach is outlined in algorithmic form below.

Algorithm 4.1: { Finds a shortest central link segment of P }
Input: A simple n-vertex polygon with central diagonal d and radius r.
Output: A shortest central link segment of P.

1. Compute the chord set S by using algorithms from [6].
2. Partition S into sets S_L, S_R, S_+, and S_-.
3. Reduce the chord set $S_X = S_+ \cup S_-$ and compute the reduced P-convex polygons C_L and C_R.
4. If $C_L = \emptyset$ or $C_R = \emptyset$ then return: "The covering radius of P is r and every point in the link center is a shortest link segment."
5. Sort the sequences of endpoints of the chords in S_+ and in S_- that belong to C_L along the boundary of P.
6. Construct the set of segments $\mathcal{A}$ and assign a pair of indices $(i_1(q), i_2(q))$ to each $q \in \mathcal{A}$. (The set $\mathcal{A}$ is obtained lexicographically sorted in increasing order on i_1 and decreasing order on i_2.)
7. For each $q \in \mathcal{A}$ find a shortest segment $\sigma(q)$ in P joining q and $C_R(q)$.
8. Return a shortest segment σ^* among the segments $\sigma(q)$, $q \in \mathcal{A}$.

Note that, although Algorithm 4.1 is based on Lemma 3.4 and Corollary 3.5, we do not explicitly construct the set of all pairs $\partial C_{L,i}$ and $\partial C_{R,i}$, for $i = 0, \ldots, \lambda$. The implementation of the first five steps of the above algorithm has been discussed above. The total time for their implementation is $O(n \log n)$. In the following Section 5 we focus on the implementation of Step 6 and in Section 6 we describe the implementation of Step 7. The efficiency of these implementations depends on the size of the set $\mathcal{A}$, which will be estimated next.

5. Computing the Segment Set $\mathcal{A}$

We begin this section by showing that $|\mathcal{A}| = 8n$. Then we describe an $O(n \log n)$ time implementation of Step 6 of Algorithm 4.1, i.e., the construction of the set $\mathcal{A}$ and the computation of the index pairs $(i_1(q), i_2(q))$ describing regions $C_R(q)$ for $q \in \mathcal{A}$.

Let us recall some relevant notation from the previous sections. We denoted by $x_1, \ldots, x_\lambda$ the sorted left-endpoints of the chords from S_X, and by x_0 and $x_{\lambda+1}$ the endpoints of the boundary chain ∂C_L. The sorted left endpoints of chords in S_+ and S_- were denoted by $u_1 < \ldots < u_{\lambda'}$ and by $v_{\lambda''} < \ldots < v_1$ respectively. Since we shall use these two sorted subsequences frequently, it will be convenient to have separate notation for them (Fig. 7):

$$U = \{x_0 = u_0 < u_1 < \ldots < u_{\lambda'} < u_{\lambda'+1} = x_{\lambda+1}\}$$

and

$$V = \{x_0 = v_{\lambda''+1} < v_{\lambda''} < \ldots < v_1 < v_0 = x_{\lambda+1}\}.$$

Note, that points x_0 and $x_{\lambda+1}$ are considered to belong to both subsequences U and V. Note also that in our notation $\lambda = \lambda' + \lambda''$ and that we assumed $x_0 = u_0 = v_{\lambda''+1}$ as well as $x_{\lambda+1} = u_{\lambda'+1} = v_0$. Finally, we denoted by $s(u_j)$ the chord in S_+ with left-endpoint $u_j \in U$ and by $s(v_j)$ the chord in S_- with left-endpoint $v_j \in V$.

Next, for each of the points x_i, we define a pair of P-convex regions $C_u(x_i)$ and $C_v(x_i)$ based on the ordered sequences U and V:

$$C_u(x_i) = \left\{ \bigcap_{\{j | u_j \leq x_i\}} P\left(s(u_j)\right) \right\} \cap C_L, \qquad C_v(x_i) = \left\{ \bigcap_{\{j | v_j \geq x_{i+1}\}} P\left(s(v_j)\right) \right\} \cap C_L,$$

where $P(s)$ denotes the half-polygon corresponding to the chord s. For $i = 0, \ldots, \lambda$, the intersections $C_u(x_i) \cap C_v(x_i)$ equal to the regions $C_{L,i}$ (see Lemma 3.4 and Fig. 7) whose boundaries $\partial C_{L,i}$ define the set $\mathcal{A}$. The regions $C_u(x_i)$, for $i = 0, \ldots, \lambda$, form a monotonically decreasing sequence (i.e. $C_u(x_{i+1}) \subset C_u(x_i)$), where each region is obtained from the previous one by intersection with a half-polygon. The same applies to regions $C_v(x_i)$ in reverse order, i.e., for $i = \lambda, \ldots, 0$. We shall use the monotonicity and this simple relation between consecutive regions for their efficient construction. The next lemma estimates the size of the set $\mathcal{A}$.

Lemma 5.1 *The set $\mathcal{A}$ contains no more than $8n$ segments.*

Proof. Recall that $\mathcal{A}$ consists of all segments on $\partial C_{L,i}$, for $i = 0, 1, \ldots, \lambda$. In our notation $C_{L,i} = C_u(x_i) \cap C_v(x_i)$ and clearly the number of the segments in $\partial C_{L,i}$

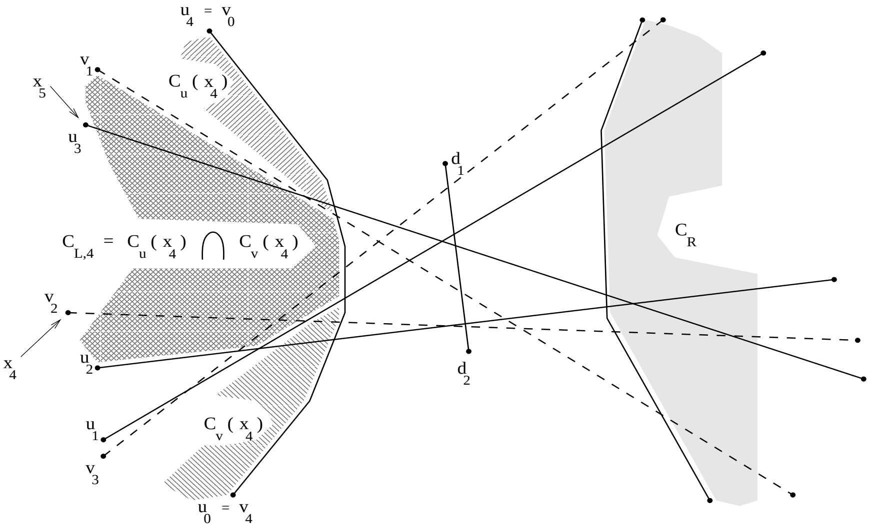

Fig. 7. Illustration to definitions from Section 5. There are three segments form S_+ (solid) and three segments form S_- (dashed). Their left endpoints plus the endpoints of ∂C_L form sequences U and V. Regions $C_u(x_4)$ and $C_v(x_4)$ as well as their intersection $C_{L,4} = C_u(x_4) \cap C_v(x_4)$ are also shown. On the right we present the region C_R.

can exceed the sum of the numbers of the segments in $\partial C_u(x_i)$ and $\partial C_v(x_i)$ by at most two. Therefore, the number of segments in $\mathcal{A}$ can exceed the number of the segments in $\cup_{i=0}^{\lambda}\{\partial C_u(x_i) \cup \partial C_v(x_i)\}$ by at most $2(\lambda+1)$.

We estimate the number of segments in this union. The definition of $C_u(x_i)$ implies (see Fig. 8):

$$\begin{array}{rcl} C_u(x_0) & = & C_L, \\ C_u(x_i) & = & \left\{ \begin{array}{ll} C_u(x_{i-1}) \cap P(s(x_i)) & \text{if } x_i \in U \\ C_u(x_{i-1}) & \text{if } x_i \in V. \end{array} \right. \end{array}$$

Since $P(s(x_i))$ is a half-polygon, $\partial C_u(x_i)$ contains at most two more segments than $\partial C_u(x_{i-1})$. Thus the number of segments in $\cup_{i=1}^{\lambda}\partial C_u(x_i)$ that do not belong to C_L is not greater than $2\lambda'$. Similarly, the number of segments in $\cup_{i=1}^{\lambda}\partial C_v(x_i)$ that do not belong to ∂C_L is not greater than $2\lambda''$. We thus obtain

$$|\mathcal{A}| \leq |\partial C_L| + 2(\lambda' + \lambda'') + 2\lambda + 2 = |\partial C_L| + 4\lambda + 2.$$

By using that $|\partial C_L| + |\partial C_R| + \lambda \leq |S|$, that ∂C_L, ∂C_R are not empty, and that $|S| \leq 2n$ (see Proposition 3.2), we get $|\mathcal{A}| \leq |\partial C_L| + 4\lambda + 2 < 8n$. □

In the sequel we shall often deal with portions of polygon (region) boundaries. We call these portions *chains*. Equivalently, a chain is an ordered sequence of segments that follows the boundary of a given polygon (region). A (simple) chain is called *convex* if it makes only left (or only right) turns during its traversal. For example, any chain on the boundary of a P-convex polygons lying inside P is convex. We assume that chains are represented as doubly linked lists.

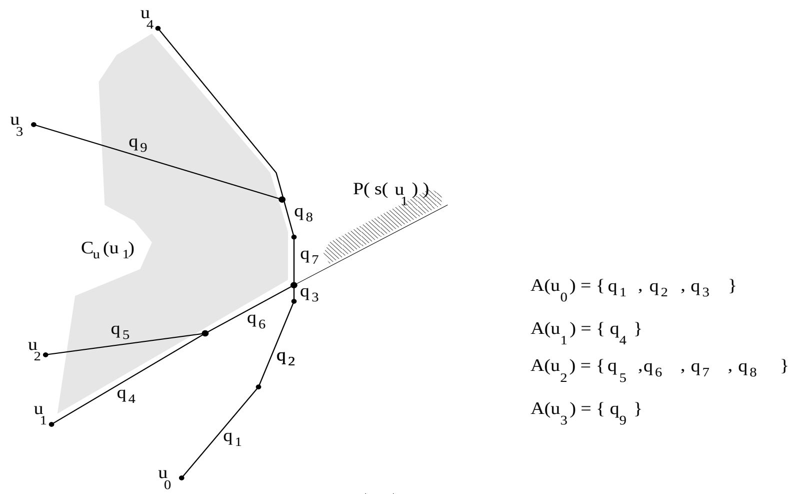

Fig. 8. Illustrates the region $C_u(u_1)$ is obtained as an intersection of $C_L = C_u(u_0)$ and $P(s(u_1))$. Segments forming non-intersecting chains $A(u_i)$ for $i = 0, 1, 2$ are presented.

We divide Step 6 of Algorithm 4.1 into two parts. In the first part, stated as Algorithm 5.1 below, we compute two sets of non-intersecting convex chains that form the boundaries $\partial C_u(u_j)$ for $j = 0, \ldots, \lambda'$ and $\partial C_v(v_j)$ for $j = 0, \ldots, \lambda''$. In the second part, stated as Algorithm 5.2, we use these chains to construct the set $\mathcal{A}$ and to compute and assign index pairs $(i_1(q), i_2(q))$ to the segments $q \in \mathcal{A}$.

Boundaries $\partial C_u(u_j)$, for $u_j \in U$, and $\partial C_v(v_j)$, for $v_j \in V$, respectively, contain common chains. Hence, it is not efficient to generate each of these boundaries separately. Instead, we compute two sets of disjoint convex chains

$$A(u_0), A(u_1), \ldots, A(u_{\lambda'}) \quad \text{and} \quad A(v_0), A(v_1) \ldots, A(v_{\lambda''}),$$

that form these boundaries. The chain $A(u_j)$, for $j = 0, \ldots, \lambda'$, is defined as the portion on the boundary of the region $C_u(u_j)$ that does not belong to $C_u(u_{j+1})$. Similarly, for $j = 0, \ldots, \lambda''$, the chain $A(v_j)$ is the portion of the boundary $\partial C_v(v_j)$ that is not in $C_v(v_{j+1})$. Formally, these chains are defined by

$$A(u_j) = \partial C_u(u_j) \setminus \partial C_u(u_{j+1}) \quad \text{for} \quad j = 0, \ldots, \lambda'$$

and

$$A(v_j) = \partial C_v(v_j) \setminus \partial C_v(v_{j+1}) \quad \text{for} \quad j = 0, \ldots, \lambda'',$$

where we assume $\partial C_u(u_0) = \partial C_u(u_{\lambda'+1}) = \partial C_v(v_0) = \partial C_v(v_{\lambda''+1}) = \partial C_L$. See Fig. 8 for an illustration. The total cardinality of these chains does not exceed the size of the set $\mathcal{A}$ and thus it is $O(n)$. Next our algorithm computes chains $A(u_j)$ for $j = 0, \ldots, \lambda'$ and $A(v_j)$ for $j = 0, \ldots, \lambda''$.

Algorithm 5.1:
Input: The chain ∂C_L and S_+ and S_- sorted by their left endpoints.
Output: The chains $A(u_j)$, $j = 0, 1, \ldots, \lambda'$, and $A(v_j)$, $j = 0, 1, \ldots, \lambda''$.

Phase A: Computes chains $A(u_j)$, $j = 0, 1, \ldots, \lambda'$.

1. Set $\partial C_u(u_0) = \partial C_L$.
2. For $j = 0, 1, \ldots, \lambda' - 1$
   find $A(u_j) = \partial C_u(u_j) \setminus (\partial C_u(u_j) \cap P(s(u_{j+1})))$ as follows:
   - 2.1. Process segments on $\partial C_u(u_j)$ in counterclockwise order starting at u_j and checking in which half-plane with respect to $s(u_{j+1})$ they lie until the intersection between $s(u_{j+1})$ and $\partial C_u(u_j)$ is found.
   - 2.2. Form $A(u_j)$ by inserting in it all segments $q \in \partial C_u(u_j)$ that do not belong to $P(s(u_{j+1}))$ and following their counterclockwise order on $\partial C_u(u_j)$. Assign a pointer to the end of $A(u_j)$ pointing to the intersection point between $s(u_{j+1})$ and $\partial C_u(u_j)$.
   - 2.3. Construct $\partial C_u(u_{j+1})$ by replacing the chain $A(u_j)$ in $\partial C_u(u_j)$ with the subsegment $s(u_{j+1}) \cap C_u(u_j)$.
3. In the same manner, construct (the last) chain $A(u_{\lambda'}) = \{q : q \in \partial C_u(u_{\lambda'}) \setminus \partial C_L\}$.

Phase B: Computes analogously chains $A(v_j)$,for $j = 0, 1, \ldots, \lambda''$.

Note that any of the boundaries $\partial C_u(u_j)$, for $j = 0, \ldots, \lambda'$, can be easily listed by using the output of Algorithm 5.1. For example to list $\partial C_u(u_j)$ for some j we list the chain $A(u_j)$ first. Then using the pointer assigned to the end of $A(u_j)$; jump into the chain $A(u_{j_1})$ ($j_1 > j$) containing the endpoint of $A(u_j)$; list the portion of $A(u_{j_1})$ after the pointer and continue until the end of $A(u_{\lambda'})$ is reached. Indeed the same applies for the boundaries $\partial C_v(v_j)$ for $j = 0, \ldots, \lambda''$.

Algorithm 5.1 runs in time linear in the size of its input, which is $O(n)$ according to Lemma 5.1. For an illustration see Fig. 8, where the region $C_u(u_1)$ is shown as an intersection between $C_u(u_0)$ and $P(s(u_1))$. The chain $A(u_0)$ in this case consists of three segments q_1, q_2, q_3.

Next we present Agorithm 5.2 which, using the chains output by Algorithm 5.1 as input, constructs the set $\mathcal{A}$ and assigns index pairs $(i_1(q), i_2(q))$ to the segments in $\mathcal{A}$. Each segment in $\mathcal{A}$ is either a segment or a subsegment of a segment from $\cup_{j=0}^{\lambda'} A(u_j)$ or $\cup_{j=0}^{\lambda''} A(v_j)$. The subsegments are determined by intersection points between pairs of chains $\partial C_u(x_i)$ and $\partial C_v(x_i)$. For example, in Fig. 9, the point $y_1 = \partial C_u(u_1) \cap \partial C_v(v_2)$ divides each of the two segments containing it into two subsegments.

It is convenient to partition the set $\mathcal{A}$ into two subsets $\{\mathcal{A}_u, \mathcal{A}_v\}$, where $\mathcal{A}_u$ contains segments that lie on boundaries $\partial C_u(u_j)$ for $j = 0, \ldots, \lambda'$ and $\mathcal{A}_v$ contains segments that lie on boundaries $\partial C_v(v_j)$ for $j = 0, \ldots, \lambda''$. Here we describe only the construction of $\mathcal{A}_u$; the construction of $\mathcal{A}_v$ is analogous.

Algorithm 5.2 simultaneously sweeps the sequences U and V and finds intersection points $y = \partial C_u(u_i) \cap \partial C_v(v_j)$ for appropriately chosen indices i and j. The algorithm inserts into $\mathcal{A}_u$ portions of the chains $A(u_0), \ldots, A(u_{\lambda'})$ depending on their position with respect to the currently computed intersection point y. The first index $i_1(q)$ assigned to a segment q that enters $\mathcal{A}_u$ is the index of the chain $A(u_j)$ to which q belongs. The second index $i_2(q)$ is the maximal index j such that $q \in C_v(v_j)$ (Fig. 9).

Algorithm 5.2:
Input: The chains $A(u_i)$, $i = 0, 1, \ldots, \lambda'$, and $A(v_j)$, $j = 0, 1, \ldots, \lambda''$ (as computed by Algorithm 5.1).
Output: The set of segments $\mathcal{A}$ and index pairs $(i_1(q), i_2(q))$ for all $q \in \mathcal{A}$. (The set $\mathcal{A}$ is obtained sorted in increasing order on i_1 and decreasing order on i_2.)

Phase A: Computes the set $\mathcal{A}_u$ and index pairs $(i_1(q), i_2(q))$ for $q \in \mathcal{A}_u$.

Set $i = 0$, $j = \lambda''$ and repeat the following steps while $i \leq \lambda'$ and $j \geq 0$:

1. Find the intersection point $y \in \partial C_u(u_i) \cap \partial C_v(v_j)$. Let $q_u \in \partial C_u(u_i)$ and $q_v \in \partial C_v(v_j))$ be the segments containing y and $\kappa(y)$ and $\mu(y)$ be the indices such that $q_u \in A(u_{\kappa(y)})$ and $q_v \in A(v_{\mu(y)})$.
(In the case where $\partial C_u(u_i)$ and $\partial C_v(v_j)$ share a common part on ∂C_L, set y to be the first point of the common part in clockwise order on ∂C_L (see y_2 on Fig. 9).)
2. For $i_1 = i, \ldots, \kappa(y) - 1$ insert segments of the chain $A(u_{i_1})$, following their counterclockwise order, into $\mathcal{A}_u$. Assign index pair $(i_1, \mu(y))$ to each of these segments.
3. Divide segment q_u into two subsegments q'_u and q''_u sharing y and replace q_u by q'_u and q''_u in $A(u_{\kappa(y)})$. Insert in $\mathcal{A}_u$ the portion of the chain $A(u_{\kappa(y)})$ that lies between $u_{\kappa(y)}$ and y, assigning to each such segment q the index pair $(\kappa(y), \mu(y))$. Discard the inserted portion from $A(u_{\kappa(y)})$.
4. Let $j(y)$ be the minimum index $j > 0$ such that $y \in C_v(v_j)$.
If $v_{j(y)} > u_{\mu+1}$, then set $i = \mu(y)$ and $j = j(y) - 1$.
If $v_{j(y)} < u_{\mu+1}$, then set $i = \mu(y) + 1$ and $j = j(y) - 1$.

Phase B: Computes analogously the set $\mathcal{A}_v$ and index pairs $(i_1(q), i_2(q))$ for $q \in \mathcal{A}_v$.

The algorithm is illustrated in Fig. 9. It is easily verified that Algorithm 5.2 correctly constructs the set $\mathcal{A}$, since it inserts in $\mathcal{A}$ those and only those segments that belong to the boundaries $\partial\left(C_u(x_i) \cap C_v(x_i)\right)$ for $i = 0, \ldots, \lambda$.

The algorithm can be implemented to run in time proportional to the size of its input and output, which according to Proposition 3.2 and Lemma 5.1 is $O(n)$. The only step that needs some additional comments, in that respect, is Step 1 computing intersection point $y = \partial C_u(u_i) \cap \partial C_v(v_j)$ for indices i and j assigned in

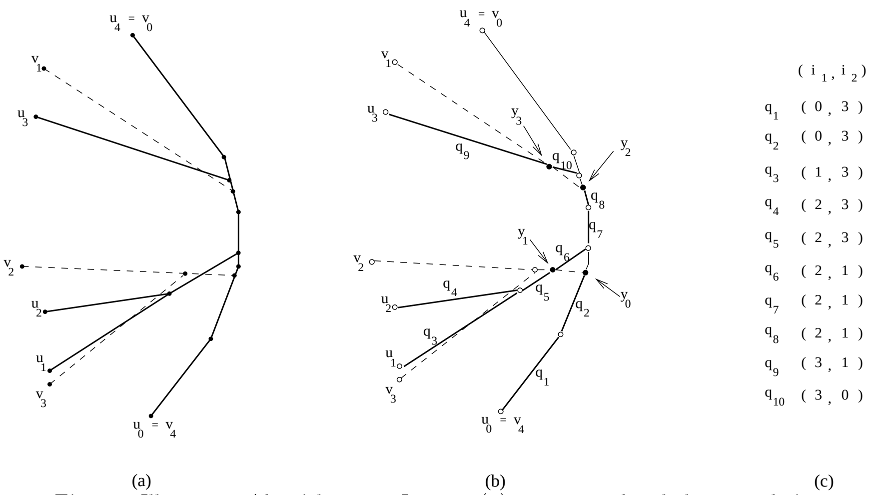

Fig. 9. Illustrates Algorithm 5.2. In part (a) segments that belong to chains $A(u_i)$ and $A(v_j)$ for $i, j = 0, 1, 2, 3$ are shown. Compare with Fig. 8 where $A(u_i)$ are listed explicitly. In part (b) segments $\{q_1, \ldots, q_{10}\}$ form part $\mathcal{A}_u$ of $\mathcal{A}$. Intersection points $y_0, \ldots, y_3$ are shown too. In part (c) index pairs (i_1, i_2) assigned to segments in $\mathcal{A}_u$ are listed.

Step 4. First, in our settings y is unique. Next, for indices i and j as assigned by the algorithm, the start-point of the chain $A(u_i)$ (which is either u_i or the last computed point y) belongs to $C_v(v_j)$ and the endpoint of $A(v_j)$ does not belong to $C_u(u_i)$. Thereby, we find the intersection point y by a simultaneous counterclockwise sweep on $\partial C_u(u_i)$ and $A(v_j)$. This is done by checking the position of the current point on $A(v_j)$ against the current segment on $\partial C_u(u_i)$ and advancing on one of the chains depending on the result of the check. This sweep takes time proportional to the size of the portion of $\partial C_u(u_i)$ that is inside $C_v(v_j)$ plus the size of the subchain of $A(v_j)$ that is outside $C_u(u_i)$. In total, this step (and the whole algorithm) makes no more than a constant number of computations on any of the segments in $\mathcal{A}$ and therefore takes $O(n)$ time.

Finally, we shall show that Eqn. (1) holds for the index pairs assigned by Algorithm 5.2. See Fig. 10 for an illustration. Consider a segment $q \in \mathcal{A}$ and its index pair $(i_1(q), i_2(q))$. The first index $i_1(q)$ is just the index of the input chain $A(u_{i_1(q)})$ containing q. By our definitions it follows that $i_1(q)$ is the maximum index such that $q \in C_u(u_{i_1(q)})$. Otherwise q belongs to the any of the half-polygons $P(s(u_j))$ for $j \leq i_1(q)$ and does not belong to any of the half-polygons $P(s(u_j))$ for $j > i_1(q)$. Similarly, one can see that $i_2(q)$ is equal to the maximum index such that q belongs $C_v(v_{i_2(q)})$ and again $q \in P(s(v_j))$ if and only if $j \leq i_2(q)$.

Let a be a point on q and (a, b) be a segment in $P \setminus P(s(q))$, where $s(q)$ is the chord in S containing q. The segment (a, b) has eccentricity $r - 1$ if and only if (a, b) intersects all half-polygons $P(s)$ for $s \in S$. We know that $a \in q$. Therefore, we have that $a \in C_L$, $a \in P(s(u_j))$ iff $j \leq i_1(q)$, and $a \in P(s(v_j))$ iff $j \leq i_2(q)$.

Consequently the segment (a, b) has eccentricity $r-1$ if and only if

$$b \in C_R \cap \left\{ \bigcap_{j > i_1(q)} P(s(u_j)) \right\} \cap \left\{ \bigcap_{j > i_2(q)} P(s(v_j)) \right\},$$

which implies Eqn. (1). We summarize the results of this section in the following statement.

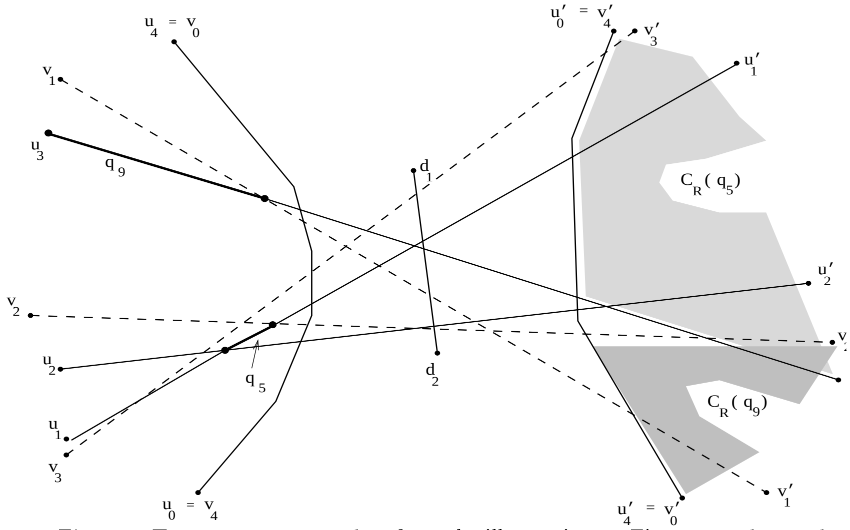

Fig. 10. Two segments q_5 and q_9 from the illustration on Fig. 9 are taken and their corresponding regions $C_R(q_5)$ and $C_R(q_9)$ are shown.

Proposition 5.2 *Algorithms* 5.1 *and* 5.2 *compute the set* $\mathcal{A}$ *and pairs of indices* $(i_1(q), i_2(q))$ *for* $q \in \mathcal{A}$ *in* $O(n)$ *time. Index pairs* $(i_1(q), i_2(q))$ *define* $C_R(q)$ *by Eqn.* (1). *The segments in* $\mathcal{A}$ *are stored in non-decreasing order of their* i_1*-index and non-increasing order of their* i_2*-index.*

6. Finding a Shortest Central Link Segment

In this section we describe an $O(n \log n)$ time implementation of Step 7 of Algorithm 4.1. In this step, for each segment $q \in \mathcal{A}$, a shortest segment $\sigma(q) \in P$ that joins q and $C_R(q)$ is computed.

The idea of the algorithm is the following. A simple polygon X is called X*-shaped* if its boundary can be decomposed into four outwards convex chains X_1, X_2, X_3, X_4. Thus X has at most four convex vertices. For each $q \in \mathcal{A}$ we will construct an X-shaped polygon $X(q)$ that contains both q and $\partial C_R(q)$ on its boundary. The polygon $X(q)$ will have the property that any segment that joins q and $\partial C_R(q)$ in P stays within $X(q)$ and vice versa. This, in particular, holds for a shortest segment $\sigma(q)$. For an illustration, see the bold polygon $X(q)$ in Fig. 11. Thus, finding of $\sigma(q)$ has now been reduced to a problem introduced by Bhattarchaya and Toussaint [4] as "computing the wingspan of a butterfly" and whose $O(\log n)$ solution has been given in [17]. Actually, a particular instance of that problem since one endpoint of the shortest segment in our case must lie on q. Since we will use the result from [17], it is stated in Lemma 6.1.

Lemma 6.1 *Let X be an X-shaped n-vertex polygon consisting of the outwards convex chains X_1, X_2, X_3, X_4. Given a representation of X allowing binary search, a shortest segment in X joining X_1 and X_3 can be found in $O(\log n)$ time.*

We shall show that, by considering the segments q by their order in $\mathcal{A}$, polygons $X(q)$ can be generated and shortest segments $\sigma(q)$ joining q and $\partial C_R(q)$ can be determined for all $q \in \mathcal{A}$ in $O(n \log n)$ total time. Recall that Algorithm 5.2 constructs the segment set $\mathcal{A}$ in two phases A and B. During phase A, the segments inserted into $\mathcal{A}$ belong to the boundaries $\partial C_u(u_i)$, consecutively for $i = 0, \ldots, \lambda'$. We denoted this part of $\mathcal{A}$ by $\mathcal{A}_u$. During phase B, segments that belong to the boundaries $\partial C_v(v_i)$ are inserted into $\mathcal{A}$, for $i = 0, \ldots, \lambda''$, consecutively, and this part of $\mathcal{A}$ was denoted by $\mathcal{A}_v$. We describe the construction of shortest segments $\sigma(q)$ for $q \in \mathcal{A}_u$. Finding the segments $\sigma(q)$ for $q \in \mathcal{A}_v$ is analogous.

Let q be any segment in $\mathcal{A}_u$. We assume that $C_R(q)$ is nonempty, since otherwise q can not contain an endpoint of a central link segment and our algorithm proceeds by examining the next segment in $\mathcal{A}$. Let us denote by $SP(x, y)$ the shortest Euclidean path inside P between points x and y in P. We define $X(q)$ as a sequence of the following four chains

$$X(q) = \left\{ \partial C_u(u_{i_1(q)}), SP(v_0, v'_{i_2(q)+1}), \partial C_R(q), SP(u'_{i_1(q)+1}, u_{i_1(q)}) \right\},$$

where $u'_1 < \ldots < u'_{\lambda'}$ and $v'_{\lambda''} < \ldots < v'_1$ denote the right endpoints of the segments in S_+ and S_-, respectively (see Fig. 11). Recall that each pair of segments in S_+ (S_-) intersect and therefore their right endpoints inherit the order of the left endpoints. We denote by u'_0 and $u'_{\lambda'+1}$ the endpoints of the boundary chain of P belonging to C_R and assume that $v'_{\lambda''+1} = u'_0$, $v'_0 = u'_{\lambda'+1}$. To simplify the notation, we let

$$\partial C_L(q) = \partial C_u(u_{i_1(q)}),\ Up(q) = SP(v_0, v'_{i_2(q)+1}),\ Dn(q) = SP(u'_{i_1(q)+1}, u_{i_1(q)})$$

and consequently

$$X(q) = \{\partial C_L(q), Up(q), \partial C_R(q), Dn(q)\}.$$

Lemma 6.2 *For any $q \in \mathcal{A}$ the corresponding polygon $X(q)$ is X-shaped. Any segment in P joining q and $C_R(q)$ lies in $X(q)$.*

Proof. To prove that $Dn(q)$ is outwards convex, we use the fact that if two points $x, y \in \partial P$ are visible from a point $z \in P$, then $SP(x, y)$ is outwards convex [12]. The chain $Dn(q)$ is a shortest path between $u_{i_1(q)}$ and $u'_{i_1(q)+1}$, which are visible from the intersection point of the segments $s(u_{i_1(q)})$ and $s(u'_{i_1(q)+1})$ (Fig. 11). Similarly, we prove that $Up(q)$ is outwards convex.

Chains $\partial C_L(q)$ and $\partial C_R(q)$ are outwards convex, since they are portions of the boundaries of P-convex polygons lying inside P (see Section 5 and Eqn. (1)).

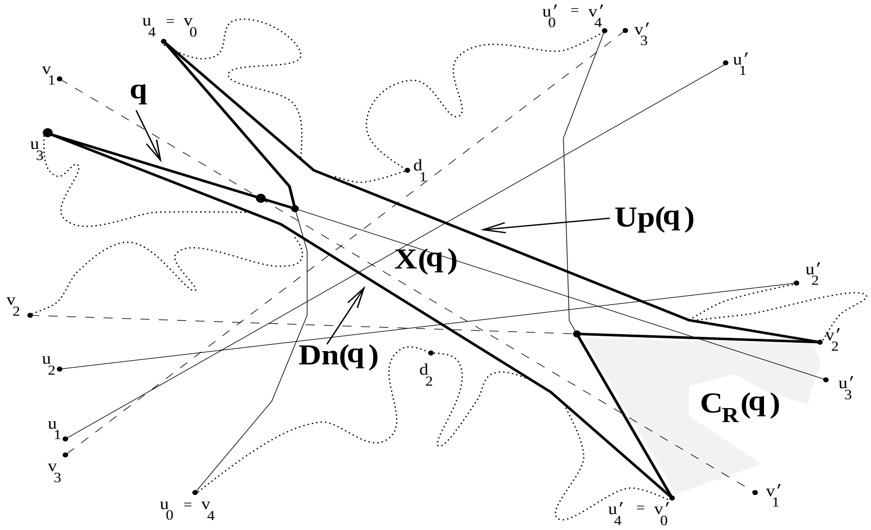

Fig. 11. The polygon $X(q)$ for a segment q corresponding to q_9 from Fig. 10. The region $C_R(q)$ is shaded. Chains $Up(q)$ and $Dn(q)$ are shortest Euclidean paths $SP(v_0, v_2')$ and $SP(v_0', u_3)$ in this case. The doted curve corresponds to the boundary of P that is relevant to chains $Up(q)$ and $Dn(q)$.

Thus, to prove that $X(q)$ is X-shaped, we need only to show that $X(q)$ is simple. Chains $\partial C_L(q)$ and $\partial C_R(q)$ do not intersect since they are separated by the central diagonal d. Next, observe that if $P(s)$ is a half-polygon and $x, y \in P(s)$, then $SP(x, y) \in P(s)$. Therefore neither chain $\partial C_L(q)$ nor $\partial C_R(q)$ intersects $Up(q)$ or $Dn(q)$, since $\partial C_L(q)$ and $Up(q)$ are separated by $s(v_0)$, $\partial C_L(q)$ and $Dn(q)$ are separated by $s(u_{i_1(q)})$, $\partial C_R(q)$ and $Dn(q)$ are separated by $s(u'_{i_1(q)+1})$, and $\partial C_R(q)$ and $Up(q)$ are separated by $s(v'_{i_2(q)+1})$.

It remains to show that $Up(q)$ and $Dn(q)$ do not intersect. For this, we observe that since $C_R(q) \neq \emptyset$, then $v'_{i_2(q)+1} < u'_{i_1(q)+1}$ and therefore

$$u_{i_1(q)} < u_{i_1(q)+1} \leq v_0 < v'_{i_2(q)+1} < u'_{i_1(q)+1}.$$

Thus $Up(q)$ and $Dn(q)$ are separated by the segment $s(u_{i_1(q)+1})$, if $i_1(q) < \lambda'$, and either by $s(u_{\lambda'})$ or by $s(v_{i_2(q)+1})$, if $i_1(q) = \lambda'$. See Fig. 11 for an illustration.

Thus $X(q)$ is X-shaped. Any segment in P joining q and $\partial C_R(q)$ lies in $X(q)$, since the chains $Up(q)$ and $Dn(q)$ are shortest paths. $\square$

Lemma 6.1 can now be used to find the shortest segment joining $\partial C_L(q)$ and $\partial C_R(q)$. We need, however, to find a shortest segment that joins q and $C_R(q)$. The next lemma shows how one can use Lemma 6.1 to solve this particular problem. In fact, it shows that the chain containing q in $X(q)$ is not used and it suffices to compute the three other chains – namely, $Dn(q)$, $\partial C_R(q)$ and $Up(q)$.

Lemma 6.3 *If the chains $Dn(q)$, $\partial C_R(q)$, and $Up(q)$ of $X(q)$ are given in a form allowing binary search, then the shortest segment joining q and $C_R(q)$ can be found in $O(\log n)$ time.*

Proof. Let $q = (a, b)$, where endpoints a and b are listed in clockwise order on $\partial C_L(q)$. Construct the tangent segments t_a from a to $Dn(q)$ and t_b from b to

$Up(q)$. Let x and y be the common vertices of t_b and $Up(q)$ and of t_a and $Dn(q)$, respectively. Clearly, points x and y can be found in $O(\log n)$ time by binary search on chains $Dn(q)$ and $Up(q)$. Then we can apply Lemma 6.1 on the (new) X-shaped polygon formed by the segment q, t_b plus the portion of the chain $Up(q)$ from x to $v'_{i_2(q)+1}$, the chain $\partial C_R(q)$, and the portion of $Dn(q)$ from $u'_{i_1(q)+1}$ to y plus t_a. In order to do a binary search on (say) the chain composed of t_b and the portion of the chain $Up(q)$ from x to $v'_{i_2(q)+1}$, we just do a binary search on $Up(q)$ and ignore the vertices that are not between x and $v'_{i_2(q)+1}$. Similarly, we can do a binary search on the portion of $Dn(q)$. □

Our approach can be outlined in algorithmic form as follows:
Algorithm 6.1:
Input: The outputs of Algorithms 5.1, 5.2, and an analogue to Algorithm 5.1 for ∂C_R , and S_X.
Output: The set of shortest segments $\sigma(q)$ for $q \in \mathcal{A}$.

Phase A: Computes the set of shortest segments $\sigma(q)$ for $q \in \mathcal{A}_u$.

1. Compute chains $Dn(q_1)$, $\partial C_R(q_1)$ and $Up(q_1)$, where q_1 is the first segment in $\mathcal{A}_u$.
2. Traverse the ordered segment set $\mathcal{A}_u$ and for each segment $q \in \mathcal{A}_u$ do steps 2.1 and 2.2below.
   - 2.1. Compute chains $Dn(q)$, $\partial C_R(q)$, and $Up(q)$ by updating their predecessors $Dn(q_1)$, $\partial C_R(q_1)$, and $Up(q_1)$, where q_1 is the predecessor of q in $\mathcal{A}_u$.
   - 2.2. Compute $\sigma(q)$ as described in Lemma 6.3.

Phase B: Computes analougously the set of shortest segments $\sigma(q)$ for $q \in \mathcal{A}_v$.

Step 1 of the above algorithm is implemented in $O(n)$ time. We shall show that the generation of chains $Dn(q)$, $\partial C_R(q)$ and $Up(q)$ (Step 2.1) in total takes $O(n \log n)$ time. Additionally we show that these chains will be represented in a form that allows binary search. Thus Step 2.2 takes $O(\log n)$ per segment and $O(n \log n)$ in total. Therefore we need to describe the implementation of Step 2.1 only.

Let us address first some data structure issues. As we mentioned in the previous section we use doubly linked lists to store chains. But this data structure itself does not support efficient binary search. To overcome this difficulty we employ three arrays and store our chains (lists) $Dn(q)$, $\partial C_R(q)$ and $Up(q)$ in these arrays. The size of the arrays has to be an upper bound on the size of the chains that are going to be stored there. So, we use two arrays of length n for the chains $Dn(q)$ and $Up(q)$ and one of length $2n$ for $\partial C_R(q)$. With each chain stored in our arrays we maintain two indices pointing to the first and last segment of the chain. As we shall see by our construction, changes between consecutive chains will occur only at the two ends of our lists and therefore they can be done in time proportional to the number of deletions and insertions while the consistency of the array representation is preserved. Corresponding updates of the two end-indices are done modulo length of the array (i.e., n or $2n$) and take constant time. Obviously, the described simple data structure allows binary search to be carried out in $O(\log n)$ time. During our discussion below we shall refer to $Dn(q)$, $\partial C_R(q)$ and $Up(q)$ simply as chains and assuming the above representation.

Now we describe the implementation of Step 2.1 of Algorithm 6.1. We shall describe the maintenance of the chains $\partial C_R(q)$ first. We assume Algorithm 5.1 has been run with respect to the region C_R and chains $A_R(u'_j)$ for $j = 0, \ldots, \lambda'$ and $A_R(v'_j)$ for $j = 0, \ldots, \lambda''$ have been computed and equipped with pointers for efficient listing of the boundaries of the regions

$$C_u^R(u'_j) = C_R \cap \left\{ \bigcap_{j'>j} P(s(u'_{j'})) \right\} \quad \text{and} \quad C_v^R(v'_j) = C_R \cap \left\{ \bigcap_{j'>j} P(s(v'_{j'})) \right\}.$$

According to Eqn. (1) we have

$$C_R(q) = C_R(i_1(q), i_2(q)) = C_u^R(u'_{i_1(q)}) \cap C_v^R(v'_{i_2(q)}).$$

By the order in $\mathcal{A}_u$ we have that $i_1(q) \geq i_1(q_1)$ and $i_2(q) \leq i_2(q_1)$. Thereby the chain $\partial C_R(i_1(q), i_2(q))$ can be obtained from $\partial C_R(i_1(q_1), i_2(q_1))$ by consecutive updates of the following two types:

$$\partial C_R(i_1, i_2) \longrightarrow \partial C_R(i_1 + 1, i_2) \qquad \text{and} \qquad \partial C_R(i_1, i_2) \longrightarrow \partial C_R(i_1, i_2 - 1).$$

Consider the first type of update. Denote by $y = y(i_1, i_2)$ the intersection point between the chains $\partial C_u^R(u'_{i_1})$ and $\partial C_v^R(v'_{i_2})$. To construct the chain $\partial C_R(i_1 + 1, i_2)$ we consider the segment s on the boundary $\partial C_u^R(u'_{i_1+1})$ that follows the last segment of the chain $A_R(u'_{i_1+1})$.

If the segment $s \in \partial C_R(i_1, i_2)$, then we obtain the chain $\partial C_R(i_1 + 1, i_2)$ from $\partial C_R(i_1, i_2)$ by replacing the portion of segments after s in the chain $\partial C_R(i_1, i_2)$ by the chain $A_R(u'_{i_1+1})$. The update in this case takes time proportional to the size of the chain $A_R(u'_{i_1+1})$ plus the number of discarded segments after s. The total time for these type updates is readily seen to be $O(n)$.

Consider now the case where $s \notin \partial C_R(i_1, i_2)$. Now we apply the sweep procedure described for Step 1 of Algorithm 5.2 and find the intersection point

$$y(i_1 + 1, i_2) = \partial C_u^R(u'_{i_1+1}) \cap \partial C_v^R(v'_{i_2}).$$

The procedure begins with the last segment in $A_R(u'_{i_1+1})$ on the $\partial C_u^R(u'_{i_1+1})$ and with the segment after $y(i_1, i_2)$ on $\partial C_v^R(v'_{i_2}))$ and scans these chains counterclockwise until the intersection point $y(i_1 + 1, i_2)$ is found. Next we obtain the chain $\partial C_R(i_1 + 1, i_2)$ from $\partial C_R(i_1, i_2)$ by replacing the portion after $y(i_1, i_2)$ in $\partial C_R(i_1, i_2)$ with the segments between $y(i_1, i_2)$ and $y(i_1 + 1, i_2)$ on the boundary $\partial C_v^R(v'_{i_2})$ plus the segments before $y(i_1 + 1, i_2)$ in the chain $A(u'_{i_1+1})$ (i.e., on the boundary $\partial C_u^R(u'_{i_1+1})$). The total time for these type updates is again $O(n)$ since any segment on the boundaries $\partial C_u^R(u'_j)$ and $\partial C_v^R(v'_j)$ can be subject to one scan, one insertion, and one deletion.

Updates of the type $\partial C_R(i_1, i_2) \longrightarrow \partial C_R(i_1, i_2 - 1)$ are symmetric to those described above and can be treated analogously. We omit further details here. Next, consider the maintenance of the chains $Up(q)$ and $Dn(q)$. We shall describe

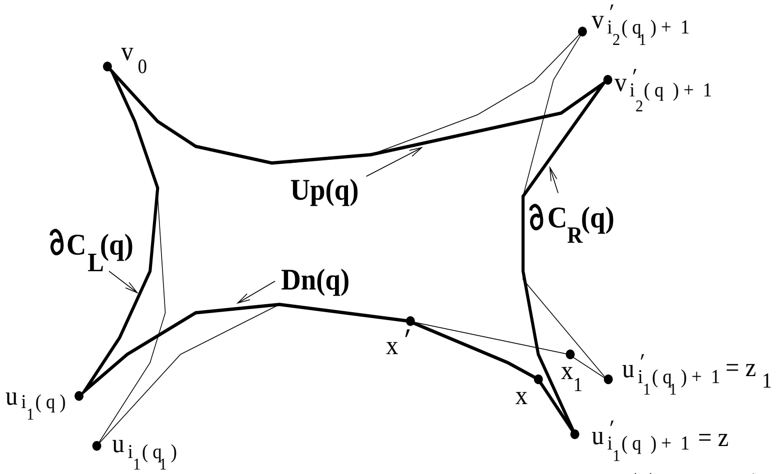

Fig. 12. Illustrates the maintenance of X-shaped polygons $X(q)$ for $q \in \mathcal{A}_u$. Two consecutive polygons $X(q_1)$ and $X(q)$ (bold line) are shown.

only how the chain $Dn(q)$ is maintained; the maintenance of the chain $Up(q)$ is done in the same manner. Recall that $Dn(q)$ is the shortest path between $u'_{i_1(q)+1}$ and $u_{i_1(q)}$. We will make use of the shortest path algorithm from [12]. After an $O(n)$ time preprocessing of an n-vertex polygon, that algorithm outputs the first γ vertices (segments) of the shortest path between any two query vertices in $O(\gamma + \log n)$ time. We assume that P has already been preprocessed for answering shortest path queries. By using the above shortest path algorithm, we will update path $Up(q_1)$ and obtain $Up(q)$ making changes only at the ends, while leaving the common part of the old and new shortest path unchanged.

Denote $z = u'_{i_1(q)+1}$, $z_1 = u'_{i_1(q_1)+1}$ and let (z_1, x_1) be the first segment from $Up(q_1)$. Compute the first segment (z, x) of the shortest path between $u'_{i_1(q)+1}$ and $u_{i_1(q)}$ in $O(\log n)$ time. Determine the relative position of x_1 and x on the boundary of P. If $x < x_1 < v_0$, then replace (z_1, x_1) by the shortest path from z to x_1 computed by the algorithm from [12]. If $x_1 < x < v_0$ (Fig. 12), then replace all segments from $Up(q_1)$ down to the first segment incident to a vertex x' greater or equal to x by the shortest path from z to x'. In either case, the time for maintaining the front portion of $Dn(q)$ is $O(\log n)$ plus time proportional to the number of deletions and insertions. The other end of the path $Dn(q_1)$ is updated in a similar manner.

The total number of deletions and insertions when maintaining $Dn(q)$ is proportional to the number of different segments that participate in these chains, which is $O(n)$. The number of the generated chains $Dn(q)$ is $O(n)$ and we spend $O(\log n)$ additional time per each update, which results in $O(n \log n)$ total time for the maintenance of the chains $Dn(q)$.

We summarize the discussion in the following lemma.

Lemma 6.4 *The set of shortest segments $\sigma(q)$ for $q \in \mathcal{A}$ is computed by Algorithm 6.1 in $O(n \log n)$ time.*

Combining the results from Sections 4, 5, and 6 we obtain Proposition 6.5 below.

Proposition 6.5 *In the case where a central diagonal exists, a shortest link segment in an n-vertex polygon can be found in $O(n \log n)$ time.*

7. The Case where no Central Diagonal Exists

Throughout this section we assume that no central diagonal d exists. We show that this case can be treated essentially in the same way as the case discussed in Sections 3, 4, 5, and 6.

If we choose any chord s of S and apply the algorithms described in Sections 3, 4, 5, and 6 on P with d denoting s, then the output will be a shortest central link segment among all central link segments intersecting $d \equiv s$ (if any). To find a shortest central link segment of P, it suffices to find three chords d_1, d_2, and d_3 in S such that their corresponding half-polygons $P(d_1)$, $P(d_2)$ and $P(d_3)$ have no common point. Then any segment that intersect $P(d_1)$, $P(d_2)$, and $P(d_3)$ must intersect at least one of the chords d_1, d_2, and d_3. Consequently any central link segment intersects at lest one of the chords d_1, d_2, and d_3. Therefore, if we know such a triple of chords d_1, d_2, and d_3, we can find a shortest central link segment by applying the algorithms described in Sections 3, 4, 5 and 6, for $d = d_1$, $d = d_2$, and $d = d_3$, respectively.

Below we present a simple algorithm that finds such a triple in $O(n \log n)$ time. Essentially we use the fact that $\bigcap_{s \in S} P(s) = \emptyset$, which by Helly's theorem implies the existence of such a triple.

Algorithm 7.1:

Input: The set $S = \{s_1, \ldots, s_m\}$.
Output: Chords d_1, d_2, and d_3 from S such that $P(d_1) \cap P(d_2) \cap P(d_3) = \emptyset$.

1. Set $P_i = P(s_i)$ for $i = 1, \ldots, m$. Find the minimum $i_1 \leq m$ such that $\bigcap_{i=1}^{i_1} P_i = \emptyset$. (Then $\bigcap_{i=1}^{i_1 - 1} P_i \neq \emptyset$.)
2. For $i = 1, 2, \ldots, i_1$ set $P'_i = P_i \cap P_{i_1}$.
(Note that by the Helly's theorem there exist $i_3 < i_2 < i_1$ such that $P_{i_3} \cap P_{i_2} \cap P_{i_1} = \emptyset$ and therefore $P'_{i_3} \cap P'_{i_2} = \emptyset$.)
3. Find the minimum $i_2 < i_1$ such that $\bigcap_{i=1}^{i_2} P'_i = \emptyset$. (Then $\bigcap_{i=1}^{i_2 - 1} P'_i \neq \emptyset$.)
4. Find $i_3 < i_2$ such that $P'_{i_3} \cap P'_{i_2} = \emptyset$. (Such i_3 exists by the remark in Step 2 and by the fact that $P'_{\bar{i}_3} \cap P'_{\bar{i}_2} \neq \emptyset$ for any $\bar{i}_3 < \bar{i}_2 < i_2$).
5. Return $d_1 = s_{i_1}$, $d_2 = s_{i_2}$, and $d_3 = s_{i_3}$.

Since the intersection between a P-convex polygon with a half-polygon can be found in a logarithmic time, Algorithm 7.1 runs in $O(n \log n)$ time. Thus the following result holds.

Proposition 7.1 *If no central diagonal exists, a shortest central link segment of any simple n-vertex polygon can be found in $O(n \log n)$ time.*

We summarize the main result of this paper in the following theorem.

Theorem 7.2 *A shortest central link segment of any simple n-vertex polygon can be found in $O(n \log n)$ time.*

A corollary of Theorem 7.2 is that both the robot placement problem and the k-visibility problem have efficient solutions. We state this result in the following theorem.

Theorem 7.3 *The optimal robot placement problem and the k-visibility problem for any simple n-vertex polygon can be solved in $O(n \log n)$ time.*

8. Conclusion

In this paper we have presented an $O(n \log n)$ algorithm for finding a shortest central link segment in an n-vertex simple polygon. Applications for this work include k-visibility and robot placement. The work provides new tools and insight into link distance related problems. It is conjectured that this work may lead to the development of a linear time algorithms for solving the shortest central link segment, link center and other related problems. The authors have made progress in this direction. An interesting problem which still needs to be solved in total linear time, is to find the shortest segments inside X-shaped polygons arising in Section 6. The relationship among these polygons needs to be exploited here.

Acknowledgements

The authors are grateful to reviewer, J.O'Rourke, for providing valuable comments which resulted in an improved presentation of this paper.

Refernces

1. L. Aleksandrov, H. Djidjev, and J.-R. Sack, *Finding a central link segment of a simple polygon in $O(n \log n)$ time*, Technical Report TR89-7, Bulgarian Academy of Sciences; also Technical Report SCS-TR-163, School of Computer Science, Carleton University, 1989.
2. D. Avis and G. Toussaint, *An optimal algorithm for determining the visibility polygon from an edge*, *IEEE Transactions on Computers*, C-30, pp. 910-914, 1981.
3. E. M. Arkin, J. S. B. Mitchell, and S. Suri. Logarithmic-time link path queries in a simple polygon. *Internat. J. Comput. Geom. Appl.*, 5(4):369–395, 1995.
4. B. Bhattacharya, P. Egyed, and G. Toussaint. *Computing the wingspan of a butterfly*, *Proceedings of the Third Canadian Conference on Computational Geometry*, pp. 88-91, 1991.
5. G. Das and G. Narasimhan, *Optimal Linear-Time Algorithm for the Shortest Illuminating Line Segment in a Polygon*, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994.
6. H. Djidjev, A. Lingas and J.-R. Sack, *An $O(n \log n)$ algorithm for computing the link center of a simple polygon*, *Discrete and Computational Geometry*, 8, pp. 131-152, 1992.

7. H. Djidjev, A. Maheshwari, and J.-R. Sack, *Link distance problems*, in J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1999.

8. J.-I. Doh and K.-Y. Chwa, An algorithm for determining visibility of a simple polygon from an internal line segment, *Journal of Algorithms*, 14, pp. 139-168, 1993.

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

10. L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Internat. J. Comput. Geom. Appl.*, 3(4):383–415, December 1993.

11. L. Guibas, J. Hershberger, and J. Snoeyink, *Compact interval trees: a data structure for convex hulls*, *Intern. J. of Computational Geometry & Applications*, 1, pp. 1-22, 1991

12. L. Guibas, J. Hershberger, *Optimal shortest path queries in a simple polygon*, *Journal of Computer and Systems Sciences*, vol. 39, pp. 126–152, 1989.

13. Y. Ke, *Polygon visibility algorithms for weak visibility and link distance problems*, Ph.D. Thesis, Johns Hopkins University, 1989.

14. Y. Ke, *An efficient algorithm for link distance problems inside a simple polygon*, In *Proccedings of the Fifth Annual ACM Symposium on Computational Geometry*, pp. 69-78, 1989.

15. K. Kolarov and B. Roth. On the number of links and placement of telescoping manipulators in an environment with obstacles. In *Proc. International Conference on Advanced Robotics*, 1991.

16. S. Kahan and J. Snoeyink, *On the bit complexity of minimum link paths: Superquadratic algorithms for problems solvable in linear time*, in *Proceedings of the 12-th Annual ACM Symposium on Computational Geometry*, pp. 151–158, 1996.

17. D. Kirkpatrick and J. Snoeyink, *Computing constrained shortest segments: butterfly wingspans in logarithmic time*, in *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pp. 163–168, 1993.

18. W. Lenhart, R. Pollack, J.-R. Sack, R. Seidel, M. Sharir, S. Suri, G. Toussaint, S. Whitesides and C. Yap, *Computing the link center of a simple polygon*, Discrete and Computational Geometry, 3, pp. 281-293, 1988.

19. A. Lingas, A. Maheshwari and J.-R. Sack, *Optimal parallel algorithms for rectilinear link distance problems*, *Algorithmica*, 14, 261-289, 1995.

20. A. Maheshwari. *Parallel algorithms for minimum link path and related problems.* Ph.D. thesis, Tata Institute of Fundamental Research, Bombay, India, 1993.

21. R. B. McMaster. Automated line generalization. *Cartographica*, 26:74–111, 1987.

22. B. Natarajan. On comparing and compressing piece-wise linear curves. Report, Hewlett Packward, 1991.

23. F. Preparata and M. Shamos, *Computational Geometry*, Springer-Verlag, New York, Heidelberg, 1985.

24. J. Reif and J. A. Storer. Minimizing turns for discrete movement in the interior of a polygon. *IEEE Journal of Robotics and Automation*, 3:182–193, 1987.

25. J.-R. Sack and S. Suri, An optimal algorithm for detecting weak visibility of a polygon, *IEEE Transactions on Computers*, 39, pp. 1213-1219, 1990.

26. S. Suri, Minimum link paths in polygons and related problems. Ph.D. Thesis, De-

partment of Computer Science, Johns Hopkins University, Baltimore, 1987.

27. R. T. Tamassia. On embedding a graph in the grid with minimum number of bends. *SIAM Journal on Computing*, 16, pp. 421-444, 1987.