On rectilinear link distance

On Rectilinear Link Distance

Mark de Berg

RUU-CS-89-13
May 1989

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

On Rectilinear Link Distance

Mark de Berg

RUU-CS-89-13
May 1989

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

On Rectilinear Link Distance

Mark de Berg

Technical Report RUU-CS-89-13
May 1989

Department of Computer Science
University of Utrecht
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

On Rectilinear Link Distance

Mark de Berg*

Abstract

Given a simple polygon P without holes all of whose edges are axis-parallel, a rectilinear path in P is a path that consists of axis-parallel segments only and does not cross any edge of P. The length of such a path is defined as the number of segments it consists of and the rectilinear link distance between two points in P is defined as the length of the shortest path connecting the two points.

We devise a data structure using $O(n \log n)$ storage such that given any two query points s and t in P, we can efficiently compute a shortest path from s to t. For the case where both query points are vertices of P the query time is $O(1 + l)$, where l is the length of a shortest path. If the query points are arbitrary points inside P then the query time becomes $O(\log n + l)$. The path that is found is not only optimal in the rectilinear link metric, it is shown to be optimal in the L_1-metric as well.

As a second problem we compute the diameter of a rectilinear polygon P. The diameter of P is the maximum distance between any two points in P. It is shown that the exact diameter can be computed in time $O(n \log n)$ and an approximation with an error of at most three in $O(n)$ time.

1 Introduction

In a simple polygon P the link distance between two points is defined as the minimum number of line segments inside P needed to connect the two points, not crossing any edge of the polygon ([17]). The introduction of this metric is motivated by the fact that often, e.g. in motion planning or broadcasting problems, it is relatively expensive to take a turn. Recently problems concerning link distance have gained a lot of attention. The problems of finding furthest neighbours for points in P and of computing the diameter of P have been studied by Suri ([17]) and Ke ([8]). Lenhart et al. ([10]) were the first to study the link centre problem: compute the set of points in P whose link distance to their furthest neighbour is minimal. They gave an $O(n^2)$ algorithm. The problem of computing the link centre in time $O(n \log n)$,

*Department of Computer Science, University of Utrecht, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands. Supported by the Dutch Organisation for Scientific Research (N.W.O.).

as posed in [13], has been solved independently by Djidjev, Lingas and Sack ([5]) and Ke ([8]).

In this paper the axis-parallel version of link distance is studied in which paths consist of axis-parallel line segments, a realistic restriction in some motion planning problems. Such paths have already been studied, e.g. in [4, 9, 14, 15], where shortest rectilinear paths in the L_1-metric are sought. Instead, we are interested in shortest rectilinear paths in the link distance metric. We will restrict ourselves in this paper to paths inside rectilinear polygons (i.e., polygons having all edges axis-parallel) without holes. (See Sack ([15]) for a general study of rectilinear polygons.)

Definition 1 *Let P be a simple rectilinear polygon. A* (rectilinear) path π (in P) *is an ordered set $\{l_1, l_2, \ldots, l_d\}$ of axis-parallel line segments (inside P) such that the beginpoint of every l_i ($1 < i \leq d$) coincides with the endpoint of l_{i-1} and no l_i ($1 \leq i \leq d$) crosses an edge of P. The length of the path, denoted as $length(\pi)$, is the number of segments it consists of.*

If $\{l_1, l_2, \ldots, l_d\}$ is the ordered set of segments of a path π, then we write $\pi = l_1 l_2 \cdots l_d$.

Definition 2 *Let s and t be two points in a simple rectilinear polygon P. The* rectilinear link distance *between s and t, denoted $d(s,t)$, is defined as: $d(s,t) = \min\{length(\pi) | \pi$ is a path in P connecting s and $t\}$.*

Two problems concerning rectilinear link distance in simple rectilinear polygons (without holes) are studied: the Query Problem and the Diameter Problem. The Query Problem asks to store a polygon P such that, given two query points s and t in P (the *source* and the *target*), a shortest path between s and t can be computed efficiently. The Diameter Problem asks to compute the diameter of P, defined as $Dia(P) = \max\{d(s,t) | s, t \in P\}$.

The sequel of this paper is organised as follows.

In section 2 Chazelle's polygon cutting theorem ([2]) is adapted to rectilinear polygons. Let P be a rectilinear polygon on n vertices $v_1, \cdots, v_n$, each assigned a real positive weight $c(v_i)$, without holes. It is shown that P can be cut with a segment lying totally inside P into two subpolygons both of total weight $\leq \frac{3}{4}C(P)$, where $C(P)$ is the total weight of P. Moreover, the cut segment can be found in linear time. Both the Query Problem and the Diameter Problem are solved by a divide-and-conquer approach making use of this Rectilinear Polygon Cutting Theorem.

In section 3 the Query Problem is considered. A structure is devised that uses $O(n \log n)$ storage in which a shortest path between two query points can be found in time $O(\log n + l)$, where l is the length of a shortest path. If the query points are vertices of the polygon then a shortest path can be found in time $O(1 + l)$. It is shown that the path that is found by our algorithm is not only optimal in the rectilinear link metric, but also in the L_1-metric.

In section 4 it is shown how the diameter of a rectilinear polygon can be computed in time $O(n \log n)$ with a divide-and-conquer algorithm. Furthermore we give a simple recursive algorithm that computes an approximation D of the diameter with $|D - Dia(P)| \leq 3$ in linear time.

Finally, in section 5, we briefly summarize our results and indicate directions for further research.

2 The Rectilinear Polygon Cutting Theorem

In this section a rectilinear version of Chazelle's Polygon Cutting Theorem ([2]) is presented. Chazelle's result can be stated as follows:

Theorem 1 ([2]) *Let P be a simple polygon on n vertices, each assigned a weight $\in \{0,1\}$, and let $C(P)$ be the total weight of the vertices. Then a diagonal between two vertices of P and lying totally inside P exists that cuts P into two polygons of weight $\leq \frac{2}{3}C(P)$. This diagonal can be computed in time $O(n)$, assuming a sorted list along some axis of the vertices is given.*

In this theorem it is assumed that the weights of the vertices incident upon the diagonal are set to zero in the resulting polygons; otherwise 2 should be added to the term $\frac{2}{3}C(P)$.

To prove his theorem, Chazelle first determines a vertical segment that cuts P into two polygons of weight $\leq \frac{2}{3}C(P)$ and then finds the desired diagonal. It would seem that, using the method to find the vertical segment, we can always find a vertical segment in our rectilinear polygon that cuts P into two polygons of weight $\leq \frac{2}{3}C(P)$. In finding the vertical segment, however, Chazelle assumes that no two vertices lie on the same vertical line. We cannot make this assumption without loss of generality, since we are dealing with rectilinear polygons and we have to extend the proof to cases where there are more vertices lying on the same vertical line.

Before we state our theorem we introduce some notation. In the remainder of this section, the vertices $v_1, \ldots, v_n$ of a rectilinear polygon P are always numbered in counterclockwise order with v_1 being the lowest of all leftmost vertices; $c(v_i)$ will be the real positive weight of vertex v_i and $C(P) = \sum_{i=1}^{n} c(v_i)$ is the weight of P. An axis-parallel segment is called a *cut segment (of P)* if it connects two edges of P and lies entirely inside P.

Theorem 2 *Let P be a rectilinear polygon having n vertices without holes. Then a cut segment exists that cuts P into two polygons having weight $\leq \frac{3}{4}C(P)$[1]. Moreover, this segment can be chosen such that it is incident upon at least one vertex.*

Proof: Following Chazelle's proof, we move a vertical cut segment through the polygon, hoping that one moment it will meet the requirements. Because we must

[1] As Chazelle, we assume that of the vertices incident upon the cut segments get weight zero; if not $2\max\{c(v_i) | 1 \leq i \leq n\}$ should be added to the term $\frac{3}{4}C(P)$.

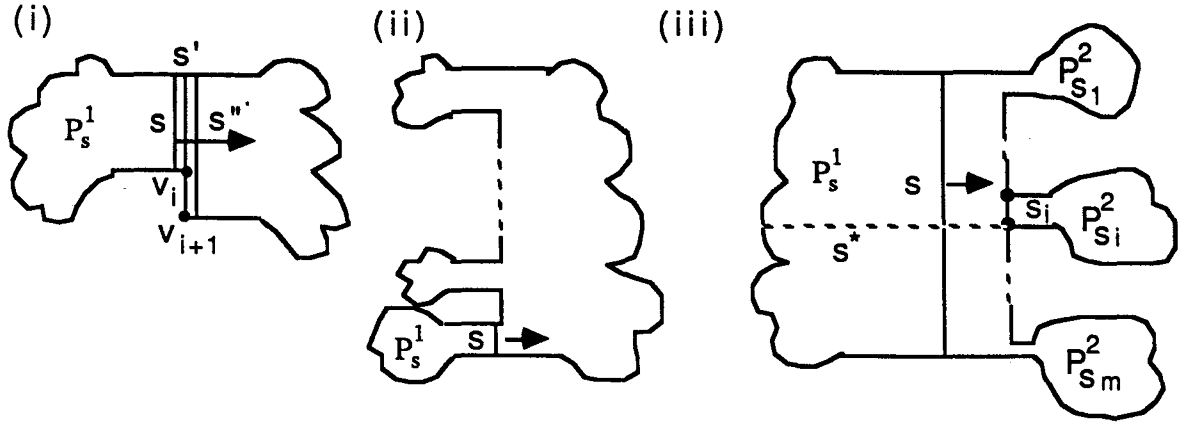


Figure 1: The three cases to consider when moving s.

also consider more vertices lying on the same vertical line (as we already noted, this degeneracy must explicitly be dealt with since the edges of the polygon as well as the cut segment are axis-parallel) this will not always be the case, but if not then we can show that a horizontal cut segment exists with the desired property.

A cut segment s connecting edges $\overline{v_iv_{i+1}}$ and $\overline{v_jv_{j+1}}$ $(j > i)$ cuts P into two polygons P_s^1 and P_s^2 (the parts of P lying resp. to the left and to the right of the segment s directed from $\overline{v_iv_{i+1}}$ to $\overline{v_jv_{j+1}}$), having weights $C(P_s^2) \leq c(v_{i+1})+\cdots+c(v_j)$ and $C(P_s^1) \leq C(P) - C(P_s^2)$. Here inequality holds when s is incident upon one or more of the vertices v_i, v_{i+1}, v_j or v_{j+1} whose weight(s) is (are) then set to zero. We start with $s = \overline{v_1v_n}$, and thus $C(P_s^1) = 0$ and $C(P_s^2) = C(P) - c(v_1) - c(v_2)$, and begin moving s to the right. We continue moving s in a way to be described below. When we reach a dead end then $C(P_s^2) = 0$. Along the way $C(P_s^2)$ decreases monotonously, hopefully by not too great amounts so that it will attain a value $\leq \frac{3}{4}C(P)$. Since we will make sure that always $C(P_s^1) \leq \frac{3}{4}C(P)$, we are done when $C(P_s^2) \leq \frac{3}{4}C(P)$.

There are three cases to consider when moving s (see Figure 1). We can always assume that $C(P_s^2) > \frac{3}{4}C(P)$ and thus $C(P_s^1) < \frac{1}{4}C(P)$, otherwise we would already have stopped.

In case (i) we have the following possibilities: Let s' be the cut segment that is incident upon v_i and s'' the cut segment just after passing v_i (refer to Figure 1 (i)). If $C(P_{s''}^2) > \frac{3}{4}C(P)$ we just continue moving s to the right, if $\frac{1}{4}C(P) \leq C(P_{s''}^2) \leq \frac{3}{4}C(P)$, then clearly s'' cuts P as desired and if $C(P_{s''}^2) < \frac{1}{4}C(P)$ then s' meets the requirements: $C(P_{s'}^1) = C(P_s^1) < \frac{1}{4}C(P)$ and $C(P_{s'}^2) = C(P_{s''}^2) < \frac{1}{4}C(P)$. (This assumes that $c(v_i)$ is set to zero in $P_{s'}^1$ and, since v_i vanishes in $P_{s'}^2$, $c(v_{i+1})$ is set to zero in $P_{s'}^2$.)

The second case is in fact the reverse of the third, so we will concentrate on the third case. Here there are two possibilities to consider (refer to Figure 1(iii)). If there is a $C(P_{s_j}^2) \geq \frac{1}{4}C(P)$, then either s_j, the segment that cuts off $P_{s_j}^2$, cuts P

in the desired way or (when $C(P^2_{s_j}) > \frac{3}{4}C(P)$) we can proceed into $P^2_{s_j}$. If there is no such $P^2_{s_j}$, then there is no vertical cut segment meeting the requirements. In this case, however, there is a horizontal cut segment that cuts P as desired: since $C(P^2_{s_j}) < \frac{1}{4}C(P)$ for all $1 \leq j \leq m$, there is an i such that $0 \leq \sum_{j=1}^{i} C(P^2_{s_j}) - \sum_{j=i+1}^{m} C(P^2_{s_j}) < \frac{1}{4}C(P)$. From this it follows that the horizontal segment s^* that cuts P into two polygons $P^1_{s^*}$, containing $P^2_{s_1}, \ldots, P^2_{s_i}$ plus the part of P^1_s above s^*, and $P^2_{s^*}$, containing $P^2_{s_{i+1}}, \ldots, P^2_{s_m}$ plus the part of P^1_s below s^*, has the desired property:

$$
\begin{aligned}
C(P^1_{s^*}) &\leq \sum_{j=1}^{i} C(P^2_{s_j}) + C(P^1_s) \\
&< \frac{1}{4}C(P) + \sum_{j=i+1}^{m} C(P^2_{s_j}) + C(P^1_s) \\
&\leq \frac{1}{4}C(P) + C(P) - C(P^1_{s^*}) + C(P^1_s) \qquad \Longrightarrow \\
C(P^1_{s^*}) &< \frac{5}{8}C(P) + \frac{1}{2}C(P^1_s) \\
&< \frac{3}{4}C(P)
\end{aligned}
$$

and

$$
\begin{aligned}
C(P^2_{s^*}) &\leq \sum_{j=i+1}^{m} C(P^2_{s_j}) + C(P^1_s) \\
&\leq \sum_{j=1}^{i} C(P^2_{s_j}) + C(P^1_s) \\
&< \frac{3}{4}C(P)
\end{aligned}
$$

We can conclude that we will always find a cut segment that cuts P into two polygons of weight $\leq \frac{3}{4}C(P)$.

It remains to show that the segments can be chosen to be incident upon at least one vertex. The segments added to handle holes clearly satisfy this condition, so we are left with the cut segment of a hole-free polygon. Suppose that we have found a cut segment s that is not incident upon a vertex and assume w.l.o.g. that s is vertical. Now move s to the right until it hits a vertex. If this vertex is an endpoint of one of the segments that are connected by s, then we are ready. Otherwise we are more or less in the situation of Figure 1 (iii) and, if moving s to the left doesn't help either, we can show in the same way that a horizontal cut segment exists. This segment s^* is incident upon a vertex (see Figure 1 (iii)). Details are left to the reader. □

Remark 1: Observe that this theorem can easily be extended to handle to polygons

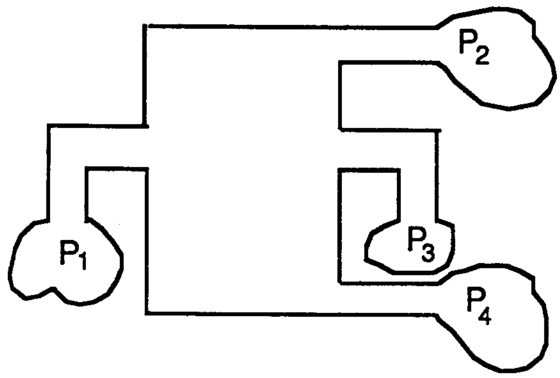


Figure 2: A polygon that cannot be cut such that the weight of the two resulting polygons is $< \frac{3}{4}C(P)$. Take $C(P_1) = C(P_2) = C(P_3) = C(P_4) = \frac{1}{4}C(P)$.

containing holes. If P contains k holes then we can find $l \leq k+1$ cut segments that cut P into two subpolygons of weight $\leq \frac{3}{4}C(P)$, as follows: We first remove the holes by adding vertical edges from the rightmost topmost vertex of every hole to the opposite edge and duplicating these edges as to obtain a polygon without holes. This can be done in $O(n \log n)$ time by a simple sweep line algorithm. Then we can apply the above procedure to find one cut segment. The other cut segments are the extra edges of whose duplicates only one is traversed when walking around one of the resulting polygons. Note that each of these extra cut segments destroys one hole, so the total number of remaining holes in the two new polygons is $k+1-l$.

Remark 2: The bound of $\frac{3}{4}C(P)$ in the above theorem is sharp. Figure 2 shows a polygon that cannot be cut any better. If, however, in the situations of Figure 1(ii) and (iii) we always have $m = 2$, then a bound of $\frac{2}{3}C(P)$ can be obtained.

From Theorem 2 it follows that in order to compute a cut segment as desired, it suffices to look at the vertex-edge visible pairs of the polygon (a vertex-edge visible pair is a vertex and an edge that can be connected by an axis-parallel line segment that lies entirely inside the polygon, i.e., a cut segment). The next lemma shows that these pairs (whose total number is $O(n)$, even in a non-rectilinear polygon, see [19]) can be computed in linear time if the polygon does not contain holes.

Lemma 1 *All vertex-edge visible pairs of a simple rectilinear polygon P on n vertices without holes can be computed in time $O(n)$.*

Proof: The computation of the pairs will consist of three steps.

First P is partitioned into a number of histograms. A histogram, sometimes called a Manhattan polygon, is a rectilinear polygon H that has one distinguished edge, the *base* of H, whose length is equal to the sum of the lengths of all other edges parallel to this base. In [12] it is shown that every rectilinear polygon without holes can be partitioned into a number of histograms $H_1, \ldots, H_m$ in time $O(n)$ such

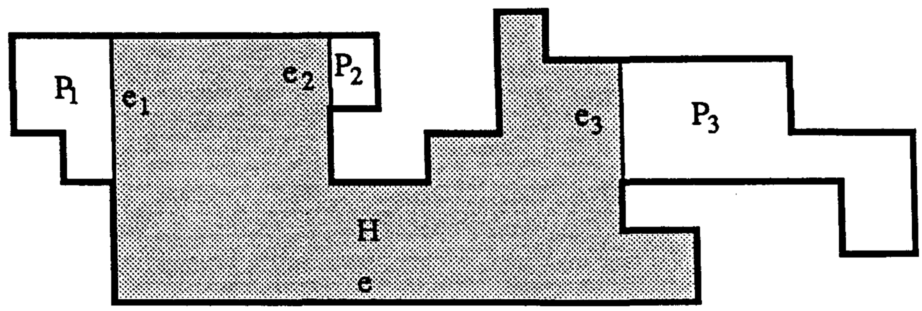


Figure 3: The partitioning of a rectilinear polygon into histograms. The non-bold segments are entrances.

that the total number of vertices of these histograms is $O(n)$. This is done by choosing an arbitrary edge e and computing the maximal histogram H lying within P with e as base. The set of edges that partition P, $HIST(P,e)$, is recursively defined as follows: If $H = P$ then $HIST(P,e) = \emptyset$. Otherwise let $H, P_1, \ldots, P_k$ be the set of polygons into which P is partitioned by H and let e_i be the segment where H touches P_i (we call this segment the *entrance* between H and P_i), then $HIST(P,e) = \bigcup_{1 \leq i \leq k}(\{e_i\} \cup HIST(P_i, e_i))$. In Figure 3 an example of a partitioning is given.

When we have partitioned P into histograms $H_1, \ldots, H_m$ as described above the vertex-edge visible pairs in each H_i are determined. The vertices that see the base of H_i are exactly the reflex vertices of H_i. The other visible pairs, that can be connected by a segment parallel to the base, can be computed as follows. Assume w.l.o.g. that the base is horizontal. Now start at the left endpoint of the base and walk upwardly along the boundary of H_i, i.e., continue walking as long as the edges are directed upward or to the right. Meanwhile push the encountered vertices on a stack. Then start walking again, this time as long as the edges are directed downward or to the left. At every encountered vertex v_i, we pop all vertices from the stack that have y–coordinate greater than the y–coordinate of v_i. All these vertices see $\overline{v_{i-1}v_i}$ (except if they are the left endpoint of a rightward directed edge; this, however, can easily be tested) and v_i sees $\overline{v_{j-1}v_j}$, where v_j is the vertex popped last. Then start walking up again, pushing the encountered vertices on the stack, etc. It is not hard to prove that this way the correct pairs are found in time $O(\sharp\text{vertices of } H_i)$. Notice that vertices that see some edge are found ordered along this edge.

Now that we have computed the visible pairs in every H_i, we only have to find the right visible edges for vertices for which we have computed that they see an edge that is an entrance. Since we have for each entrance $e_{i,j}$ between histograms H_i and H_j for both histograms a sorted list of the vertices that see $e_{i,j}$, these edges can easily be found by walking simultaneously along both lists; the edge of H_j visible from some vertex v_k in H_i is namely equal to $\overline{v_l v_{l+1}}$ (or $\overline{v_{l-1}v_l}$, depending on whether the list is ordered clockwise or counterclockwise), where v_l is the vertex in the list of H_j that has been encountered just before v_k was encountered. Note that the situation that the newly found edge is also an entrance can only occur once for every vertex, so this does not impose any significant problems. Computing the edges visible from the vertices that see an entrance thus takes $O(n)$ time in total.

We see that that every step of the algorithm takes only linear time, which proves the lemma. □

Theorem 3 *Let P be a simple rectilinear polygon on n vertices without holes. Then a cut segment that cuts P into two polygons having weight $\leq \frac{3}{4}C(P)$ can be computed in time $O(n)$.*

Proof: From the above lemma it follows that the $O(n)$ segments that are sufficient to consider can be computed in linear time. Furthermore a segment can be tested in constant time after $O(n)$ preprocessing (see [2]). The time bound follows. □

3 The Query Problem

The problem that we will consider is stated as follows: Store a simple rectilinear polygon P on n vertices (without holes) in a data structure such that, given two query points s and t in P (the *source* and the *target*), a rectilinear link distance shortest path between s and t can be computed efficiently. First both source and target are assumed to be vertices of the polygon. Then the solution is extended to handle arbitrary points inside the polygon as query points. Finally we show that the path that is computed by our algorithm not only has a minimal number of segments, but that it is optimal in the L_1-metric as well.

3.1 Vertices as query points

Because we do not want to use quadratic storage, we cannot store for each vertex information about the direction in which to leave to every other vertex. Therefore we take another approach. Let e be a segment that cuts P into two subpolygons P_1 and P_2. For all source-destination pairs with the source lying in another subpolygon than the target, the path must cross e and thus the direction in which to leave is towards e. For all other pairs we have reduced the problem to finding a shortest path in a subpolygon of P that can be treated in the same way. The Rectilinear Polygon Cutting Theorem of the previous section guarantees us (assign each vertex weight 1) that e can be chosen such that this resulting polygon has $\leq \frac{3}{4}n+2$ vertices, n being the number of vertices of P. Thus P is stored in a binary tree T that can recursively be described as follows: If P is a rectangle then T is a leaf. Otherwise, let e be a segment that cuts P into two polygons P_1 and P_2 both having $\leq \frac{3}{4}n + 2$ vertices. Now T consists of one subtree representing P_1 and one subtree representing

P_2. Thus each node δ in T represents a subpolygon P_δ of P and P_δ is cut into $P_{lson(\delta)}$ and $P_{rson(\delta)}$ by a segment e_δ. Since e_δ cuts P_δ in a balanced way, the depth of the tree is $O(\log n)$. The search path in T of a vertex v naturally follows those nodes δ where $v \in P_\delta$. Thus it goes to the left at nodes δ such that $v \in P_{lson(\delta)}$ and to the right if $v \in P_{rson(\delta)}$. The path ends when a leaf is reached or when v is incident upon e_δ.

Given a source s and a destination t, we proceed as follows. Let γ_s and γ_t be the leaves (or nodes) where the search paths to s resp. t end. If both paths end in the same leaf or in the same node then a shortest path from s to t is trivial to compute. Otherwise let δ^* be the node where the paths split, i.e., the lowest node δ such that both s and t are in P_δ. Observe that in fact δ^* is the lowest common ancestor of γ_s and γ_t. Now we know that any path from s to t must cross e_{δ^*}. Hence, we store at every node δ of T for every vertex of P_δ information about a shortest path to e_δ. Before we give a lemma that enables us to compute a shortest path from s to t from shortest paths from s and t to e_{δ^*}, we need some notation. For a cut segment e and a vertex v of P, let $e(v,d)$ be the part of e that can be reached from v with a path π of length d such that the last segment of π is perpendicular to e. Furthermore let the (rectilinear link) distance from a vertex v to a segment e be defined as $d(v,e) = \min\{d(v,q) | q \in e\}$, the distance from v to (one of) the closest point(s) on e. Now the lemma that we need can be stated.

Lemma 2 *Let e cut P into two subpolygons such that s and t lie in different subpolygons and let $d(s,e) = d_s$ and $d(t,e) = d_t$. Then we have*

$$d(s,t) = d_s + d_t + \Delta$$

$$\text{where } \Delta = \begin{cases} -1 & \text{if } e(s,d_s) \cap e(t,d_t) \neq \emptyset \\ 0 & \text{if } e(s,d_s) \cap e(t,d_t) = \emptyset \ \wedge \\ & \quad (\ e(s,d_s+1) \cap e(t,d_t) \neq \emptyset \ \vee \ e(s,d_s) \cap e(t,d_t+1) \neq \emptyset\ ) \\ +1 & \text{otherwise} \end{cases}$$

Proof: If $e(s,d_s) \cap e(t,d_t) \neq \emptyset$, then paths from s and t of lengths d_s and d_t respectively exist that reach e at the same place. Thus the two segments incident upon e now form one segment and the resulting path has length $d_s + d_t - 1$. Clearly a path that is shorter cannot exist; it would contradict either $d(s,e) = d_s$ or $d(t,e) = d_t$.

If $e(s,d_s) \cap e(t,d_t) = \emptyset$, but $e(s,d_s+1) \cap e(t,d_t) \neq \emptyset$ (or $e(s,d_s) \cap e(t,d_t+1) \neq \emptyset$), then we can take paths of lengths $d_s + 1$ and d_t (or d_s and $d_t + 1$) that meet on e resulting in a path of length $(d_s + 1) + d_t - 1 = d_s + d_t$. Again it is easily seen that no shorter path can exist under the given conditions.

Finally, if neither of the conditions for $\Delta = +1$ and $\Delta = 0$ is true, then we can always take paths of lengths d_s and d_t and join them by a segment on e, thus giving a path of length $d_s + d_t + 1$. Since the conditions for $\Delta = +1$ and $\Delta = 0$ are not

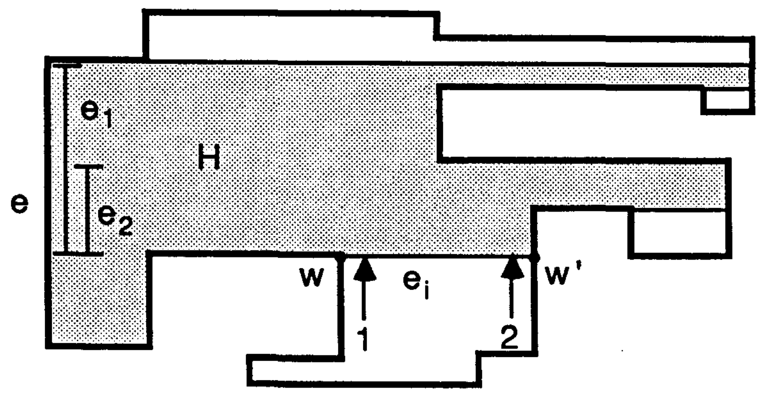


Figure 4: e_1 is the part of e that can be reached when H is entered at point 1, e_2 is the part that can be reached when H is entered a point 2. e_i is w–oriented.

only sufficient, but also necessary, a shorter path cannot exist. □

So what we now want is to compute the part of a cut segment reachable by a shortest (or almost shortest) path from a vertex in one of the subpolygons, i.e., we want to compute the reachable part of an edge of the subpolygon. We are thus left with the following problem. Given a polygon P and an edge e of P, compute $e(v, d_v)$ and $e(v, d_v + 1)$ (where $d_v = d(v, e)$) for every vertex v of P. To this end we prove that for any vertex v at distance $d_v > 2$ from e a vertex v_{next} at distance $d_v - 1$ from e (and at distance 2 or 1 from v) exists such that any point on $e(v, d_v)$ can be (optimally) reached via v_{next}.

Lemma 3 *Let v be a vertex of P with $d(v, e) = d_v > 2$. Then a vertex v_{next} of P exists such that $d(v_{next}, e) = d_{v_{next}} = d_v - 1$ and $e(v_{next}, d_{v_{next}}) = e(v, d_v)$. Moreover, every point on $e(v, d_v)$ can be reached from v by a path $\pi = l_1 l_2 \cdots l_{d_v}$ with $v_{next} \in l_2$.*

Proof: Again we will use the partitioning of P into histograms (with e as starting edge) as described in the proof of Lemma 1. Recall that H was the maximal histogram lying within P with e as base and that $P_1, \ldots, P_k$ were the induced subpolygons. The edges e_i where P_i touched H were called entrances and they were the starting edges for the partitioning of the P_i's into histograms. Observe that $d(v, e) = 1$ for $v \in H$ and $d(v, e) = d(v, e_i) + 1$ for $v \in P_i$ (and $v \notin H$).

Let $v \in P_i$ with $d_v > 2$ and consider a path from v to e. This path must cross e_i. Note that it is always profitable to cross e_i as close (in the ordinary, Euclidean, sense) to e as possible. The part of e that can be reached is namely equal to the segment running from e_i to the edge of H visible from the point on e_i where H is entered and, since H is a histogram with base e, these segments grow as this point comes closer to e (refer to Figure 4). Therefore we define an entrance $e_i = \overline{ww'}$ to histogram H with base e to be *w–oriented* if w is closer to e than w'. If w' is closer then e_i is w'–oriented.

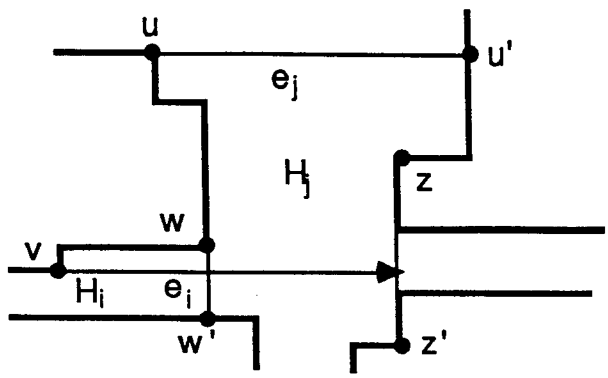


Figure 5: Illustration of the proof of Lemma 3.

Let $\{H_1, \ldots, H_m\}$ be the set of histograms that results from the partitioning scheme described above. (With a slight abuse of notation, we will still use H to denote the unique histogram with e as base, when convenient.) Suppose $v \in H_i$ where v is not incident upon the (w–oriented) base $e_i = \overline{ww'}$ of H_i. Let e_i be the entrance to histogram H_j with base $e_j = \overline{uu'}$ and suppose w.l.o.g. that e_j is horizontal with $u_x \leq w_x \leq u'_x$ (refer to Figure 5). By definition of orientation we should, if e_j is u–oriented, cross e_j as close to u as possible. Hence the path from v should go through w and w is a choice for v_{next} that satifies the demands. On the other hand, if e_j is u'–oriented then z (where $\overline{zz'}$ is the edge of H_j that is visible from v inside $H_i \cup H_j$, see Figure 5) is a choice for v_{next} meeting the requirements. □

The lemma above readily gives us a way to compute $e(v, d_v)$ for all vertices v of P in an efficient way. First the vertex-edge visible pairs that we need are determined. These are vertex-edge visible pairs inside some $H_i \cup H_j$ for vertices of H_i, with H_i and H_j as in the proof of the lemma. Using the same approach as for determining all vertex-edge visible pairs in P (see the proof of Lemma 1) this can be done in $O(n)$ time. (In fact the pairs we need were found as intermediate results in the algorithm given there.) Once this has been done we proceed as follows. For vertices v with $d_v = 1$ (thus $v \in H$), we trivially have $e(v, d_v) = e(v, 1) = [v_y : v_y]$ (or $[v_x : v_x]$ if e is horizontal). For vertices $v \in H_i$ at distance 2 from e, $e(v, 2)$ can easily be found using the edge of H that is visible from v inside $H_i \cup H$. $e(v, 2)$ is then equal to the segment from the entrance between H_i and H to the edge of H visible from v. For vertices at distance > 2 we can apply the lemma above. Since the orientation of the entrances can easily be determined during the process in constant time per entrance and the visible edges that are needed are precomputed, the vertex v_{next} can be found in $O(1)$ time per vertex. Thus the segments $e(v, d_v)$ for all vertices can be computed in linear time in total.

Once this has been done, the computation of $e(v, d_v + 1)$ (which is useless if

$e(v, d_v+1) \subseteq e(v, d_v)$, as is clear from Lemma 2) is not hard, as the following lemma shows.

Lemma 4 *For every vertex v of P with $d_v \geq 2$, either there exists a vertex v_{next2} such that $e(v, d_v+1) = e(v_{next2}, d_{v_{next2}})$ or $e(v, d_v+1) \subseteq e(v, d_v)$.*

Proof: By definition, $e(v, d)$ is the part of e that can be reached with a path $\pi = l_1 l_2 \cdots l_d$ such that l_d is perpendicular to e. Hence the difference between the lengths of two paths to e that leave a vertex v in the same direction, i.e. both having a horizontal first segment or both having a vertical first segment, will always be even. Because for a vertex $v \in H_i$ the first segment of a shortest path to e leaves in the direction of the base e_i of H_i, the first segment of any path of length d_v+1 must be parallel to e_i. We let this first segment be the largest segment inside H_i directed as $\overrightarrow{u'u}$, where $e_i = \overline{uu'}$ is u–oriented. This way we might be able to reach a larger part of e. If this first segment has length zero, then $e(v, d_v+1) \subseteq e(v, d_v)$. Otherwise let $\overline{ww'}$ be the edge of H_j where the first segment ends. Then, assuming w is closer to e_i than w', w is clearly a vertex such that $e(v, d_v+1) = e(w, d_w)$. $\square$

Using vertex-edge visible pairs, these vertices v_{next2} (as well as $e(v, 2)$ for vertices v with $d_v = 1$) can be computed in linear time. (If $e(v, d_v+1) \subseteq e(v, d_v)$, we let $v_{next2} = v$. The reader easily verifies that this does not influence the correctness of the algorithm.) Now we are ready to complete the description of our data structure for the Query Problem inside a polygon. We have:

- A binary tree T representing P as follows: If P is a rectangle then T is a leaf. Otherwise, let $e_{root(T)}$ be a segment cutting P into two subpolygons P_1 and P_2, both having $\leq \frac{3}{4}n+2$ vertices. Now T consists of two subtrees, representing P_1 and P_2 respectively. At every node $\delta \in T$ we store for every vertex v of P_δ (the subpolygon represented by δ) a record containing the following information: $d_v = d(v, e_\delta)$, $e_\delta(v, d_v)$ and $e_\delta(v, d_v+1)$ as well as pointers to (the records stored at δ for) v_{next} and v_{next2} (assuming $d_v > 2$).

- For every vertex $v_i \in P$ there is a pointer PTR_i to the node or leaf $\delta_i \in T$ where the search path to v_i ends and an array ANC_i. $ANC_i[lev]$ points to the record that is stored for v_i at the ancestor of δ_i at level lev.

A shortest path from $s = v_i$ to $t = v_j$ is now found as follows:

1. Follow PTR_i and PTR_j to obtain δ_i and δ_j. If $\delta_i = \delta_j$, then a shortest path from v_i to v_j is trivial to compute. Otherwise compute δ^*, the lowest common ancestor of δ_i and δ_j.

2. Follow $ANC_i[lev(\delta^*)]$ to obtain $d_{v_i} = d(v_i, e_{\delta^*})$, $e_{\delta^*}(v_i, d_{v_i})$ and $e_{\delta^*}(v_i, d_{v_i}+1)$. Do the same for v_j. Compute Δ according to Lemma 2 and decide whether the next vertex on the path from v_i (v_j) to e_{δ^*} is $(v_i)_{next}$ or $(v_i)_{next2}$ ($(v_j)_{next}$

or $(v_j)_{next2}$). From this vertex on, follow v_{next} pointers towards e_{δ^*} until a vertex at distance 2 from e_{δ^*} is reached. (The details for the case $d_{v_i} \leq 2$ are straightforward.)

3. Glue the paths together.

This leads to:

Lemma 5 *A data structure exists in which the rectilinear link distance between two query vertices of a rectilinear polygon P on n vertices can be computed in time $O(1)$ and a shortest path in time $O(1+l)$, where l is the length of the path. The structure uses $O(n \log n)$ storage and can be built in time $O(n \log n)$.*

Proof: The correctness of the algorithm follows immediatly from Lemma's 2, 3 and 4. (See section 3.3 for more details about the glueing operation.)

The query time is clear if δ^*, the lowest common ancestor of δ_i and δ_j can be computed in constant time. After $O(n)$ preprocessing, this is indeed possible ([7]). (Note that every time a v_{next} pointer is followed, we find a segment of the path.)

We will now prove the building time, from which the bound on the storage readily follows. Consider the following recursive building algorithm. Compute a cut segment $e = e_{root(T)}$ of P that cuts P into P_1 and P_2, with $|P_1|, |P_2| \leq \frac{3}{4}n + 2$. According to Theorem 3 this takes $O(n)$ time. Then compute $e(v_i, d_{v_i})$, $e(v_i, d_{v_i}+1)$ and the vertices $(v_i)_{next}$ and $(v_i)_{next2}$ for every vertex v_i of P_1 and P_2. As was shown above, this can also be done in linear time. Store this information and store a pointer to it in $ANC_i[lev_{curr}]$, where lev_{curr} denotes the level of T we are currently considering.

Next, build the two subtrees corresponding to P_1 and P_2. Now $T(n)$, the building time so far, is equal to $T(|P_1|) + T(|P_2|) + O(n)$ and, having $|P_1|, |P_2| \leq \frac{3}{4}n + 2$, this is $O(n \log n)$. The nodes δ_i where the search paths to v_i ends and thus the pointers PTR_i are easily computed during the building process. As we already noted, the additional information that is needed for the lowest common ancestor algorithm takes only linear time to compute. □

Remark: The $O(1)$-algorithm of [7] for finding lowest common ancestors runs on a random access machine ([1]). On a pointer machine ([16, 18]) finding lowest common ancestors requires $\Omega(\log \log n)$ time per query (see [7]). This bound has been achieved ([11]) and, hence, our algorithm for computing shortest paths runs in time $O(\log \log n + l)$ on a pointer machine. (Notice that the arrays ANC_i that are used have length $O(\log n)$ and can thus be replaced by search trees with $O(\log \log n)$ search time.)

3.2 Arbitrary points as query points

In this section it is shown how the data structure devised for handling vertices as query points, as described above, can also be used to solve the general problem where the query points are not confined to the vertices of P, but are arbitrary points in P.

Consider the subdivision of P induced by the cut segments once more. First it has to be determined which regions (rectangles) contain the two query points, that is, the leaves γ_s andγ_t where the paths to s and t end. Using the optimal point location method of Edelsbrunner et al. ([6]), this can be done in time $O(\log n)$ with a structure that uses $O(n)$ storage. Observe that, since we can compute the vertex-edge visisble pairs in linear time, we can turn P into a monotone subdivision in linear time and, hence, the point location structure can be built in linear time. Again a shortest path is trivial to compute if the two query points s and t are in the same rectangle. If they are not, then the path between s and t again crosses e_{δ^*}, with δ^* the lowest common ancestor of γ_s and γ_t. It is even true that the three key lemmas to solve the problem, Lemma 2, 3 and 4, are still valid. The vertices v_{next} and v_{next2} of Lemmas 3 and 4 respectively, that can be precomputed if the query points are vertices, now have to be determined as a part of the query. Thus we need a data structure to solve the following problem: given an axis-parallel query ray $\vec{r}$ starting at an arbitrary point q in some rectilinear polygon, compute the first edge that is hit by $\vec{r}$. This polygon is $H_i \cup H_j$, according to the proofs of the considered lemmas, with $q \in H_i, q \notin H_j$ and H_j adjacent to H_i and closer to e (see Figure 5).

Note that H_i and H_j as well as the direction of $\vec{r}$ can again be determined using point location techniques. Thus at every node in T, we need such a ray-shooting structure for every histogram H_i. Now the required edge can be determined with two ray-shooting queries: first a query in H_i, then — if the first edge hit is H_i's base — a query from the intersection point of $\vec{r}$ with the base in the same direction in H_j. We could use the structure devised by Chazelle and Guibas ([3]) for the ray-shooting problem, but this would be a rather brute approach to our much more restricted problem. Moreover, the preprocessing of their structure is $O(n \log n)$ which would result in a preprocessing time of $O(n \log^2 n)$ for our total structure. Fortunately we can obtain a solution to our problem that requires only linear preprocessing.

Let H be a histogram with horizontal base b. A query with a vertical ray is easily solved by a binary search on the x-coordinates of the vertices of H. Queries with a horizontal ray are solved using a locus approach: from every reflex vertex of H we add a horizontal edge to the edge that is (horizontally) visible from this vertex. Note that these extra edges can be computed in linear time by Lemma 1. Now the answer to a query with a horizontal ray are contant in each resulting region (depending on whether the ray is directed to the right or to the left, of course). Observe that the subdivision is monotone and, hence, the region which contains the starting point of the query ray can be determined in $O(\log n)$ time with a structure using $O(n)$ space and preprocessing ([6]). Thus the extra storage and preprocessing that is needed at some node δ is $O(|P_\delta|)$, since $\sum |H_i| = O(|P_\delta|)$ for the partitioning of P_δ into histograms H_i as used. Because the query time of our ray-shooting structure is $O(\log n)$, the total query time becomes $O(\log n + l)$. We conclude:

Lemma 6 *A data structure exists in which the rectilinear link distance between two query points in a rectilinear polygon can be computed in time $O(\log n)$ and a shortest* *path between the two points in time $O(\log n + l)$, where l is the length of the path. The structure uses $O(n \log n)$ storage and can be built in time $O(n \log n)$.*

3.3 Obtaining L_1-optimal paths

We will now investigate the relation between the rectilinear link distance metric and the L_1-metric. In the L_1-metric, the length of a line segment $\overline{pq}$ is equal to $|p_x - q_x| + |p_y - q_y|$. The length of a path π in the L_1-metric, denoted as $length_{L_1}(\pi)$, is naturally defined as the sum of the lengths of the segments π consists of. Hence, the length of a rectilinear path in the L_1-metric is equal to its Euclidean length. We will show that the paths computed by the query algorithm of the previous section are not only optimal in the rectilinear link distance metric, but also in the L_1-metric, provided that the glueing operation is performed correctly. Notice that optimality in one of the metrics does not automatically imply optimality in the other metric and that the fact that between any two points in a polygon there is a path that is optimal in both metrics is no longer true if we allow the polygon to have holes.

To obtain L_1-optimal paths we have to perform the glueing operation in a special way. Let e be the cut segment through which the path between the two query points s and t should pass. (If there is no such segment, i.e., s and t are in the same rectangle, then a rectilinear link optimal path is evidently L_1-optimal.) Assume that $d(s, e)$ and $d(t, e)$ are both ≥ 2. (The case where one or both of these distances are < 2 is left as an exercise to the reader.) If the paths from s and t to e are denoted by π_s and π_t respectively, we have the following information available for the glueing operation: a vertex v_s of P on the one but last segment of π_s, a subsegment $e(s) = [b_s : e_s]$ of e reachable by π_s and a point v_t and a segment $e(t) = [b_t : e_t]$ defined analogously. Assume w.l.o.g. that e is vertical and that $e_s \geq e_t$. The paths are now glued together as follows: If $e(s) \cap e(t) = \emptyset$ then let π_s reach e at b_s, let π_t reach e at e_t and add the segment on e from b_s to e_t to the path (Figure 6(i)). If $e(s) \cap e(t) \neq \emptyset$ then connect π_s and π_t at point $\max(b_s, b_t)$ on e if both paths 'come from below', i.e., $(v_s)_y \leq b_s$ and $(v_t)_y \leq b_t$ (Figure 6(ii)) and connect the paths at point $\min(e_t, e_s)$ otherwise (Figure 6(iii)). To prove that our algorithm with this glueing operation yields a path that is optimal in the L_1 metric, we use the following:

Lemma 7 *Let π and π' be two paths from x to y that intersect only in x and y. Suppose π contains no two consecutive convex vertices (not counting x and y), where a vertex of π is convex if its interior angle in R, the region enclosed by π and π', is convex. Then $length_{L_1}(\pi) \leq length_{L_1}(\pi')$.*

The proof of this lemma is straightforward and therefore omitted.

Lemma 8 *The query algorithm given in the previous section with the glueing operation as described above yields a path that is not only optimal in the rectilinear link distance metric, but also in the L_1 metric.*

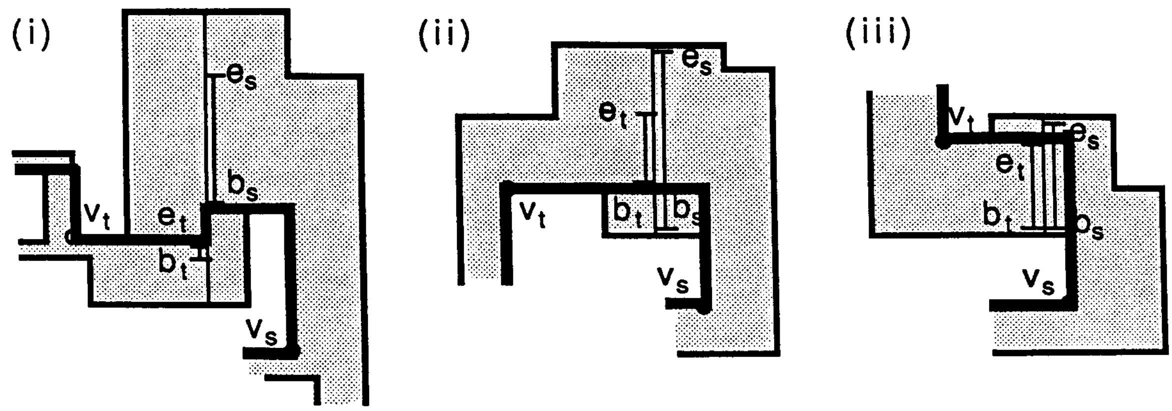


Figure 6: The three different cases for the glueing operation.

Proof: Let π be the path found by the algorithm and let π' be an L_1-optimal path. Let x and y be two consecutive intersection points between π and π' and denote the portions of π and π' between x and y by ρ resp. ρ'. We will show that ρ contains no two consecutive convex vertices (in R, the region enclosed by ρ and ρ'). By Lemma 7, this will prove the lemma.

Let $x, r_1 \cdots r_k, y$ and $x, r'_1 \cdots r'_l, y$ be the enumeration of the vertices on ρ and ρ' respectively. Suppose for a contradiction that ρ contains two consecutive convex vertices r_i, r_{i+1}. Recall that every segment of a path to the cut segment e crosses an entrance (except the last segment, which is involved in the glueing operation). Let $\overline{r_i r_{i+1}}$ cross the w-oriented entrance $\overline{ww'}$, then it follows from the construction of π that $\overline{r_i r_{i+1}}$ crosses $\overline{ww'}$ as close to w as possible, i.e. there must be an edge of P on the same side of $\overline{r_i r_{i+1}}$ as w. Since $\overrightarrow{r_{i+1} r_{i+2}}$ is directed as $\overrightarrow{w'w}$ (because $\overline{ww'}$ is w-oriented) and r_{i+1} is convex (by assumption), this edge lies in R. This contradicts the fact that ρ and ρ' are valid non-intersecting paths in P. (Here we use the fact that P is hole-free.) See Figure 7.

For the segments involved in the glueing operation, a similar argument can be given. (Note that the extra segment added in Figure 6 is always incident upon exactly one convex vertex.) □

Summarizing the results of this section, we have:

Theorem 4 *A data structure exists in which the rectilinear link distance between two query points in a rectilinear polygon can be computed in time $O(\log n)$. A path between the two points that is optimal in both the rectilinear link metric and the L_1-metric can be found in time $O(\log n + l)$, where l is the (rectilinear link) length of the path. The structure uses $O(n \log n)$ storage and can be built in time $O(n \log n)$. If the query points are vertices of the polygon then the query times become $O(1)$ and $O(1 + l)$ respectively.*

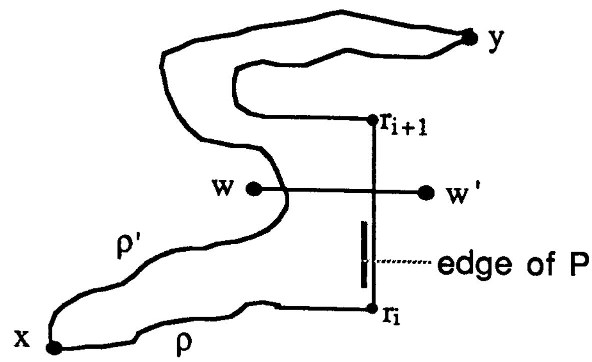


Figure 7: ρ cannot contain two convex vertices.

4 The Diameter Problem

As a second problem concerning rectilinear link distance in rectilinear polygons, we treat the diameter problem. Thus we want to compute the diameter of a rectilinear polygon P on n vertices, without holes, in the rectilinear link distance metric. This is denoted $Dia(P)$ and is defined as $Dia(P) = \max\{d(p,q)|p,q \in P\}$. It is readily seen that there will always be a pair of vertices at this maximal distance. It is even true that there will always be a pair of convex vertices at distance $Dia(P)$, so that we can restrict ourselves to the convex vertices of P. (This is also true in the 'ordinary' link distance metric, see [10].)

4.1 Computing the exact diameter

The exact diameter of a rectilinear polygon P is computed with the divide-and-conquer algorithm given below:

1. If P is a rectangle, then $Dia(P) = 2$, otherwise go to step 2.
2. Compute a cut segment e of P that cuts P into two subpolygons P_1 and P_2, such that $|P_1|, |P_2| \le \frac{3}{4}n + 2$.
3. Compute $Dia(P_1)$ and $Dia(P_2)$ recursively.
4. Compute $M = \max\{d(v,w)|v \in P_1, w \in P_2\}$.
5. Let $Dia(P) := \max(Dia(P_1), Dia(P_2), M)$.

The correctness of this algorithm is obvious. By the Rectilinear Polygon Cutting Theorem, step 2 can be performed in $O(n)$ time. Now if $T(n)$ is the time that is spent for the total algorithm and $f(n)$ the time for step 4, then for $T(n)$ the following recurrence holds:

$$T(n) = T(m) + T(n-m) + f(n) + O(n), \qquad \frac{1}{4}n - 2 \le m \le \frac{3}{4}n + 2 \quad (1)$$

In the remainder of this section it is shown how $M = \max\{d(v,w) | v \in P_1, w \in P_2\}$ can be computed in linear time, leading according to (1) to an overall running time of $O(n \log n)$.

Let P be a rectilinear polygon on n vertices and let e cut P into two subpolygons P_1 and P_2 and let $d_1 = \max\{d(v,e) | v \in P_1\}$ and $d_2 = \max\{d(w,e) | w \in P_2\}$. Furthermore define $P_i^d = \{u \text{ is a vertex of } P_i | d(u,e) = d\}$ $(i = 1,2)$ to be subset of vertices of P_i at distance d from e. From Lemma 2 it immediatly follows that $M = d_1 + d_2 + \Delta$, with $\Delta \in \{+1, 0, -1\}$. E.g., when there are vertices $v \in P_1^{d_1}, w \in P_2^{d_2}$ with $(e(v,d_v) \cup e(v,d_v+1)) \cap e(w,d_w) = \emptyset$ and $e(v,d_v) \cap (e(w,d_w) \cup e(w,d_w+1)) = \emptyset$, then $d(v,w) = d_v + d_w + 1 = d_1 + d_2 + 1$ and $M = d_1 + d_2 + 1$ if and only if there is such a pair. To be more precise, we have:

Lemma 9 $M \;=\; d_1 + d_2 + \Delta$

$$\text{where } \Delta = \begin{cases} +1 & \text{if there is a pair } v \in P_1^{d_1}, w \in P_2^{d_2} \text{ such that:} \\ & \quad (e(v,d_v) \cup e(v,d_v+1)) \cap e(w,d_w) = \emptyset \;\wedge \\ & \quad e(v,d_v) \cap (e(w,d_w) \cup e(w,d_w+1)) = \emptyset \\ -1 & \text{if for all pairs } v \in P_1^{d_1}, w \in P_2^{d_2}: \\ & \quad e(v,d_v) \cap e(w,d_w) \neq \emptyset \qquad \text{and} \\ & \text{for all pairs } v \in P_1^{d_1}, w \in P_2^{d_2-1}: \\ & \quad (e(v,d_v) \cup e(v,d_v+1)) \cap e(w,d_w) \neq \emptyset \;\vee \\ & \quad e(v,d_v) \cap (e(w,d_w) \cup e(w,d_w+1)) \neq \emptyset \qquad \text{and} \\ & \text{for all pairs } v \in P_1^{d_1-1}, w \in P_2^{d_2}: \\ & \quad (e(v,d_v) \cup e(v,d_v+1)) \cap e(w,d_w) \neq \emptyset \;\vee \\ & \quad e(v,d_v) \cap (e(w,d_w) \cup e(w,d_w+1)) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Note that all segments needed for the evaluation of Δ can be computed in linear time according to the previous section. Before we describe how the conditions that determine the value of Δ can be evaluated efficiently, it is convenient to introduce some more notation. Suppose $e(u,d_u) = [x_1 : x_2]$ and $e(u,d_u) \cup e(u,d_u+1) = [y_1 : y_2]$. If $d_u > 1$ (we omit the details for the case $u \in H$ as they are straightforward) then paths from u to e of length d_u as well as paths of length $d_u + 1$ must enter H, the maximal histogram inside P_1 (or P_2, depending on where u lies), through the same entrance and, hence, we either have $x_1 = y_1 < x_2 \leq y_2$, or $x_2 = y_2 > x_1 \geq y_1$. Therefore we split the set of (convex) vertices of P_1 and of P_2 into subsets V and $\tilde{V}$ and subsets W and $\tilde{W}$ respectively, according to the distinguished cases. Thus $u \in V$ iff u is a vertex of P_1 such that $x_1 = y_1 < x_2 \leq y_2$ and $u \in \tilde{V}$ iff u is a vertex of P_1 such that $x_2 = y_2 > x_1 \geq y_1$; the vertices of P_2 are similarly split into W and $\tilde{W}$. In other words, $u \in V \cup W$ iff a path from u enters H in an upward (or rightward, if the base of H is horizontal) direction. Now for a vertex u of P, we define u_1, u_2 and u_3 to be such that:

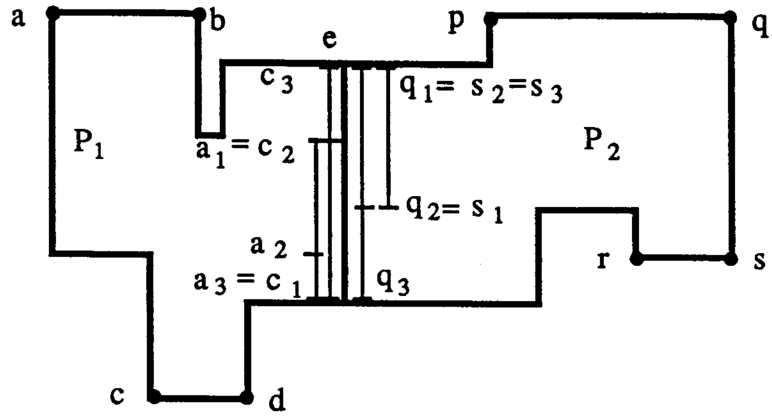


Figure 8: $a, b \in \tilde{V}$, $c, d \in V$, $p, q \in \tilde{W}$ and $r, s \in W$.

If $u \in V \cup W$ then $e(u, d_u) = [u_1 : u_2]$ and $e(u, d_u) \cup e(u, d_u + 1) = [u_1 : u_3]$.
If $u \in \tilde{V} \cup \tilde{W}$ then $e(u, d_u) = [u_2 : u_1]$ and $e(u, d_u) \cup e(u, d_u + 1) = [u_3 : u_1]$.

See figure 8 for an illustration of these definitions. Below we show how the conditions of the lemma can be evaluated for the cases $v \in V, w \in W$ and $v \in V, w \in \tilde{W}$, i.e., we show how to compute $M_{V,W} = \max\{d(v, w) | v \in V, w \in W\}$ and $M_{V,\tilde{W}} = \max\{d(v, w) | v \in V, w \in \tilde{W}\}$. The computation of $M_{\tilde{V},W}$ and $M_{\tilde{V},\tilde{W}}$ is done in a similar way. After having computed these values it remains to observe that $M = \max(M_{V,W}, M_{V,\tilde{W}}, M_{\tilde{V},W}, M_{\tilde{V},\tilde{W}})$.

We start with the case where $v \in V, w \in W$. The conditions for the various values of Δ can now be expressed as follows (V^d and W^d denote the subset of points of V and W respectively that are at distance d from e):

$$\Delta = \begin{cases} +1 & \text{if there is a pair } v \in V^{d_1}, w \in W^{d_2} \text{ such that: } v_1 > w_3 \ \vee \ v_3 < w_1 \\ -1 & \text{if for all pairs } v \in V^{d_1}, w \in W^{d_2} : \ v_1 \le w_2 \ \wedge \ v_2 \ge w_1 \quad \text{and} \\ & \quad \text{for all pairs } v \in V^{d_1}, w \in W^{d_2 - 1} : \ v_1 \le w_3 \ \wedge \ v_3 \ge w_1 \quad \text{and} \\ & \quad \text{for all pairs } v \in V^{d_1 - 1}, w \in W^{d_2} : \ v_1 \le w_3 \ \wedge \ v_3 \ge w_1 \\ 0 & \text{otherwise} \end{cases}$$

The evaluation of the conditions for $\Delta = -1$ is easy now. For the first condition, for example, we just have to compute $M_1 = \max\{v_1 | v \in V^{d_1}\}$, $M_2 = \min\{w_2 | w \in W^{d_2}\}$, $M_3 = \min\{v_2 | v \in V^{d_1}\}$ and $M_4 = \max\{w_1 | w \in W^{d_2}\}$. Now the condition is equal to $M_1 \le M_2 \ \wedge \ M_3 \ge M_4$. The two other conditions for $\Delta = -1$ can be tested in the same way and, since the condition for $\Delta = +1$ could as well be stated as: "NOT(for all pairs $v \in V^{d_1}, w \in W^{d_2} : \ v_1 \le w_3 \ \wedge \ v_3 \ge w_1$)", also this condition can be tested in a simple way. Thus the evaluation of the conditions for Δ for the case $v \in V, w \in W$ can be done in $O(n)$ time.

Now consider the case $v \in V, w \in \tilde{W}$. This time the conditions of the lemma can be expressed as:

$$\Delta = \begin{cases} +1 & \text{if there is a pair } v \in V^{d_1}, w \in \tilde{W}^{d_2} \text{ such that:} \\ & \quad v_1 > w_1 \;\vee\; (v_2 < w_3 \;\wedge\; v_3 < w_2) \\ -1 & \text{if for all pairs } v \in V^{d_1}, w \in \tilde{W}^{d_2}: \\ & \quad v_1 \le w_1 \;\wedge\; v_2 \ge w_2 \qquad \text{and} \\ & \text{for all pairs } v \in V^{d_1}, w \in \tilde{W}^{d_2-1}: \\ & \quad v_1 \le w_1 \;\wedge\; (v_2 \ge w_3 \;\vee\; v_3 \ge w_2) \text{ and} \\ & \text{for all pairs } v \in V^{d_1-1}, w \in \tilde{W}^{d_2}: \\ & \quad v_1 \le w_1 \;\wedge\; (v_2 \ge w_3 \;\vee\; v_3 \ge w_2) \\ 0 & \text{otherwise} \end{cases}$$

The first condition for $\Delta = -1$ can be checked in the same way as above in linear time. The two other conditions for $\Delta = -1$ and the one for $\Delta = +1$ are again similar, so we will restrict ourselves to the evaluation of the condition for $\Delta = +1$. This condition is equal to "(there is a pair $v \in V^{d_1}, w \in \tilde{W}^{d_2} : v_1 > w_1$) or (there is a pair $v \in V^{d_1}, w \in \tilde{W}^{d_2} : v_2 < w_3 \wedge v_3 < w_2$)". The first part is again easy to check, so it remains to evaluate the second part:

$$\text{there is a pair } v \in V^{d_1}, w \in \tilde{W}^{d_2}: \quad v_2 < w_3 \quad \wedge \quad v_3 < w_2 \tag{2}$$

Now associate with each $v \in V^{d_1}$ a point $v^* = (v_2, v_3)$ in the plane and similarly with each $w \in \tilde{W}^{d_2}$ a point $w^* = (w_3, w_2)$. Call the resulting planar point sets V^* and W^*. Then, according to (2), we have to look for the existence of a pair v^*, w^* such that v^* is dominated by w^*. Using a scanline approach, this is easily tested: move the scanline from left to right over the plane and keep track of the lowest point in V^* encountered so far; if a point in W^* is encountered that lies above this point then we have found a dominance pair. If the points in V^* and W^* are sorted on their first coordinates then this takes linear time.

So we need a sorted list of the v_2 and w_3 values. Suppose that the cut segment e is vertical. Observe (see Figure 4) that the v_2 and w_3 values, which are endpoints of $e(v, d_v)$ and $e(w, d_w + 1)$, always coincide with the y–coordinate of some vertex of P. Moreover this vertex can be determined during the computation of $e(v, d_v)$ and $e(w, d_w + 1)$. After presorting the vertices of the polygon once, we can maintain a sorted list of the vertices during the recursive calls without significant overhead. This way it is possible to obtain a sorted list of the v_2 and w_3 values in linear time. Details are left to the reader.

Following the above approach leads to $f(n) = O(n)$ in (1), giving the following result:

Theorem 5 *The rectilinear link diameter of a rectilinear polygon on n vertices without holes can be computed in $O(n \log n)$ time.*

4.2 Computing an approximation of the diameter

Sometimes it may be sufficient to have a close approximation of the diameter instead of the exact diameter. Below it is shown that if we are willing to accept a small loss in accuracy a considerable gain in efficiency can be made: a simple recursive algorithm is given that computes an approximation D of the diameter, with $|D - Dia(P)| \leq 3$, in linear time. To this end we introduce an approximate distance function d':

$$d'(x,y) = \min\{length(\pi) | \pi = l_1 \cdots l_{d'} \text{ connects } x \text{ and } y,\ l_1 \text{ and } l_{d'} \text{ are vertical}\}$$

Note that

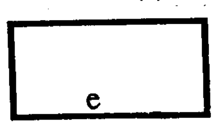
$$\begin{array}{ll} d(x,y) \leq d'(x,y) \leq d(x,y) + 2 & \text{and} \\ d(x,e) \leq d'(x,e) \leq d(x,e) + 1 & \text{for a horizontal cut segment } e \end{array}$$

if no two horizontal edges on the same line are at distance 1 from each other. This degeneracy can be avoided by a minor transformation of the polygon, which moves the troublesome edges slightly. If we have for each edge a sorted list of the visible vertices available (which can be obtained in linear time by Lemma 1), this transformation can be performed in linear time.

Procedure $MAXDIST$ takes as input a polygon P, a horizontal edge e of P whose two endpoints are convex vertices of P and a subsegment s of e. It computes an approximation D of $Dia(P)$. $MAXDIST$ works as follows. Imagine moving e into P until it hits a vertex. Now e cuts off a rectangle from P. Call the remaining polygon(s) P' (and P'') and the edge(s) that touches the rectangle e' (e''); see the figures in the detailed description given below. Obviously, either there are two points $x, y \in P'$ at distance $Dia(P)$ from each other, or one of the points lies on e. To be able to handle the latter case, we let $MAXDIST$ compute, besides D, the value $M = \max\{d'(x,s) | x \in P\}$, where $d'(x,s) = \min\{d'(x,y) | y \in s\}$. The reason for the the introduction of s is clear from, e.g., case (iii): to be able to compute the maximum distance to e, we need the maximum distance to a subsegment of e', not to e' itself. The algorithm distinguishes five cases according to the type of the first vertex encountered when e is moved upward. Note that the transformation of the polygon as described above ensures that no two cases occur simultaneously. In the algorithm, s' (s'') denotes the (orthogonal) projection of s onto e' (e'').

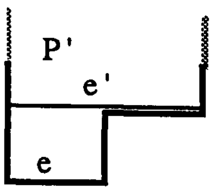
procedure $MAXDIST(P$:polygon, e:edge of P, s:subsegment of e, var M, D:integer);

case (i)



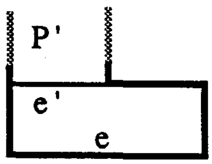
if $e = s$
then $M := 1$; $D := 3$
else $M := 3$; $D := 3$;

case (ii)



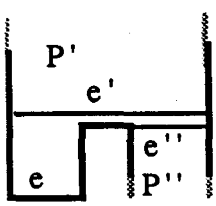
$MAXDIST(P', e', s', M', D')$,
$M := M'$;
$D := \max(D', M)$;

case (iii)



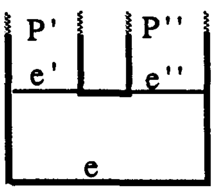
if $s' \cap e' = \varnothing$
then $MAXDIST(P', e', e', M', D')$; $M := M' + 2$
else $MAXDIST(P', e', s' \cap e', M', D')$; $M := M'$;
$D := \max(D', M)$;

case (iv)



$MAXDIST(P', e', s', M', D')$;
$MAXDIST(P'', e'', e'', M'', D'')$;
$M := \max(M', M'' + 2)$;
$D := \max(D', D'', M, M' + M'' - 2)$;

case (v)



if $s' \cap e' = \varnothing$
then $MAXDIST(P', e', e', M', D')$; $M_1 := M' + 2$
else $MAXDIST(P', e', s' \cap e', M', D')$; $M_1 := M'$;
if $s'' \cap e'' = \varnothing$
then $MAXDIST(P'', e'', e'', M'', D'')$; $M_2 := M'' + 2$
else $MAXDIST(P'', e'', s'' \cap e'', M'', D'')$; $M_2 := M''$;
$M := \max(M_1, M_2)$;
$D := \max(D', D'', M, M' + M'' - 1)$;

end $MAXDIST$;

Theorem 6 *An approximation D of the rectilinear link diameter of a rectilinear polygon on n vertices without holes, where $|D - Dia(P)| \leq 3$, can be computed in $O(n)$ time.*

Proof: Procedure $MAXDIST$ given above clearly works in linear time if we can decide in constant time which of the five cases occurs and determine the edges that play a role in that case. Using the sorted list of visible vertices for the two edges that are adjacent to e, which can be obtained for every edge in linear time as a preprocessing step (Lemma 1), this can indeed be done in constant time.

We will prove the correctness of the algorithm by induction on n, the number of vertices of P. (Observe that n is even and ≥ 4.) $n = 4$ (case (i)) is clearly handled correctly so suppose $n > 4$. The crucial observation here is that although an approximation of the diameter is computed, the value of M will be exact. This ensures that there will be no accumulation of errors in the recursive procedure. If we also keep in mind that M is the maximum *approximate* distance from any point in P to s, i.e., we only consider paths that leave s vertically, then it is easy to prove that the algorithm handles the four possible cases for $n > 4$ (note that these are indeed all possible cases, since the two endpoints of e are convex vertices) correctly.

case (ii): This a special case of (iv) (namely with $P'' = \emptyset$).

case (iii): If $s' \cap e' = \emptyset$ then any path from a point in P' must make two more turns after crossing e' to reach s since the last segment of the path must be vertical, so $M \geq M' + 2$. On the other hand, any path that reaches e' can be extended to reach s with two extra links, so $M \leq M' + 2$. Hence, $M = M' + 2$.

Now suppose $s' \cap e' \neq \emptyset$ and consider a shortest path $\pi = l_1 \cdots l_m$ from $x \in P'$ to s with l_1 and l_m vertical. Obviously if l_m crosses $s' \cap e'$ then the length of the subpath π' to s' is equal to the length of π. If not (l_m has its upper endpoint on or below s') we can — without changing the length of π — move l_m such that the line containing l_m crosses $s' \cap e'$ and then move l_{m-1} upward until l_m crosses $s' \cap e'$. Hence $d'(x, s') \leq d'(x, s)$. $d'(x, s') \geq d'(x, s)$ follows directly from the fact that the last segment of any path to s' should be vertical and can be extended to reach s. Thus $M = \max\{d'(x, s) | x \in P\} = \max\{d'(x, s') | x \in P'\} = M'$.

To prove that D is a correct approximation of $Dia(P)$, we note that by induction $|D' - \max\{d(x, y) | x, y \in P'\}| \leq 3$. Furthermore $|M - \max\{d(x, y) | x \in e, y \in P'\}| \leq 2$, since we have:

$$\begin{array}{rcll} M & = & \max\{d'(x,s) | x \in P'\} & \\ & \leq & \max\{d'(x,y) | x \in P', y \in s\} & \\ & \leq & \max\{d'(x,y) | x \in P', y \in e\} & \\ & \leq & \max\{d(x,y) | x \in P', y \in e\} + 2 & \quad \text{and} \\ M & = & \max\{d'(x,s) | x \in P'\} & \\ & \geq & \max\{d(x,s) | x \in P'\} & \\ & \geq & \max\{d(x,y) | x \in P', y \in e\} - 1. & \end{array}$$

Consequently, $|\max(D', M) - \max\{d(x, y) | x, y \in P\}| = |\max(D', M) - Dia(P)| \leq 3$.

case (iv): We only prove that D is an approximation of the diameter with an error of at most 3. The proof that M is computed correctly uses the same arguments as in case (iii). Again by induction we have $|D' - \max\{d(x, y) | x, y \in P'\}| \leq 3$ and $|D'' - \max\{d(x, y) | x, y \in P''\}| \leq 3$. $|M - \max\{d(x, y) | x \in P, y \in e\}| \leq 2$ is proved as in (iii), so it remains to prove that

$$|(M' + M'' - 2) - \max\{d(x, y) | x \in P', y \in P''\}| \leq 3.$$

This follows from

$$\max\{d(x, y) | x \in P', y \in P''\} = \max\{d(x, e') | x \in P'\} + \max\{d(e', y) | y \in P''\} + \Delta,$$

and the fact that

$$\begin{aligned}
&\Delta \in \{-1, 0, +1\},\\
&\max\{d(x, e') | x \in P'\} \geq \max\{d(x, s') | x \in P'\} - 1 \geq M' - 3,\\
&\max\{d(x, e') | x \in P'\} \leq \max\{d'(x, e') | x \in P'\} \leq M',\\
&\max\{d(e', y) | y \in P''\} = \max\{d(e'', y) | y \in P''\} \geq M'' - 1 \quad \text{and}\\
&\max\{d(e', y) | y \in P''\} \leq M''.
\end{aligned}$$

We can conclude that D is indeed an approximation of the diameter with an error of at most three.

case (v): This is an easy generalization of case (iii). □

5 Concluding Remarks

In this paper we have studied the concept of rectilinear link distance in a simple rectilinear polygon without holes. Two problems concerning this new notion were treated. Firstly, a data structure was devised with which a shortest path between two query points could be computed in time $O(\log n + l)$ (l being the length of the path). It uses $O(n \log n)$ storage. If both query points are vertices of the polygon then a shortest path can even be found in time $O(1 + l)$. The paths found by the query algorithm were also proved to be optimal in the L_1-metric. Secondly, it was shown that the diameter of a rectilinear polygon in the link distance metric can be computed in time $O(n \log n)$ and approximated (with an error of at most three) in linear time.

The solutions to both problems make use of a rectilinear version of Chazelle's polygon cutting theorem, which is also presented in this paper. It states that any simple rectilinear polygon without holes (or having l holes) can be cut into two subpolygons by a (or $\leq l + 1$) segment(s) such that the weights of the resulting polygons are $\leq \frac{3}{4}$ of the weight of the original polygon, which is optimal. Here the weight of a polygon is the sum of the weights of its vertices. To find this cut segment takes only linear time (or $O(n \log n)$ in case there are holes).

Some open problems concerning rectilinear link distance remain. First of all, the results of this paper are not (proved to be) optimal and might be improved. Furthermore the computation of the rectilinear link centre of a polygon is of interest. An interesting thing to note here is that the rectilinear link centre, opposed to the 'ordinary' link centre (see [10]), is not necessarily connected. (A counterexample is left to the interested reader.) Finally, all problems could also be studied in the (much more difficult) case of polygons containing holes or in the three or multi dimensional case.

Acknowledgement

I would like to thank Mark Overmars for his many helpful comments, which improved the contents of this paper considerably.

References

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] Chazelle, B., A Theorem on Polygon Cutting with Applications, *Proc. 23rd Annual IEEE Symp. on Foundations of Computer Science*, 1982, pp. 339-349.

[3] Chazelle, B., and L.J. Guibas, Visibility and Intersection Problems in Plane Geometry, *Proc. 1st Annual ACM Symp. on Computational Geometry*, 1985, pp. 135-146.

[4] Clarkson, K.L., S. Kapoor and P.M. Vaidya, Rectilinear Shortest Paths through Polygonal Obstacles in $O(n(\log n)^2)$ time, *Proc. 3rd Annual ACM Symp. on Computational Geometry*, 1987, pp. 251-257.

[5] Djidjev, H.N., A. Lingas and J. Sack, An $O(n \log n)$ Algorithm for Computing the Link Centre in a Simple Polygon, *Proc. 6th Annual Symp. on Theoretical Aspects of Computer Science (STACS '89)*, 1989, pp. 96-107.

[6] Edelsbrunner, H., L.J. Guibas and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Comput.* 15(1985), pp. 317-340.

[7] Harel, D. and R.E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM J. Comput.* 13(1984), pp. 338-355.

[8] Ke, Y., An Efficient Algorithm for Link Distance Problems, *Proc. 5th Annual ACM Symp. on Computational Geometry*, 1989, to appear.

[9] Larson, R.C., and V.O. Li, Finding minimum rectilinear paths in presence of barriers, *Networks* 11(1981), pp. 285-304.

[10] Lenhart, W., R. Pollack, J. Sack, R. Seidel, M. Sharir, S. Suri, G. Toussaint, S. Whitesides and C. Yap, Computing the Link Centre of a Simple Polygon, *Proc. 3rd Annual ACM Symp. on Computational Geometry*, 1987, pp. 1-10.

[11] van Leeuwen, J., Finding lowest common ancestors in less than logarithmic time, unpublished report, 1976.

[12] Levcopoulos, C., Heuristics for Minimum Decompositions of Polygons, Linköping Studies in Science and Technology. Dissertations. No. 155, 1987.