An Optimal Data Structure for Shortest Rectilinear Path Queries in a Simple Rectilinear Polygon

Sven Schuierer*

Abstract

We present a data structure that allows to preprocess a rectilinear polygon with n vertices such that, for any two query points, the shortest path in the rectilinear link or L_1-metric can be reported in time $O(\log n + k)$ where k is the link length of the shortest path. If only the distance is of interest, the query time reduces to $O(\log n)$. Furthermore, if the query points are two vertices, the distance can be reported in time $O(1)$ and a shortest path can be constructed in time $O(1 + k)$. The data structure can be computed in time $O(n)$ and needs $O(n)$ storage. As an application we present a linear time algorithm to compute the diameter of a simple rectilinear polygon w.r.t. the L_1-metric.

1 Introduction

In many applications such as VLSI-design [1] rectilinear paths play an important role. Usually one is interested in minimizing the length of a path between two points and numerous results have been obtained in this area [2, 3, 4]. A natural metric to measure the length of a rectilinear path is the L_1-metric but sometimes it is also useful to take the number of turns on the path into account. This gives rise to the so called (*rectilinear*) *link metric* which is defined as the number of line segments a path consists of.

A variety of problems have been considered in this context. Das and Narasimhman [5] give an $O(n \log n)$ algorithm to compute the shortest

*Institut für Informatik, Universität Freiburg, Rheinstr. 10, D-79104 Freiburg, Fed. Rep. of Germany, email: schuierer@informatik.uni-freiburg.de

rectilinear link path among a set of disjoint rectilinear polygons. As a by-product they also obtain a linear space data structure to answer shortest rectilinear link path queries in time $O(\log n)$ if one of the query points is fixed. Since it is often desirable to take also the L_1-distance into account, Yang *et al.* [6] construct a graph among the obstacles that preserves shortest rectilinear link and L_1-paths and that can be used to compute the L_1-shortest path between two points with the least number of links or vice versa. de Berg *et al.* [7] consider a linear combination of the rectilinear link and the L_1-metric and construct a data structure that answers shortest path queries according to this combined metric among rectilinear line segments that are allowed to intersect. Again one of the query points is fixed. The time needed for the construction of the data structure is $O(n^2)$ while queries can be answered in $O(\log n)$ time. The storage requirement is $O(n \log n)$. If the L_1-metric alone is considered, two types of data structures have been developed for distance queries among rectangular obstacles given two arbitrary points. The first one by Atallah and Chen [8] answers queries in time $O(\log n)$ and needs $O(n^2 \log n)$ preprocessing time and $O(n^2)$ space while the second one by ElGindy and Mitra [9] answers queries in time $O(\sqrt{n})$ and needs $O(n\sqrt{n})$ preprocessing time and space.

If we restrict ourselves to simple rectilinear polygons, the situation gets considerably simpler since it can be shown that there is always a rectilinear path between two points that is shortest w.r.t. both the rectilinear link and the L_1-metric [10, 11]. We call such a path a *smallest* path [12, 13]. In this setting de Berg [10] presents a data structure that allows to answer the following types of queries. If we are given two arbitrary points inside a polygon, the rectilinear link distance and the L_1-distance between the two points can be reported in time $O(\log n)$. If we are given two vertices of the polygon, both distances can be reported in constant time. Furthermore, a smallest path, i.e., a path that is optimal w.r.t. both metrics can be constructed in an additional number of steps proportional to its link length. His data structure needs $O(n \log n)$ space and preprocessing. We summarize the results in the following table (k is used to denote the link length of the shortest path).

In this paper we present a data structure that achieves the same time complexities as shown in Table 1 but improves both the space and preprocessing requirements to linear in the number of vertices of the polygon.

Table 1: Time needed for L_1-, rectilinear link distance, and smallest path queries.

	Distance (L_1- or rl-distance)	Path (shortest w.r.t. L_1- and rl-distance)
vertices	$O(1)$	$O(1+k)$
points	$O(\log n)$	$O(\log n + k)$

This should also be contrasted with the Euclidean world where there exists a data structure to answer shortest L_2-path queries in a simple polygon in $O(\log n)$ time with linear time preprocessing and space requirements [14] but to achieve the same query time for the link distance $O(n^3)$ preprocessing time and space are needed [15].

The paper is organized as follows. After introducing some of the definitions and notations that are needed in the rest of the paper in Section 2, we invesitgate in Section 3 the structure of shortest paths in the L_1- and the rectilinear link metric. Section 4 deals with a data structure to answer efficiently which points on the boundary of a histogram H can be reached from a query point p with a fixed number of links and how to compute a smallest path from p to H. In Section 5 we show how to connect two given smallest paths inside a histogram. In Section 6 we put the various results together and present the complete query algorithm. Finally, Section 7 deals with an application of our data structure where we show that it can be applied to compute the diameter of a polygon w.r.t. the L_1-metric.

2 Definitions

Let $\mathcal{C}$ be a simple, closed curve consisting of n axes-parallel line segments such that no two consecutive segments are collinear. We define a *simple rectilinear polygon* P to be the union of $\mathcal{C}$ and the set of points that are enclosed by $\mathcal{C}$. A (*rectilinear*) *path* $\mathcal{P}$ is a curve that consists of axes-parallel line segments inside P. From now on, whenever we talk of polygons, we mean simple rectilinear polygons and whenever we talk of paths, we mean rectilinear paths in P.

Let e and e' be two axes-parallel line segments in P. We say a polygonal path $\mathcal{P}$ from line segment e to line segment e' is *admissible* if it is rectilinear and the first link of $\mathcal{P}$ is orthogonal to e and the last link is orthogonal to e'. The *(rectilinear) link length of* $\mathcal{P}$, denoted by rl-length($\mathcal{P}$), is defined as the number of links $\mathcal{P}$ consists of. A *rl-shortest path* is an admissible path $\mathcal{P}$ from any point of e to any point of e' such that there is no other admissible path from e to e' with fewer links. We define $rl(e, e')$ to be the number of links of an rl-shortest path from e to e'. A point p is considered a degenerate line segment and, hence, if $\mathcal{P}$ starts (or ends) at p and $\mathcal{P}$ is admissible, then all axes-parallel directions are allowed for the first (or last) link of $\mathcal{P}$.

If p and q are two points in P and $\mathcal{P}$ is a rectilinear path between them, we define the *L_1-length* of $\mathcal{P}$ as the sum of the Euclidean lengths of its links and denote it by L_1-length($\mathcal{P}$). A path $\mathcal{P}$ from p to q is called a *L_1-shortest* path if there is no other path from p to q with lesser L_1-length. The *L_1-distance* between p and q, denoted by $L_1(p, q)$, is defined as the L_1-length of a L_1-shortest path from p to q. As mentioned before it can be shown that there always exists a path $\mathcal{P}$ from p to q that is L_1-shortest and rl-shortest. Following McDonalds and Peters [13] we call such a path a *smallest* path.

A very important concept in the context of rectilinear link distance is that of a d-interval. If e is an axis-parallel line segment in P and p is a point in P, we define the *d-interval of p on e* to be the set of points on e that p can reach with an admissible path of length d or less and denote it by $e(p, d)$. It is easy to show that $e(p, d)$ is either empty or a connected subsegment of e (see Figure 1a). Note that if $rl(p, e) = d$, then $e(p, d+1)$ may properly contain $e(p, d)$ as illustrated in the figure.

A *histogram* is a monotone rectilinear polygon with one of the monotone chains forming a single line segment called the *base* of the histogram. A *horizontal (vertical)* histogram is a histogram with a horizontal (vertical) base. An *upper* histogram is a horizontal histogram with its interior above the base. Similarly, *lower, left*, and *right* histograms are defined. We say H is a *maximal histogram of* b if b is the base of H and there is no other histogram in P with the same base that properly contains H (see Figure 1b). Note that a line segment may have two maximal histograms, one on each side.

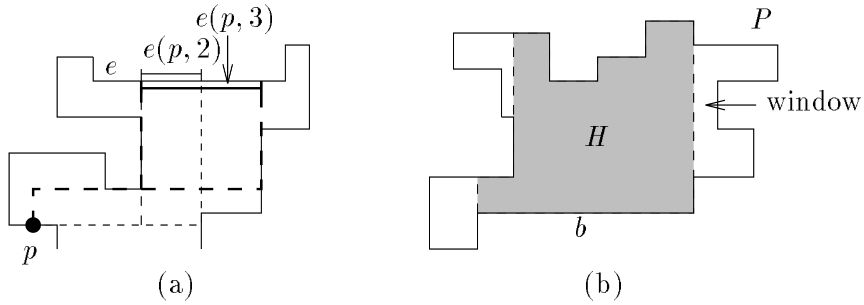

Figure 1: The definition of an interval and a maximal histogram.

A *window* is a maximal segment on the boundary of a histogram that is completely contained in the *interior* of P with the exception of its end points. A vertical window is called a *west* (*east*) *window* if the interior of the histogram is to the right (left) of it. Note that every window splits P into two parts.

Since we will be dealing with trees quite a bit, we give short definitions of the standard terminology in connection with trees. A rooted tree T is a connected, planar, acyclic graph with one designated node, called the *root* r of T. The *depth* of a node v is the length of the path from r to v in T. We denote it by $\text{depth}(v)$. The node v' immediately before v on the unique path $\mathcal{P}$ from r to v is called the *parent* of v and v is called a *child* of v'. The nodes on $\mathcal{P}$ are called the *ancestors* of v and v is called a *descendant* of a node on $\mathcal{P}$. If v_1 and v_2 are two nodes, then the *lowest common ancestor*, denoted by $lca(v_1, v_2)$, is the node of T with the highest depth that is an ancestor of both v_1 and v_2.

3 Smallest Paths in a Simple Rectilinear Polygon

The main structure that allows us to answer smallest path queries efficiently is given by the *histogram tree* which is defined below.

3.1 The Histogram Tree

Let P be a simple rectilinear polygon and e some edge of P. We define the *histogram partition* $\mathcal{H}(P, e)$ of P w.r.t. e inductively as follows [16]. If P is a histogram with base e, then $\mathcal{H}(P, e) = \{P\}$. Otherwise, let H be the maximal histogram of e in P and $w_1, \ldots, w_k$ the windows of H. Each window w_i splits P into two subpolygons one of which does not contain H. We denote this subpolygon by P_{w_i}. The histogram partition is now given by $\mathcal{H}(P, e) = \bigcup_{1 \leq i \leq k} \mathcal{H}(P_{w_i}, w_i) \cup \{H\}$. For illustration see Figure 2a. If a triangulation of the polygon is given—see (Ref. [17]) for a linear time algorithm to compute a triangulation—a histogram partition can be computed in linear time [16].

Note that each window of a histogram in $\mathcal{H}(P, e)$ is the base of some other histogram and vice versa with the only exception of the histogram with base e. Hence, we will use bases and windows interchangeably. Since a window splits P into two subpolygons and every base b is a window, we can speak of the *subpolygon* P_b *of* b which is meant to denote the subpolygon of P that contains the histogram of $\mathcal{H}(P, e)$ with base b. We also say that b *cuts off* P_b in this case.

If we are given a point p in P, we define the *histogram of* p to be the histogram of $\mathcal{H}(P, e)$ that contains p and denote it by H_p. If p is on the boundary between two histograms, then H_p is defined as the one histogram of the two that is closer to e. The base of H_p is denoted by b_p.

There is a natural tree structure associated to the histogram partition. We call it the *histogram tree of* P *and* e and denote it by $\mathcal{T}(P, e)$ (see Figure 2b). It is defined as follows. The nodes of $\mathcal{T}(P, e)$ are the histograms of $\mathcal{H}(P, e)$. There is an edge between histogram H_1 and H_2 in $\mathcal{T}(P, e)$ if the base of H_1 is a window of H_2. The root of $\mathcal{T}(P, e)$ is the histogram with base e. If we use the terminology of trees in connection with histograms, it is implied that we refer to the tree structure of $\mathcal{T}(P, e)$. From now on we will assume that any histogram we refer to is an element of $\mathcal{H}(P, e)$ if not explicitly stated otherwise.

The first observation we make is that a subtree of $\mathcal{H}(P, e)$ that is rooted at histogram H with base b contains exactly those histograms that form the subpolygon P_b of b. Hence, if we are given two query points p_1 and p_2 in the histograms H_1 and H_2, respectively, we conclude that a smallest

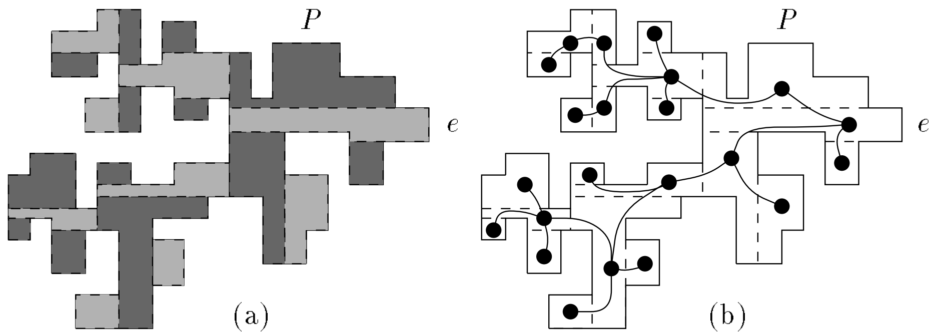

Figure 2: The tree associated to the histogram partition of P.

path from p_1 to p_2 has to pass through the lowest common ancestor H_{lca} of H_1 and H_2. This leads to a three step procedure to answer a smallest path query.

(i) Find H_1 and H_2.

(ii) Compute the parts of the smallest path from p_1 to p_2 that start at p_1 and p_2, respectively, and end at H_{lca}.

(iii) Connect the two parts in H_{lca}.

In the following we treat only the case that H_1 and H_2 are two histograms of $\mathcal{H}(P, e)$ such that neither H_1 is an ancestor of H_2 nor vice versa. If one of the histograms turns out to be an ancestor of the other, the results presented here can easily be adapted.

In the remainder of this section we denote the lowest common ancestor of H_1 and H_2 by H_{lca} and the window of H_{lca} that cuts off the subpolygon containing histogram H_1 (H_2) by w_1 (w_2). For the ease of discussion we assume from now on that H_{lca} is an upper histogram unless explicitly stated otherwise.

3.2 Shortest Paths in the L_1-Metric

In order to construct a smallest path that goes through H_{lca} we need to find a way to show that it indeed minimizes the distance. Since smallest paths are shortest w.r.t. both the L_1- and the rectilinear link metric, two separate investigations have to be carried out. We start with the following observation about L_1-shortest paths.

Lemma 3.1 *If p is a point in P and c a line segment that splits P into two parts, then there is one point q on c such that $L_1(p,q) = L_1(p,c)$ and $L_1(p,p') = L_1(p,q) + L_1(q,p')$, for all points p' on c.*

Proof: If $rl(p,c) = 1$, then the claim is obvious. So assume that $rl(p,c) > 1$ and let p' be some point on c. The line segment c divides P into two parts P_1 and P_2. Let P_1 be the subpolygon that contains p and H the maximal histogram of c that is contained in P_1. Furthermore, let $\mathcal{P}$ be a L_1-shortest path from p to p'. There is one window w of H that is intersected by $\mathcal{P}$. Note that w is independent of p'. Let p_w be the intersection point of $\mathcal{P}$ and w and let q be the orthogonal projection of p_w onto c. Hence, we obtain

$$\begin{aligned} L_1(p,p') &= L_1(p,p_w) + L_1(p_w,p') = L_1(p,p_w) + L_1(p_w,q) + L_1(q,p') \\ &\geq L_1(p,q) + L_1(q,p'). \end{aligned}$$

This immediately implies that $L_1(p,c) = L_1(p,q)$ and that q is unique since q is independent of the position of p_w on w. □

In the following we denote the unique point q on c with $L_1(p,q) = L_1(p,c)$ as *the L_1-projection of p onto c*. With the help of the definition of the L_1-projection of p onto c we can now characterize how a L_1-shortest path between the two points p_1 and p_2 can be constructed.

Lemma 3.2 *If q_i is defined as the L_1-projection of p_i onto w_i, for $i = 1,2$, then $L_1(p_1,p_2) = L_1(p_1,q_1) + L_1(q_1,q_2) + L_1(q_2,p_2)$.*

Proof: Let $\mathcal{P}$ be a L_1-shortest path between p_1 and p_2. Let r_1 (r_2) be the first intersection point of $\mathcal{P}$ with w_1 (w_2).

$$\begin{aligned} L_1(p_1,p_2) &= L_1\text{-length}(\mathcal{P}) = L_1(p_1,r_1) + L_1(r_1,r_2) + L_1(r_2,p_2) \\ &= L_1(p_1,q_1) + L_1(q_1,r_1) + L_1(r_1,r_2) + L_1(r_2,q_2) + L_1(q_2,p_2) \\ &\geq L_1(p_1,q_1) + L_1(q_1,q_2) + L_1(q_2,p_2) \end{aligned}$$

which completes the proof. □

The above lemma suggests to first compute q_1 and q_2 and two L_1-shortest paths from p_i to q_i and then to connect them in H_{lca}. Since we are interested in smallest paths, we will not follow this idea directly. We first have to consider shortest paths in the rectilinear link metric.

3.3 Shortest Paths in the Rectilinear Link Metric

We now consider the rectilinear link metric where *d-intervals* play a crucial role. Recall that the d-interval $w(p,d)$ of a point p on a window w is the set of points on w such that there is an admissible path of d links or less from p to w. Observe that the only interesting values for d are $rl(p,w)$ and $rl(p,w)+1$ since we have $w(p,d)=\emptyset$ if $d < rl(p,w)$ and $w(p,d)=w$ if $d > rl(p,w)+1$. We call the intervals $w(p,d)$ with $d = rl(p,w)$ or $d = rl(p,w)+1$ *non-trivial*. If we denote $rl(p_i,w_i)$ by ρ_i, then the above considerations lead to the following lemma.

Lemma 3.3 *There is a rl-shortest path $\mathcal{P}$ from p_1 to p_2 such that the first link of $\mathcal{P}$ that intersects w_1 orthogonally is either the ρ_1^{st} or the $(\rho_1+1)^{\mathrm{st}}$ link of $\mathcal{P}$.*

We show that with the help of the non-trivial intervals of p_1 on w_1 and p_2 on w_2 it is possible to compute the length of a rl-shortest path from p_1 to p_2. To this end let $I_i(d)$ be defined as $w_i(p_i,d)$, for $i=1,2$ and $k(p_1,p_2)$ as the minimum of $rl(I_1(d_1), I_2(d_2)) + \sum_{i=1,2} d_i - \rho_i$ where d_i ranges over the values in $\{\rho_i, \rho_i+1\}$.

The intuitive meaning behind this definition is that we have to consider the possibility that it might pay off to spend one more link on the path from p_1 to w_1 or from p_2 to w_2 than necessary if it is then easier to connect the two paths. This can be seen if we expand the definition of $k(p_1,p_2)$. $k(p_1,p_2)$ is defined as the minimum of the four values

(i) $rl(I_1(\rho_1), I_2(\rho_2))$,

(ii) $rl(I_1(\rho_1+1), I_2(\rho_2))+1$,

(iii) $rl(I_1(\rho_1), I_2(\rho_2+1))+1$, and

(iv) $rl(I_1(\rho_1+1), I_2(\rho_2+1))+2$.

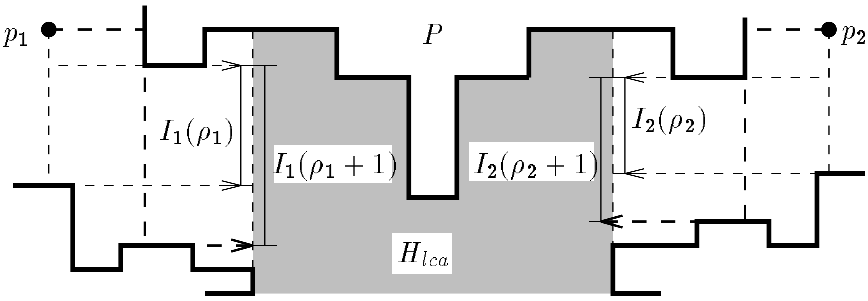

Figure 3: Illustrating that it may be useful to spend one more link on the paths from p_1 and p_2 to w_1 and w_2.

Each link we spend more than necessary on the path from p_1 to w_1 and p_2 to w_2 is taken into account by adding it to the distance between the two intervals that can be reached with the given number of links. As an example consider Figure 3. Here $\rho_1 = \rho_2 = 2$ and $k(p_1, p_2)$ is the minimum of $\{5, 4, 4, 3\}$ and, hence, $k(p_1, p_2) = rl(I_1(3), I_2(3)) + 2$. As it is shown in the following lemma this implies that $rl(p_1, p_2) = \rho_1 + \rho_2 - 2 + k(p_1, p_2) = 2 + 2 - 2 + 3 = 5$.

Lemma 3.4 *If $k(p_1, p_2)$ is defined as above, then*

$$rl(p_1, p_2) = \rho_1 + \rho_2 - 2 + k(p_1, p_2).$$

Proof: To see that $rl(p_1, p_2) \geq \rho_1 + \rho_2 - 2 + k(p_1, p_2)$ let $\mathcal{P}$ be a rl-shortest path from p_1 to p_2. Let l_1 be the first horizontal link of $\mathcal{P}$ that intersects w_1 as seen from p_1 and l_2 the first horizontal link that intersects w_2 as seen from p_2. Suppose l_1 is the d_1^{st} of $\mathcal{P}$ as seen from p_1 and l_2 is the d_2^{nd} link of $\mathcal{P}$ as seen from p_2. By Lemma 3.3 we can assume that $\rho_i \leq d_i \leq \rho_i + 1$, for $i = 1, 2$. Hence,

$$\begin{aligned} rl\text{-length}(\mathcal{P}) &= d_1 + d_2 + rl(I_1(d_1), I_2(d_2)) - 2 \\ &= \rho_1 + \rho_2 + \sum_{i=1,2} d_i - \rho_i + rl(I_1(d_1), I_2(d_2)) - 2 \\ &\geq \rho_1 + \rho_2 - 2 + k(p_1, p_2) \end{aligned} \tag{1}$$

since $k(p_1, p_2)$ minimizes $rl(I_1(d_1), I_2(d_2)) + \sum_{i=1,2} d_i - \rho_i$, for $\rho_i \leq d_i \leq \rho_i + 1$. Note that we have to subtract two since we consider admissible paths and, therefore, the links that meet at w_1 and w_2 can be joined into one.

On the other hand, since there exist d_1 and d_2 such that equality holds in (1) and $\mathcal{P}$ is a rl-shortest path from p_1 to p_2 we also have $rl(p_1, p_2) \leq \rho_1 + \rho_2 - 2 + k(p_1, p_2)$ which completes the proof. $\square$

In order to be able to compute $k(p_1, p_2)$ we are left with the following problems.

(i) Find the intervals $I_1(\rho_1)$, $I_1(\rho_1 + 1)$, $I_2(\rho_2)$, and $I_2(\rho_2 + 1)$.

(ii) Compute the distance $rl(I_1, I_2)$ between each pair of intervals in H_{lca}, for $I_1 \in \{I_1(\rho_1), I_1(\rho_1 + 1)\}$ and $I_2 \in \{I_2(\rho_2), I_2(\rho_2 + 1)\}$.

4 A Data Structure for Interval Queries

In this section we are concerned with a query problem of the following type. Given a point p and an ancestor H of H_p such that window w of H cuts off the subpolygon containing H_p report $w(p, rl(p, w))$ and $w(p, rl(p, w) + 1)$. We will reduce this problem to a tree problem.

The first observation we need is that though there are a quadratic number of possible intervals on all windows of $\mathcal{H}(P, e)$, there are only a linear number of *interval end points.* In the next lemma we show that, indeed, the interval end points are given by the orthogonal projection of the vertices of the histograms onto their bases.

Lemma 4.1 *Let p be a point in P and H an ancestor of H_p with base b. If $d \geq 2$, then the end points of $b(p, d)$ are orthogonal projections of two vertices of H onto b.*

Furthermore, if $rl(p, b) + 1 \geq d$ and $rl(p, b) \geq 2$, then the L_1-projection of p onto b is one of the end points of $b(p, d)$.

Proof: Assume for a contradiction and without loss of generality that H is an upper histogram and that the right end point q of $b(p, d)$ is not the vertical projection of a vertex of H. Let $\mathcal{P}$ be a path of rl-length d from p to b and let ℓ be the last link of $\mathcal{P}$. Because q is not the vertical projection of a vertex of H, ℓ does not touch a vertical edge of P. Hence, ℓ can be moved slightly to the right to obtain an admissible path $\mathcal{P}'$ to

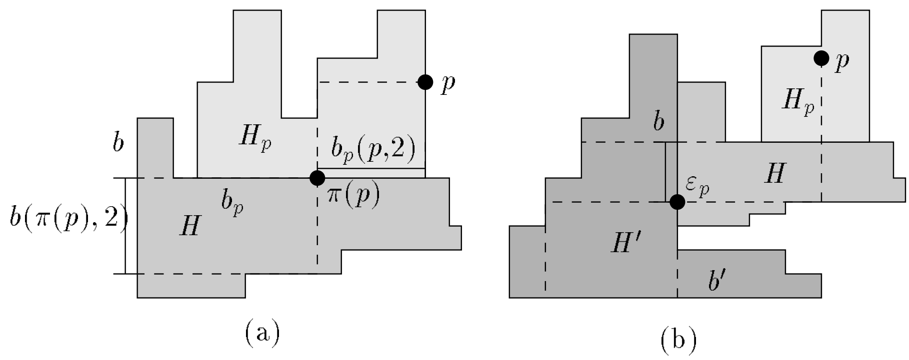

Figure 4: The definition of the parent $\pi(p)$ and the entry ε_p point of p.

a point q' on b to the right of q with rl-length$(\mathcal{P}') = d$ contradicting the fact that q is the right end point of $b(p, d)$.

Recall from the proof of Lemma 3.1 that the L_1-projection of p onto b is the projection of w onto b where w is the window of H that cuts off the subpolygon containing b. From this the second claim easily follows. □

Since the total complexity of the histogram decomposition is linear as shown by Levcopoulos [16], there is only a linear number of possible interval end points in P. We denote this set by $\mathcal{I}$.

The second observation that allows for the efficient computation of intervals is that, for each interval, there is one end point that "dominates" all other points in the interval. This is stated more precisely in the next lemma.

Lemma 4.2 *If H is an upper histogram with base b and I a vertical interval on the boundary of H with lower end point p_l, then $b(r, 2) \subseteq b(p_l, 2)$, for all $r \in I$.*

Proof: Let l be the horizontal line segment in H that p_l reaches with one link and $\mathcal{P}$ a two link path from r to a point p_b on b. Since p_l is below r, the vertical link of $\mathcal{P}$ intersects l and, hence, there is also a two-link path from p_l to p_b. □

Lemma 4.2 in particular implies that $b(I, 2) = \bigcup_{r \in I} b(r, 2) \subseteq b(p_l, 2)$ and leads us to the following definition which is illustrated in Figure 4a.

Definition 4.1 *Let p be a point in $\mathcal{I}$ and H the parent of H_p with base b. By Lemma 4.2 there is one end point p_l of $I = b_p(p, 2)$ with $b(p_l, 2) =$*

$\bigcup_{r \in I} b(r,2)$. *We call* p_l *the* parent of p *and denote it by* $\pi(p)$.

Note that $\pi(p)$ is only defined if the parent of H_p exists, that is, if $rl(p,e) > 1$.

The parent relation immediately induces a tree structure on the points in $\mathcal{I}$ and we can now talk of the depth of point in $\mathcal{I}$ etc. Of course, we do not obtain one tree but a forest of rooted trees. The roots of the trees are the points in $\mathcal{I}$ that are on the windows of the root histogram of $\mathcal{T}(P,e)$.

In order to compute this forest in linear time we proceed as follows. Let H be a horizontal histogram with windows $w_1, \ldots, w_k$ and base b. For each point p of $\mathcal{I}$ on w_i, $1 \leq i \leq k$, we compute the vertical edge e_p of H which is horizontally visible from p. A vertical edge e_v is said to be *horizontally visible* from a point p if there is a horizontal line segment that is contained in the interior of H with the exception of its end points and that connects p to a point on e_v. By the triangulation algorithm of Chazelle [17] this can be computed in time proportional to the size of H and $H \cap \mathcal{I}$.

Note that the x-coordinate of w_i determines one end point of $b(p,2)$ and the x-coordinate of the corresponding edge e_p the other end point. Note also that $\pi(p)$ is the end point of $b(p,2)$ that is closer to the base of the parent of H by Lemma 4.2. Hence, we can compute $\pi(p)$, for all points $p \in \mathcal{I} \cap H$, in one scan of the upper boundary of H in time $O(|\mathcal{I} \cap H| + |H|)$ where $|\cdot|$ denotes the cardinality of a set. So the total time used to perform this computation for all histograms $H \in \mathcal{H}(P,e)$ is still linear.

As a by-product of the above algorithm we also obtain $b_p(p,2)$, for each point $p \in \mathcal{I}$. Furthermore, if v is a vertex of P that is projected onto p, it is easily seen that $b_p(v,2) = b_p(p,2)$ which we can also compute for each vertex. Hence, we can introduce a notion for the vertices of P which is similar to the parent relation among the points of $\mathcal{I}$ by defining the "parent" of v to be the parent of p. This idea is formalized and extended to arbitrary points in the following definition. For illustration refer to Figure 4b.

Definition 4.2 *Let* p *be a point in* P, H *the parent of* H_p *with base* b, *and* H' *the parent of* H *with base* b'. *By Lemma 4.2 there is one end point*

p_l of $I = b(p, 2)$ with $b'(p_l, 2) = \bigcup_{r \in I} b'(r, 2)$. We call p_l the entry point *of p and denote it by ε_p.*

There is one fundamental difference between Definition 4.1 and Definition 4.2 which can best be seen if we consider a point p in $\mathcal{I}$. The parent of p is a point on the base of H_p while the entry point of p is a point on the base of the parent of H_p. The reason for this difference is that the parent relationship is designed to construct admissible paths from p to its ancestors while for the entry point it is not important whether the first link is orthogonal to the window p belongs to or not.

The following lemma shows how the tree structure of $\mathcal{I}$ can be employed to find one non-trivial interval of point p in $\mathcal{I}$ on window w if $rl(p, w)$ is known. In order to do so we define $\pi^k(p) = \pi(\pi^{k-1}(p))$, for $k > 0$, and $\pi^0(p) = p$.

Lemma 4.3 *Let p be a point in $\mathcal{I}$ and H an ancestor of H_p such that window w of H cuts off the subpolygon containing H_p. If $\rho = rl(p, w)$, then $w(p, \rho + 1) = w(\pi^{\rho-1}(p), 2)$.*

Proof: The proof is by induction on ρ. It obviously holds if $\rho = 1$. So assume that $\rho > 1$. Let H' be the child of H that is also an ancestor of H_p and w' the window of H' that cuts off the subpolygon containing H_p. We first show that $w(p, \rho + 1) \subseteq w(\pi^{\rho-1}(p), 2)$. To this end let r be some point in $w(p, \rho + 1)$ and $\mathcal{P}$ an admissible path of length $\rho + 1$ from p to r. The path $\mathcal{P}$ intersects w' in a point p'. By Lemma 4.2 we have $w(p', 2) \subseteq w(p'_l, 2)$ where p'_l is the closest point of $w'(p, \rho)$ to w. On the other hand, we have $w'(p, \rho) = w'(\pi^{\rho-2}(p), 2)$ by the induction hypothesis. Hence, by Definition 4.1 p'_l is the parent of $\pi^{\rho-2}(p)$ and r belongs to $w(\pi^{\rho-1}(p), 2)$.

To see the reverse inclusion just note that there is an admissible path $\mathcal{P}'$ of length ρ from p to p'_l by the induction hypothesis and, hence, any point on w that is reachable from p'_l with an admissible path of length two can be reached from p with $\rho + 2 - 1$ links. $\square$

If we consider arbitrary points in P, then we obtain the following result.

Lemma 4.4 *Let p be a point in P and H an ancestor of H_p such that window w of H cuts off the subpolygon containing H_p. If $\rho = rl(p, w) \geq 3$ and ε_p is the entry point of p, then $w(p, \rho) = w(\pi^{\rho-3}(\varepsilon_p), 2)$.*

Proof: Let ε_p be the entry point of p. Since $rl(\varepsilon_p, w) = \rho - 2$, Lemma 4.3 implies that $w(\varepsilon_p, \rho - 1) = w(\pi^{\rho-3}(\varepsilon_p), 2)$. Therefore, it suffices to show that $w(\varepsilon_p, \rho - 1) = w(p, \rho)$. To see this, first note that $w(\varepsilon_p, \rho - 1) \subseteq w(p, \rho)$ since there is a path of two links from p to ε_p that is orthogonal to the window w' which ε_p belongs to and the first link of a path of $\rho - 1$ links from ε_p to w is also orthogonal to w'.

To see the reverse inclusion consider the base b of H_{ε_p}. By the definition of ε_p there is an admissible path of three links from p to any point of $b(p, 3)$ such that the middle link contains ε_p. This implies that any point that can be reached with ρ links from p can be reached with $\rho - 1$ links from ε_p which proves the claim. □

In order to compute the $\rho + 1$-interval we just have to consider a different entry point (see Figure 4a).

Definition 4.3 *Let p be a point in P and H the parent of H_p with base b. By Lemma 4.2 there is one end point p_l of $I = b_p(p, 2)$ with $b(p_l, 2) = \bigcup_{r \in I} b(r, 2)$. We call p_l the* second entry point of p *and denote it by δ_p.*

Note that if p is in $\mathcal{I}$, then the second entry point δ_p and the parent $\pi(p)$ of p are identical. With the help of the second entry point of p there is an analogous result to Lemma 4.4 for the interval $w(p, \rho + 1)$.

Lemma 4.5 *Let p be a point in P and H an ancestor of H_p such that window w of H cuts off the subpolygon containing H_p. If $\rho = rl(p, w) \geq 2$ and δ_p is the second entry point of p, then $w(p, \rho + 1) = w(\pi^{\rho-2}(\delta_p), 2)$.*

Proof: Similar to the proof of Lemma 4.4. □

Note that Lemma 4.5 is also a direct generalization of Lemma 4.3 since for a point p in $\mathcal{I}$ we have $w(p, \rho + 1) = w(\pi^{\rho-2}(\delta_p), 2) = w(\pi^{\rho-2}(\pi(p)), 2)$ by the remark before the lemma.

Above we always assumed that $rl(p, w)$ is already known. Hence, we also have to address the question how to compute $rl(p, w)$ efficiently which is dealt with in the following lemma.

Lemma 4.6 *If p is a point in P and H an ancestor of H_p such that window w of H cuts off the subpolygon containing H_p, then the link distance from p to w is* $\mathrm{depth}(H_p) - \mathrm{depth}(H)$.

Proof: We prove the claim by induction on $rl(p, w)$. First note that by the definition of H_p, $rl(p, w) \geq 1$ if $H \neq H_p$. Therefore, if w is the base of H_p, then we can reach a point on w with one link. Hence, the claim holds if $\mathrm{depth}(H_p) - \mathrm{depth}(H) = 1$. Now let H' be the histogram with base w and w' the window of H' that cuts off the subpolygon containing H_p. Let $\rho = rl(p, w)$. Clearly, we have $rl(p, w') = \rho - 1$ since any path from p to w intersects w' and w can be reached with one link from w'; hence, $\mathrm{depth}(H') - \mathrm{depth}(H_p) = \rho - 1$ by the induction hypothesis. Since the base of H' is a window of H, H' is a child of H, and we have $\mathrm{depth}(H) - \mathrm{depth}(H_p) = \rho$. □

We now put together the various considerations of this section. So let p be point in P and w the window of the ancestor H of H_p that cuts off the subpolygon that contains H_p. We assume that the rl-distance ρ from p to w is at least three links. The following algorithm outputs the ρ-interval and $(\rho + 1)$-interval of p on w.

Algorithm Interval Query

Input: A query point p and the window w of a histogram H which cuts off the subpolygon containing p.

Output: the two non-trivial intervals of p on w.

1. let H_p be the histogram containing p.
2. let $\rho := \mathrm{depth}(H_p) - \mathrm{depth}(H)$;
   (* compute $w(p, \rho)$ *);
3. compute the ancestor α of the entry point ε_p of p at depth $\mathrm{depth}(\varepsilon_p) - (\rho - 3)$;
4. output $w(\alpha, 2)$ which is $w(p, \rho)$;
   (* compute $w(p, \rho + 1)$ *);
5. compute the ancestor β of the second entry point δ_p of p at depth $\mathrm{depth}(\delta_p) - (\rho - 2)$;
6. output $w(\beta, 2)$ which is $w(p, \rho + 1)$.

Since the algorithm works only for $\rho \geq 3$, the case $\rho = 2$ remains to be considered. If $\rho = 2$, then the 2-interval of p on w either is already precomputed for vertices or can be found by point location for points. The 3-interval of p on w can be computed as in the second part of Algorithm *Interval Query*.

In order to analyse the time needed to execute the above algorithm note that if p is an arbitrary point in P, then the histogram H_p of p, the parent $\pi(p)$ of p, $b_p(p,2)$, and the entry and second entry point of p can be easily computed in logarithmic time with the help of the point location data structure of Edelsbrunner *et al.* [18]; if p is vertex of P or belongs to $\mathcal{I}$, then we can assume that H_p, $\pi(p)$, $b_p(p,2)$, and the entry and second entry point of p are precomputed as mentioned before. This yields the time bounds for Steps 1, 4, 6, and the entry point computation in Steps 3 and 5. For Step 2 note that the distance from p to w can be computed in constant time since we can assume the depth of a histogram is precomputed. Steps 3 and 5 can also be carried out in constant time by a result of Berkman and Vishkin on computing level ancestors in trees [21] if $\mathcal{I}$ is preprocessed appropriately in linear time.

4.1 Computing a Smallest Path to a Histogram

In the previous subsection we showed how to compute a data structure that allows efficient computation of the non-trivial intervals of a point p on a window w. In this subsection we show how to compute a smallest path from p to w. As before let $\rho = rl(p, w)$. It is easy to see that an admissible rl-shortest path from p to any point p_w on $w(p, \rho)$ can be obtained by connecting p to the entry point ε_p of p, the i^{th} parent of ε_p to the $i + 1^{\text{st}}$ parent of ε_p with an admissible path of two links, for $0 \le i \le \rho - 3$ and, finally, by connecting $\pi^{\rho-3}(\varepsilon_p)$ to p_w with two links. Let the path constructed in this way be called $\mathcal{P}$. We claim that $\mathcal{P}$ is also a L_1-shortest path.

Lemma 4.7 *If $\mathcal{P}$ is defined as above, then* L_1-length$(\mathcal{P}) = L_1(p, p_w)$.

Proof: We prove the claim by induction. Observe that the path from p to the entry point ε_p of p is a L_1-shortest path. Now let ρ be $rl(p, w)$. We prove the induction basis $\rho = 3$ and the induction step for $\rho > 3$ simultaneously. Consider $\pi^{\rho-3}(\varepsilon_p)$ and let w' be the window it belongs to. Let q be the L_1-projection of p onto w and q' be the L_1-projection of p onto w'. Since $\pi^{\rho-3}(\varepsilon_p)$ is the closest point of $w'(p, \rho - 1)$ to w by definition and q' is one of the end points of $w'(p, \rho - 1)$ by Lemma 4.1, $L_1(q', q) = L_1(q', \pi^{\rho-3}(\varepsilon_p)) + L_1(\pi^{\rho-3}(\varepsilon_p), q)$; furthermore, $L_1(p, p_w) = L_1(p, q') + L_1(q', q) + L_1(q, p_w)$ by Lemma 3.2 which proves the claim if we apply the above observation to ε_p resp. the induction hypothesis to $\pi^{\rho-3}(\varepsilon_p)$. □

A similar statement holds if p_w belongs to $w(p, \rho + 1)$.

In order to compute the L_1-distance from $p \in \mathcal{I}$ to some point p_w we proceed as follows. If r is the root of the tree containing p, we define the L_1-*depth* of a point $p \in \mathcal{I}$ to be the length of a L_1-shortest path $\mathcal{P}$ from p to r and denote it by L_1-depth(p). As we have proved in Lemma 4.7, L_1-length$(\mathcal{P})$ is equal to the sum of the L_1-distances between the nodes on the path in the tree between p and r and, hence, can be precomputed for all nodes in one traversal of the tree. So in order to compute the L_1-distance between p and a point p_w on window w we have the following algorithm.

Algorithm L_1-Distance to Histogram

Input: A query point p and a point p_w on the window w of a histogram H that cuts off the subpolygon containing p.

Output: $L_1(p, p_w)$

1. let ε_p be the entry point of p;
2. compute $\rho = rl(p, w)$;
3. find the ancestor α of ε_p at depth $(\text{depth}(\varepsilon_p) - (\rho - 3))$;
4. compute $L_1(\varepsilon_p, \alpha) = L_1\text{-depth}(\varepsilon_p) - L_1\text{-depth}(\alpha)$;
5. compute $L_1(\alpha, p_w)$;
6. set $L_1(p, p_w) = L_1(p, \varepsilon_p) + L_1(\varepsilon_p, \alpha) + L_1(\alpha, p_w)$.

Clearly, all steps but the first one can be carried out in constant time while for the first step point location is needed if p is not a vertex. If p is a vertex, the entry point of p can be precomputed as mentioned before.

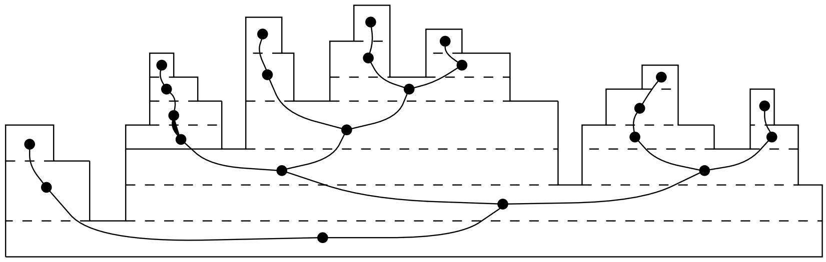

Figure 5: The tree associated to a histogram.

5 Connecting the Paths

As a last step we have to show how to compute the L_1- and rl-distance between two given intervals in an upper histogram H and how to connect the two intervals we choose by a subpath of a smallest path. In order to do so we subdivide H by extending the horizontal edges to the left or right until they meet another edge of H. Thus, H is divided into a number of rectangles or *cells*. We denote this cell decomposition of H by S_H. There is a natural tree structure T_H associated to S_H. The nodes of T_H are the cells of S_H. There is an edge between two nodes if the two cells are adjacent in the subdivision. The root of the tree is the cell adjacent to the base of H (see Figure 5).

For a point p in H, we denote the cell of S_H (= node of T_H) p belongs to by C_p. If R is a rectangle, we denote its projection onto the base b of H by $b(R)$. Clearly, S_H and T_H can be computed in time proportional to the size of H.

With the help of the above defined subdivision we can now give a characterization of the rl-distance between two intervals.

Lemma 5.1 *If H is an upper histogram and I_1 and I_2 are two vertical line segments on the boundary of H whose lower end points belong to the rectangles R_1 and R_2, then*

$$rl(I_1, I_2) = \begin{cases} 1 & \textit{if the two } y\textit{-intervals of } I_1 \textit{ and } I_2 \textit{ overlap} \\ & \textit{and } R_1 \textit{ intersects } I_2 \textit{ or vice versa;} \\ 5 & \textit{if } lca(R_1, R_2) \textit{ is different from } R_1 \textit{ and } R_2\textit{;} \\ 3 & \textit{otherwise.} \end{cases}$$

Proof: We first note that an admissible path between two intervals that are both vertical always consists of an odd number of links since the first and last link have to be horizontal.

Let the lower end of I_1 (I_2) be l_1 (l_2) and h_1 (h_2) the maximal horizontal line segment incident to l_1 (l_2).

To see the first claim note that $rl(I_1, I_2) = 1$ if there is a horizontal line that connects I_1 and I_2 in H. This implies that either h_1 intersects I_2 or vice versa which in turn implies the conditions of the claim. If the conditions are met and, say, R_1 intersects I_2, then either h_1 or h_2 connect I_1 to I_2.

To see the second claim we first observe that rectangle R is an ancestor of rectangle R' in T_H if and only if $b(R)$ contains $b(R')$. Furthermore, if $b(R) \cap b(R') \neq \emptyset$, for any two rectangles R and R', then either $b(R)$ contains $b(R')$ or vice versa. Hence, $lca(R_1, R_2)$ differs from R_1 and R_2 if and only if $b(R_1) \cap b(R_2) = \emptyset$. Now if there is a path of three or less links connecting I_1 and I_2, then $b(R_1) \cap b(R_2) \neq \emptyset$ since the projection of the vertical link of the path onto the base is an intersection point. On the other hand, if there is an intersection point of $b(R_1)$ and $b(R_2)$, we can connect h_1 and h_2 with one vertical segment which yields a path of length three.

We finally note that we can connect any two points in a histogram with three links and we need at most two more to make the path admissible. Hence, any admissible path between two intervals in H has length at most five which completes the proof. $\square$

Now assume we are given two query points p_1 and p_2 in P such that the smallest path $\mathcal{P}$ from p_1 to p_2 passes through H. Let the windows that cut off the subpolygons that contain p_1 and p_2, respectively, be denoted by w_1 and w_2. Furthermore, let I_1 be the smallest non-trivial interval of p_1 on w_1 such that $\mathcal{P}$ passes through I_1 and I_2 the smallest non-trivial interval of p_2 on w_2 such that $\mathcal{P}$ passes through I_2. We now describe an algorithm to connect the two intervals with a smallest path which is part of a smallest path from p_1 to p_2. Let $I_1 = [l_1, u_1]$ and $I_2 = [l_2, u_2]$ where u_1 (u_2) is the upper and l_1 (l_2) the lower end point of I_1 (I_2). W.l.o.g. we assume that l_1 is below l_2.

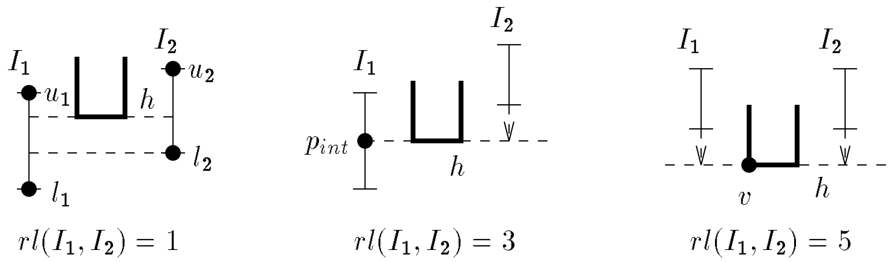

Figure 6: Several cases can occur when connecting two intervals.

Algorithm Connecting the Intervals
Input: Two points p_1 and p_2 and their intervals I_1 and I_2 on the boundary of H;
Output: The part of the smallest path from p_1 to p_2 that is contained in H.
let q_i be the L_1-projection of p_i onto w_i, for $i = 1, 2$;
let h_i be the maximal horizontal line segment in H incident to q_i, for $i = 1, 2$;
let $R = lca(C_{u_1}, C_{u_2})$ and h_R the upper edge of R;
let h_{u_1} be the maximal horizontal line segment in H incident to u_1;
let h be the lower of the two line segments h_R and h_{u_1};
if $rl(I_1, I_2) = 1$
  then if h_1 intersects I_2 **or** h_2 intersects I_1
    then connect I_1 to I_2 with h_1 or h_2;
    else connect I_1 to I_2 with h;
if $rl(I_1, I_2) = 3$
  then if $C_{u_1} = C_{u_2}$
    then connect q_1 to q_2 with a three link path in C_{u_1};
    else connect q_2 to h with one vertical link;
if $rl(I_1, I_2) = 5$
  then connect q_1 to h with one vertical link;
  connect q_2 to h with one vertical link;

Figure 6 illustrates the different cases that can occur. Note that a path from a point p on I_1 to a point q on I_2 induces a path in T_H from C_p to C_q and, hence, always passes through $lca(C_p, C_q)$. Note also that the path constructed for $rl(I_1, I_2) = 3$ and $= 5$ is not admissible and actually a path of length 2 and 3, respectively. But it is easy to see that we can construct an admissible path of length 3 and 5 by moving the vertical links a small enough amount to the left and right, respectively, and by adding a horizontal link to each vertical link.

In the following let $\mathcal{P}_{I_1 I_2}$ be the path constructed by Algorithm *Connecting the Intervals*. In order to show that Algorithm *Connecting the Intervals* is correct we first note that there is a rl-shortest path that contains $\mathcal{P}_{I_1 I_2}$ by our choice of I_1 and I_2. So we only have to show that there is also a L_1-shortest path that contains $\mathcal{P}_{I_1 I_2}$.

Let r_1 be the end point of $\mathcal{P}_{I_1 I_2}$ on I_1 and r_2 the end point of $\mathcal{P}_{I_1 I_2}$ on I_2. Let $\mathcal{P}'_{I_1 I_2}$ be the path $\mathcal{P}_{I_1 I_2}$ augmented with the two vertical line segments from r_i to the L_1-projection of p_i onto w_i which we denote by q_i. With this notation we can show the following result.

Lemma 5.2 $\mathcal{P}'_{I_1 I_2}$ *is a* L_1*-shortest path from* q_1 *to* q_2*.*

Proof: Let l_1 be below l_2, $R = lca(C_{u_1}, C_{u_2})$, and h as in the algorithm. Furthermore, let p_{int} be the intersection point of h and I_1 if it exists. We distinguish three cases according to the algorithm.

Case 1 $rl(I_1, I_2) = 1$.
If h_1 connects I_1 to I_2, then $\mathcal{P}'_{I_1 I_2}$ is a two link path and, hence, the L_1-length of $\mathcal{P}'_{I_1 I_2}$ is $L_1(q_1, q_2)$ as claimed; a similar statement holds, if h_2 connects I_2 to I_1. If neither of this is true, then $q_2 = u_2$ since l_1 is below l_2. If $q_1 = l_1$, then $\mathcal{P}'_{I_1 I_2}$ is a monotone path from q_1 to q_2, hence L_1-shortest. Otherwise, $q_1 = u_1$ and any path from u_1 to u_2 passes through h. Since h directly connects I_1 to I_2, $\mathcal{P}'_{I_1 I_2}$ has length $L_1(q_1, q_2)$.

Case 2 $rl(I_1, I_2) = 3$.
If $C_{q_1} = C_{q_2}$, then q_1 is connected to q_2 by a monotone three link path of length $L_1(q_1, q_2)$. Otherwise, R intersects I_1 since l_1 is below l_2. If $q_1 = l_1$, then the path from l_1 to p_{int} to q_2 is monotone and, therefore, has length $L_1(q_1, q_2)$. If $q_1 = u_1$, then any path from u_1 to a point in I_2 intersects h and L_1-length$(\mathcal{P}'_{I_1 I_2}) = L_1(q_1, q_2)$ as in Case 1.

Case 3 $rl(I_1, I_2) = 5$.
There is an edge e of H incident to two reflex vertices that intersect $h = h_R$ between q_1 and q_2. Let v be one vertex of e. Clearly, L_1-length$(\mathcal{P}'_{I_1 I_2}) =$

$L_1(q_1, v) + L_1(v, q_2)$. Since every path $\mathcal{P}$ from q_1 to q_2 has to intersect the vertical line segment starting in v downwards, $L_1(q_1, v) + L_1(v, q_2)$ is a lower bound on the length of a path from q_1 to q_2 and, hence, $\mathcal{P}'_{I_1I_2}$ is an L_1-shortest path.

$\square$

Lemma 5.2 immediately implies that there is a L_1-shortest path that contains $\mathcal{P}_{I_1I_2}$ since $L_1(p_1, p_2) = L_1(p_1, q_1) + L_1(q_1, q_2) + L_1(q_2, p_2)$ by Lemma 3.2.

With the above result the task of finding the distance between two intervals essentially reduces to computing a number of interval intersection and containment queries and lowest common ancestor queries. A query can be carried out in constant time if the rectangle of the subdivision of H the lower end point of an interval belongs to is precomputed; otherwise it takes $O(\log n)$ time to locate the rectangles of the interval end points.

6 Smallest Path Queries in P

We are now able to put the results together and obtain the following algorithm to compute a smallest path between two points p_1 and p_2 in P.

Algorithm Smallest Path Query
Input: Two points p_1 and p_2 in P;
Output: A smallest path $\mathcal{P}$ from p_1 to p_2;

1. compute H_{p_1} and H_{p_2};
2. let $H_{lca} = lca(H_{p_1}, H_{p_2})$;
3. let $\rho_i = \text{depth}(H_{p_i}) - \text{depth}(H_{lca})$;
4. compute $I_i(\rho_i)$ and $I_i(\rho_i + 1)$, for $i = 1, 2$;
5. compute $rl(I_1(d_1), I_2(d_2))$, for $d_1 \in \{\rho_1, \rho_1 + 1\}$ and $d_2 \in \{\rho_2, \rho_2 + 1\}$;
6. compute $k(p_1, p_2)$;
7. let I_1 and I_2 be the two intervals associated to $k(p_1, p_2)$;
8. compute the path $\mathcal{P}_{I_1I_2}$and let r_1 be the end point of $\mathcal{P}_{I_1I_2}$ on I_1 and r_2 the end point of $\mathcal{P}_{I_1I_2}$ on I_2;
9. compute a smallest path $\mathcal{P}_i$ from p_i to r_i, for $i = 1, 2$;
10. let $\mathcal{P}$ be the concatenation of $\mathcal{P}_1$ with $\mathcal{P}_{I_1I_2}$ and $\mathcal{P}_2$;

The correctness of the algorithm can now be easily shown.

Lemma 6.1 *The path $\mathcal{P}$ constructed by Algorithm* Smallest Path Query *is a smallest path from p_1 to p_2.*

Proof: Clearly, $\mathcal{P}$ is a rl-shortest path by Lemma 3.4 since it has rl-length $\rho_1 + \rho_2 + k(p_1, p_2) - 2$. We now turn to proving that $\mathcal{P}$ is also a L_1-shortest path. Let q_i again be defined as the L_1-projection of p_i onto w_i. Observe that L_1-length$(\mathcal{P}_i) = L_1(p_i, q_i) + L_1(q_i, r_i)$ and L_1-length$(\mathcal{P}_{I_1 I_2}) + L_1(q_1, r_1) + L_1(q_2, r_2) = L_1(q_1, q_2)$ by Lemma 5.2; hence, we immediately obtain that $L_1(p_1, p_2) = L_1(p_1, r_1) + L_1(r_1, r_2) + L_1(r_2, p_2)$ which equals L_1-length$(\mathcal{P}_1) + L_1$-length$(\mathcal{P}_{I_1 I_2}) + L_1$-length$(\mathcal{P}_2)$ by Lemmata 4.7 and 5.2. $\square$

If we are only interested in the *distance* between the two points, then we can also use the above algorithm if we replace the last two steps as follows.

Algorithm Distance Query

$9'$. compute L_1-length$(\mathcal{P}_i)$, for $i = 1, 2$ and L_1-length$(\mathcal{P}_{I_1 I_2})$;
$10'a$. let $rl(p_1, p_2) = d_1 + d_2 + k(p_1, p_2) - 2$;
$10'b$. let $L_1(p_1, p_2) = L_1$-length$(\mathcal{P}_1) + L_1$-length$(\mathcal{P}_{I_1 I_2}) + L_1$-length$(\mathcal{P}_2)$.

The analysis of the Algorithm *Smallest Path Query* can be summarized as follows. Steps 2, 3, 5, 6, 7, 8, and 10 take constant time. In Steps 1 and 4 point location has to be done if we are given two arbitrary points. In order to do so we can preprocess P in linear time such that point location queries can be answered in time $O(\log n)$ [18]. Clearly, Step 9 can be carried out in time proportional to the link length k of the path and, thus, a smallest path query between two arbitrary points takes $O(\log n + k)$ time.

If, on the other hand, we take two vertices, point location can be avoided in Steps 1 and 4 by a linear time preprocessing phase as we pointed out before which implies that the query time is $O(1 + k)$.

Finally, since Step $9'$ can be carried out in constant time by the Algorithms *L_1-Distance to Histogram* and *Connecting the Intervals* and Steps $10'a$ and $10'b$ clearly need only constant time, Algorithm *Distance Query* can be executed in time $O(\log n)$ for arbitrary points and $O(1)$ for vertices.

We have shown the following theorem.

Theorem 6.2 *Let P be a simple rectilinear polygon with n edges. It is possible to preprocess P in linear time such that, for any two points p_1 and p_2 in P,*

(i) *a smallest path $\mathcal{P}$ between p_1 and p_2 can be reported in time $O(\log n + k)$ if k is the rl-length of $\mathcal{P}$ and the rl-distance as well as the L_1-distance between p_1 and p_2 can be reported in time $O(\log n)$.*

(ii) *if p_1 and p_2 are two vertices of P, then a smallest path $\mathcal{P}$ between p_1 and p_2 can be reported in time $O(1 + k)$ if k is the rl-length of $\mathcal{P}$ and the rl-distance as well as the L_1-distance between p_1 and p_2 can be reported in constant time.*

7 Applications

Though the data structure presented above has many potential applications, it is in particular interesting to look at the problem of computing the *diameter* of a simple rectilinear polygon. The *diameter* of a polygon P w.r.t. metric d is defined as the maximum value $d(p_1, p_2)$ of two points p_1 and p_2 in P. The diameter problem w.r.t. the rl-metric has been investigated before and a linear time algorithm was given.[22] There is also a linear time algorithm to compute the diameter w.r.t. the L_1-metric [23].

In the following we are concerned with the problem of computing the diameter of P w.r.t. the L_1-metric. Furthermore, we show how to compute a L_1-furthest neighbour for each vertex of P where a L_1-*furthest neighbour* of a point p is a point that has the maximal L_1-distance in P to p. In (Ref. [23]) it is shown that a L_1-furthest neigbour of a point p is a convex vertex. This implies in particular that the diameter of P is spanned by two convex vertices.

Computing L_1-furthest neighbours is equivalent to computing the maximum of each row of the $n \times n$ matrix $D = (d_{ij})$ with $d_{ij} = L_1(v_i, v_j)$ if $v_1, \ldots, v_n$ are the vertices of P ordered counterclockwise around the boundary of P. We have the following lemma.

Lemma 7.1 $d_{ij} + d_{kl} \leq d_{ik} + d_{jl}$, *for all* $1 \leq i < j < k < l \leq n$.

Proof: Let $1 \leq i < j < k < l \leq n$ and consider the vertices v_i, v_j, v_k, and v_l. Let $\mathcal{P}_{ik}$ be a L_1-shortest path from v_i to v_k. $\mathcal{P}_{ik}$ divides P

into two parts one of which contains v_j and the other of which contains v_l. Hence, a L_1-shortest path $\mathcal{P}_{jl}$ from v_j to v_l intersects $\mathcal{P}_{ik}$ in a point p_{int}. Let $d_m = L_1(v_m, p_{int})$, for $m \in \{i, j, k, l\}$. Then, $d_{ij} \leq d_i + d_j$ and $d_{kl} \leq d_k + d_l$ by the triangle inequality of L_1. Furthermore, since p_{int} is on the L_1-shortest path from v_i to v_k, we obtain $d_{ik} = d_i + d_k$ and, by a similar argument, $d_{jl} = d_j + d_l$ which taken together with the previous inequalities yields the claim. $\square$

Since $d_{ij} + d_{kl} \leq d_{ik} + d_{jl}$, for all $1 \leq i < j < k < l \leq n$, D can be transformed into a $n \times (2n-1)$ matrix $D' = (d'_{ij})$ with $d'_{i,i+j} = d_{ij}$, for $1 \leq i, j \leq n$ and $d'_{ij} = -\infty$, otherwise, which is *totally monotone*, i.e., if, $d_{ij} < d_{ij'}$, then $d_{i'j} < d_{i'j'}$, for $i < i'$ and $j < j'$ [24]. By the well-known matrix-searching technique introduced by Aggarwal *et al.* [24], it is possible to compute the row-wise maxima of D' in time $O(ne(n))$ if it takes $e(n)$ time to evaluate one entry of the matrix. Since $e(n) = O(1)$ with the above data structure, the matrix-searching technique yields a linear time algorithm to compute a L_1-furthest neighbours for all vertices of P. We summarize our considerations in the following theorem.

Theorem 7.2 *If P is a simple rectilinear polygon, then the L_1-diameter of P and a L_1-furthest neighbour for each vertex of P can be computed in time linear in the number of vertices of P.*

8 Conclusions

In this paper we presented a data structure for answering rectilinear link and L_1-distance queries between two points in a simple rectilinear polygon. The achieved query time as well as the preprocessing time and the space requirements are optimal. The computed path is shortest w.r.t. both metrics. If we leave the setting of a simple polygon, the query problem becomes considerably harder. There is no known subquadratic algorithm to solve even the one shot problem for a metric that is a linear combination of the rectiliner link and the L_1-metric among polygonal obstacles.

As an application we were able to show that the L_1-diameter of a polygon as well as a L_1-furthest neighbour of each vertex can be computed in linear time.

A natural generalization of the considered problem is to allow the paths to follow more than two orientations. It is not clear whether the presented technique can be extended to handle this problem as well. Very little is known about shortest paths with a fixed number of orientations.

9 Acknowledgements

This work was supported by the Deutsche Forschungsgemeinschaft under Grant No. Ot 64/8-1. I would like to thank Bengt Nilsson for the many helpful discussions. I would also like to thank the two unknown referees who took great care in reading previous versions of this paper and whose valuable comments and suggestions have greatly improved the presentation of the results as well as lead to the correction of some erroneous statements.

References

[1] P. Widmayer, "On Shortest Paths in VLSI design", Technical Report 19, Institut für Informatik, Universität Freiburg, Germany, 1990.

[2] K.L. Clarkson, S. Kapoor, and P.M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n \log^2 n)$ time", *Proc. 3rd Annual Symposium on Computational Geometry*, Waterloo, Ont., 1987, pp. 251–257.

[3] P.J. de Rezende, D.T. Lee, and Y.F. Wu, "Rectilinear shortest paths with rectangular barriers", *Journal of Discrete and Computational Geometry*, **4** (1989) 41–53.

[4] R.C. Larson and V.O. Li, "Finding minimum rectilinear distance paths in the presence of barriers", *Networks*, **11** (1981) 285–304.

[5] G. Das and G. Narasimhan, "Geometric searching and link distance", in *Algorithms and Data Structures, Proc. 2nd Workshop on Algorithms and Data Structures, WADS'91*, eds. F. Dehne, J.-R. Sack, F. Santoro, (Springer Verlag, Heidelberg, 1991) pp. 261–272.

[6] C. D. Yang, D. T. Lee, and C. K. Wong, "On bends and lengths of rectilinear paths: a graph-theoretic approach", in *Algorithms and Data Structures, Proc. 2nd Workshop on Algorithms and Data Structures, WADS'91*, eds. F. Dehne, J.-R. Sack, F. Santoro (Springer Verlag, Heidelberg, 1991) pp. 320–330.

[7] M. de Berg, M. van Kreveld, B. Nilsson, and M. Overmars, "Finding shortest paths in the presence of orthogonal obstacles using a combined L_1 and link metric", in *SWAT'90, Proc. 2nd Scandinavian Workshop on Algorithm Theory*, eds. J.R. Gilbert R. Karlsson (Springer Verlag, Heidelberg, 1990) pp. 213-224.

[8] M. Atallah and D. Chen, "On parallel rectilinear obstacle-avoiding paths", *Proc. 5th Canadian Conference on Computational Geometry*, Waterloo, Ont., Aug. 1993, pp. 210–215.

[9] H. ElGindy and P. Mitra, "Orthogonal shortest route queries among axes parallel rectangular obstacles", Technical Report No. SOCS 91.07 (July 1991), School of Computer Science, McGill University.

[10] M. de Berg, "On rectilinear link distance", *Computational Geometry: Theory and Applications*, **1** (1991) 13–34.

[11] J. Hershberger and J. Snoeyink, "Computing minimum length paths of a given homotopy class", in *Algorithms and Data Structures, Proc. 2nd Workshop on Algorithms and Data Structures, WADS'91*, eds. F. Dehne, J.-R. Sack, F. Santoro, (Springer Verlag, Heidelberg, 1991), pp. 331–342.

[12] A. Ferreira and J. Peters, "Finding smallest paths in rectilinear polygons on a hypercube multiprocessor", *Proc. 3rd Canadian Conference on Computational Geometry*, Vancouver, B.C., Aug. 1991, pp. 162–165.

[13] K. McDonalds and J. Peters. "Smallest Paths in Simple Rectilinear Polygons". *IEEE Transactions on Computer Aided Design* **11**, 1992, pp. 864–875.

[14] L. Guibas and J. Hershberger, "Optimal shortest path queries in a simple polygon", *Journal of Computer Systems and Science*, **39** (1989), pp. "50–63".

[15] E. Arkin, J. Mitchell, and S. Suri, "Optimal link path queries in a simple polygon", In *Proc. 3rd Symposium on Discrete Algorithms*, Jan. 1992, pp. 269-279.

[16] Christos Levcopoulos, "Heuristics for Minimum Decompositions of Polygons", PhD thesis, University of Linköping, Sweden, 1987.

[17] Bernard Chazelle, "Triangulating a simple polygon in linear time", *Discrete & Computational Geometry*, **6** 1991, 485–524.

[18] H. Edelsbrunner, L.J. Guibas, and J. Stolfi, "Optimal point location in a monotone subdivision", *SIAM Journal of Computing*, **15** (1986) 317–340.

[19] D. Harel and R. E. Tarjan, "Fast algorithms for finding nearest common ancestors", *SIAM Journal on Computing*, **13** (1984) 338–355.

[20] B. Schieber and U. Vishkin, "On finding lowest common ancestors: simplification and parallelization", *SIAM Journal of Computing*, **17** (1988) 1253–1262.

[21] O. Berkman and U. Vishkin, "Recursive*-tree parallel data-structure", *Proc. 30th Annual Symposium on Foundations of Computer Science*, Research Triangle Park, NC, Oct. 1989, pp. 196–202.

[22] B.J. Nilsson and S. Schuierer. Computing the rectilinear link diameter of a polygon. In *Computational Geometry–Methods, Algorithms, and Applications*, *Proc. Int. Workshop on Computational Geometry CG'91*, eds. H. Bieri H. Noltemeier (Springer Verlag, Heidelberg, 1991) pp. 203–216.

[23] S. Schuierer. *An Optimal Algorithm to Compute the L_1-Diameter and Center of a Simple Rectilinear Polygon.* Technical Report 49, Institut für Informatik, Universität Freiburg, 1994.

[24] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, "Geometric applications of a matrix searching technique", *Algorithmica*, **2** (1987) 195–208.