# Spanning Ratio and Maximum Detour of Rectilinear Paths in the $L_1$ Plane


Ansgar Grüne[1], Tien-Ching Lin[2], Teng-Kai Yu[2,3], Rolf Klein[1]
Elmar Langetepe[1], D.T. Lee[2,3], Sheung-Hung Poon[4]

[1]Institut für Informatik I, Universität Bonn, Bonn, Germany
[2]Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan
[3]Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
[4]Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
ansgar.gruene@googlemail.com, kero@iis.sinica.edu.tw, tkyu@ntu.edu.tw
rolf.klein@uni-bonn.de, elmar.langetepe@informatik.uni-bonn.de
dtlee@iis.sinica.edu.tw, spoon@cs.nthu.edu.tw



**Abstract.** The spanning ratio and maximum detour of a graph $G$ embedded in a metric space measure how well $G$ approximates the minimum complete graph containing $G$ and metric space, respectively. In this paper we show that computing the spanning ratio of a rectilinear path $P$ in $L_1$ space has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model and describe a deterministic $O(n \log^2 n)$ time algorithm. On the other hand, we give a deterministic $O(n \log^2 n)$ time algorithm for computing the maximum detour of a rectilinear path $P$ in $L_1$ space and obtain an $O(n)$ time algorithm when $P$ is a monotone rectilinear path.



**Key words:** rectilinear path, maximum detour, spanning ratio, dilation, $L_1$ metric, Manhattan plane


## 1 Introduction

Given a connected graph $G = (V, E)$ embedded in a metric space $M$, the *detour* between any two distinct points $p_i$, $p_j$ in $U = \bigcup_{e \in E} e$ is defined as

$$\delta_G(p_i, p_j) = \frac{d_G(p_i, p_j)}{\|p_i, p_j\|_M},$$

where $\|p_i, p_j\|_M$ denotes the distance between $p_i$ and $p_j$ in $M$ and $d_G(p_i, p_j)$ is the shortest path between $p_i$ and $p_j$ on $G$. The *maximum detour* $\delta(G)$ of $G$ is defined as the maximum detour over all pairs of distinct points in $U$, i.e.,

$$\delta(G) = \max_{p_i, p_j \in U, p_i \neq p_j} \delta_G(p_i, p_j).$$

If we restrict the points $p_i$, $p_j$ to the vertex set of $G$, then the maximum detour is also called *spanning ratio*, *dilation* or *stretch factor* $\sigma(G)$ of $G$, i.e.,

$$\sigma(G) = \max_{p_i, p_j \in V, p_i \neq p_j} \delta_G(p_i, p_j).$$

Given any connected graph embedded in any metric space, the spanning ratio can be computed in a straightforward manner by computing the all-pairs shortest paths of G. By using Dijkstra's algorithm [8] with Fibonacci heaps [9], we can find the spanning ratio in $O(n(m+n \log n))$ time and $O(n)$ space, where $n$ and $m$ are the numbers of vertices and edges, respectively.

Sometimes the geometric properties of special graph classes can be exploited to obtain a better upper bound [3, 13, 18]. If $G$ is a connected graph embedded in the Euclidean space $\mathbb{R}^2$, it is easy to see that the maximum detour is infinite if $G$ is non-planar. But if $G$ is planar, we can compute the maximum detour by first computing shortest paths for all pairs of vertices in $O(n^2 \log n)$ time (since $|E| = O(n)$) and then using this information to find the maximum detour between each pair of edges. Wulff-Nilsen [19] recently gave an algorithm for computing the maximum detour of a planar graph in $\mathbb{R}^2$ in $O(n^{\frac{3}{2}} \log^3 n)$ expected time. The case of $G$ being a planar polygonal chain is of particular interest. Agarwal et al. [1] gave an $O(n \log n)$ time randomized algorithm for computing the spanning ratio or maximum detour of a polygonal path in $\mathbb{R}^2$, and used it to obtain an $O(n \log^2 n)$ time randomized algorithm for computing the spanning ratio or maximum detour of cycles and trees in $\mathbb{R}^2$. They also claimed that it is possible to obtain a deterministic algorithm for computing the spanning ratio or maximum detour of a polygonal path in $O(n \log^c n)$ running time by parametric search, for some constant $c > 2$.

Ebbers-Baumann et al. [5] developed an $\varepsilon$-approximation algorithm that runs in $O(\frac{n}{\varepsilon} \log n)$ time for computing the maximum detour of a polygonal chain in $\mathbb{R}^2$. Narasimhan and Smid [15] studied the problem of approximating the spanning ratio of an arbitrary geometric connected graph in $\mathbb{R}^d$. They gave an $O(n \log n)$-time algorithm that computes a $(1 - \varepsilon)$-approximate value of the spanning ratio of a path, cycle, or tree in $\mathbb{R}^d$.

In this paper, we show that computing the spanning ratio of a rectilinear path $P$ in $L_1$ space has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model and describe a deterministic $O(n \log^2 n)$ time algorithm. This is the first sub-quadratic deterministic algorithm for computing the spanning ratio of a polygonal path embedded in a metric space avoiding complicated parametric search methods. We also give a deterministic $O(n \log^2 n)$ time algorithm for computing the maximum detour of a rectilinear path $P$ in $L_1$ space, and we obtain an optimal deterministic $O(n)$ time algorithm when $P$ is a monotone rectilinear path.

## 2 Preliminaries and Problem Definition

In this section we present the preliminaries and give the formal problem definitions. In the $L_1$ *plane* (also called Manhattan plane), the distance of two points $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ is defined as $\|p_i, p_j\|_{L_1} = d_{L_1}(p_i, p_j) = |x_i - x_j| + |y_i - y_j|$. A *path* $P = (V, E)$ of $n \geq 2$ vertices is a connected undirected graph, in which every vertex has degree two, except the two end vertices of degree one. If all of the edges of a path are either horizontal or vertical, we call this path a *rectilinear path*. In this paper, we will focus on rectilinear paths in which a vertex is either an end vertex or a corner vertex. A corner vertex $v \in V$ is a common vertex of a horizontal edge and a vertical edge and has degree 2. In general, vertices may not necessarily exist only at corners or at ends.

But the existence of non-corner and non-end vertices will not affect the correctness and complexities of our algorithms. Thus the algorithms presented in this paper can solve the problem for general rectilinear paths as well. Figure 2-1 (a) shows an example, where we can find that apart from the two end vertices of the rectilinear path, the other vertices are placed at corners.

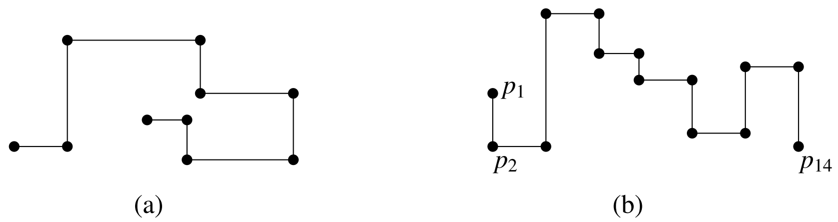



Figure 2-1 (a) A rectilinear path with all vertices at corners.
(b) A rectilinear path that is monotone with respect to the $x$-axis.

If a rectilinear path has non-decreasing $x$-coordinates from one of its end vertices to the other, we say that this path is *monotone* with respect to the $x$-axis. Monotone with respect to the $y$-axis can be defined similarly. Without loss of generality, we assume that monotone rectilinear paths in this paper are all monotone with respect to the $x$-axis. We refer to the vertices of an $n$-vertices monotone rectilinear path $P$ from its left end to its right end as $p_1, p_2, ..., p_n$. Figure 2-1 (b) shows an example of a $x$-monotone rectilinear path.

Consider a connected graph $G = (V, E)$ in the $L_1$ plane. The distance (weight) of an edge $e \in E$ is defined as the $L_1$ distance of its two incident vertices, and the distance of any two points $p_i$ and $p_j$ on $G$ (not necessarily in $V$) is defined as the length of the shortest path between them on $G$, denoted as $d_G(p_i, p_j)$.

In this paper we will compute the *spanning ratio* and *maximum detour* of a rectilinear path $P$ in the $L_1$ plane. The rest of the paper is organized as follows. In Section 3, we show a lower bound for computing the spanning ratio of a rectilinear path $P$ in the $L_1$ plane, even for the case when the path $P$ is monotone. Section 4 gives a deterministic $O(n \log^2 n)$ time algorithm for computing the spanning ratio of $P$. Section 5 gives an $O(n \log^2 n)$ time algorithm for computing the maximum detour of $P$ and an $O(n)$ time algorithm when the path is monotone. We conclude in Section 6.

## 3 The Lower Bound

In this section we show that computing the spanning ratio of a rectilinear path $P$ in the $L_1$ plane has a lower bound $\Omega(n \log n)$ in the algebraic computation tree model. The proof follows an idea of Grüne et al's presentation [10] at EuroCG'03 that has not yet been submitted for publication.

The Integer Element Distinctness Problem is to decide whether $n$ integers $y_1, y_2, \ldots, y_n$ are all distinct. It is known that this problem has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model [20]. We will show that we can transform an instance $y_1, y_2, \ldots, y_n$ of Integer Element Distinctness Problem into an instance $P = (V, E)$ in

$O(n)$ time. Let $y_{max} = \max_{1\le i\le n} y_i$ and $y_{min} = \min_{1\le i\le n} y_i$. If $y_{min}$ is negative, we add $|y_{min}| + 1$ to every number to make all numbers positive. We set the vertex set $V = \{p_{4i-3} = (\frac{2i-2}{2n}, \hat{y} + i), p_{4i-2} = (\frac{2i-1}{2n}, \hat{y} + i), p_{4i-1} = (\frac{2i-1}{2n}, y_i), p_{4i} = (\frac{2i}{2n}, y_i) \mid i = 1, 2, \dots, n\}$, where $\hat{y} = 3y_{max}+2n+1$ (the reason will be shown later), and the edge set $E = \{e_j = (p_j, p_{j+1}) \mid j = 1, 2, \dots, 4n - 1\}$. Then $P = (p_1, p_2, \dots, p_{4n})$ is a rectilinear path that is monotone with respect to the $x$-axis. We say a vertex $p_i$ in $P$ is a *low vertex* if its $y$-coordinate is smaller than $\hat{y}$, and a *high vertex* otherwise.

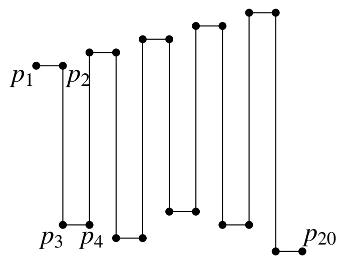



Figure 3-1 Transforming an instance of the integer element distinctness problem into a rectilinear path.

Figure 3-1 is an example of transforming an instance $(3, 2, 4, 3, 1)$ of Integer Element Distinctness Problem into a rectilinear path. By substituting $n = 5$, $i = 1$ and $y_1 = 3$ into the formula, we have $p_1 = (\frac{2i-2}{2n}, \hat{y} + i) = (0, \hat{y} + 1)$, and $p_2$, $p_3$ and $p_4$ are $(\frac{1}{10}, \hat{y}+1)$, $(\frac{1}{10}, 3)$ and $(\frac{2}{10}, 3)$, respectively. It is easy to see that the $y$-coordinates of high vertices are nondecreasing from left to right, but the $y$-coordinates of low vertices vary according to the values of $y_i$'s.

**Lemma 1.** *Let $p_i, p_{i+1}, p_j, p_{j+1}$ be four vertices in $P$, where $p_i$ and $p_{i+1}$ have the same $y$-coordinate, $p_j$ and $p_{j+1}$ have the same $y$-coordinate, and $i + 1 < j$. We have*

$$\delta_P(p_i, p_{j+1}) \le \delta_P(p_{i+1}, p_{j+1}) = \delta_P(p_i, p_j) \le \delta_P(p_{i+1}, p_j).$$

*Proof.* $\delta_P(p_i, p_j) = \frac{d_P(p_i,p_j)}{d_{L_1}(p_i,p_j)} = \frac{|\overline{p_ip_{i+1}}|+d_P(p_{i+1},p_j)}{|\overline{p_ip_{i+1}}|+d_{L_1}(p_{i+1},p_j)} \le \frac{d_P(p_{i+1},p_j)}{d_{L_1}(p_{i+1},p_j)} = \delta_P(p_{i+1}, p_j)$.
$\delta_P(p_{i+1}, p_{j+1}) = \frac{|\overline{p_jp_{j+1}}|+d_P(p_{i+1},p_j)}{|\overline{p_jp_{j+1}}|+d_{L_1}(p_{i+1},p_j)} = \frac{|\overline{p_ip_{i+1}}|+d_P(p_{i+1},p_j)}{|\overline{p_ip_{i+1}}|+d_{L_1}(p_{i+1},p_j)} = \delta_P(p_i, p_j)$.
$\delta_P(p_i, p_{j+1}) = \frac{d_P(p_i,p_{j+1})}{d_{L_1}(p_i,p_{j+1})} = \frac{|\overline{p_ip_{i+1}}|+d_P(p_{i+1},p_{j+1})}{|\overline{p_ip_{i+1}}|+d_{L_1}(p_{i+1},p_{j+1})} \le \frac{d_P(p_{i+1},p_{j+1})}{d_{L_1}(p_{i+1},p_{j+1})} = \delta_P(p_{i+1}, p_{j+1})$. □

Lemma 1 shows that for any four vertices in such a situation only $(p_{i+1}, p_j)$ can contribute to the spanning ratio. We call such a pair of vertices a *candidate pair*.

**Lemma 2.** *If a candidate pair $(p_i, p_j)$ have one low vertex and one high vertex, then there exists another candidate pair of vertices, both are high vertices or low vertices, such that their detour is larger than $\delta_P(p_i, p_j)$.*

*Proof.* Without loss of generality, we assume that $p_i$ is to the left of $p_j$, $p_i$ is a high vertex, and $p_j$ is a low vertex. Let vertex $p_k$ be the next high vertex to the right of $p_j$. Since $\hat{y} = 3y_{max} + 2n + 1 > y_{max} + n + 1$, we have

$$\delta_P(p_i,p_j) = \frac{d_P(p_i,p_j)}{d_{L_1}(p_i,p_j)} \le \frac{d_P(p_i,p_j)}{\hat{y}-y_{max}} < \frac{d_P(p_i,p_j)}{n+1} \le \frac{d_P(p_i,p_j)}{d_{L_1}(p_i,p_k)} \le \frac{d_P(p_i,p_k)}{d_{L_1}(p_i,p_k)} = \delta_P(p_i,p_k).$$

The case of $p_i$ being a low vertex and $p_j$ being a high vertex is similar. $\square$

**Lemma 3.** *If a candidate pair $(p_i, p_j)$ are both high or both low vertices with different y-coordinates, then $\delta_P(p_i, p_j) \le \frac{4n}{3}(\frac{1}{2} + \hat{y} + n - y_{min})$.*

*Proof.* Without loss of generality, we assume that $p_i$ is to the left of $p_j$. Let the distance between $p_i$ and $p_j$ along the $x$-axis be $\frac{2m-1}{2n}$.

$$\begin{aligned}\delta_P(p_i,p_j) = \quad & \frac{d_P(p_i,p_j)}{L_1(p_i,p_j)} \quad \le \quad \frac{\frac{2m-1}{2n}+2m(\hat{y}+n-y_{min})}{\frac{2m-1}{2n}+1} \quad = \quad \frac{\frac{2-\frac{1}{m}}{2n}+2(\hat{y}+n-y_{min})}{\frac{2-\frac{1}{m}}{2n}+\frac{1}{m}} \\ \le \quad & \frac{\frac{1}{n}+2(\hat{y}+n-y_{min})}{\frac{3}{2n}} \le \frac{4n}{3}(\frac{1}{2n}+\hat{y}+n-y_{min}) \le \frac{4n}{3}(\frac{1}{2}+\hat{y}+n-y_{min})\end{aligned}$$

Since $L_1(p_i, p_j) \ge \frac{2m-1}{2n} + 1$, $d_P(p_i, p_j) \le \frac{2m-1}{2n} + 2m(\hat{y} + n - y_{min})$ and $\frac{2-\frac{1}{m}}{2n} + \frac{1}{m} \ge \frac{1}{2n} + \frac{1}{m} \ge \frac{1}{2n} + \frac{1}{n} = \frac{3}{2n}$, we have $\delta_P(p_i, p_j) \le \frac{4n}{3}(\frac{1}{2n} + \hat{y} + n - y_{min}) \le \frac{4n}{3}(\frac{1}{2} + \hat{y} + n - y_{min})$. $\square$

**Lemma 4.** *If a candidate pair $(p_i, p_j)$ are both low vertices with the same y-coordinate, then $\delta_P(p_i, p_j) \ge 2n(\hat{y} - y_{max})$.*

*Proof.* Let the distance between $p_i$ and $p_j$ along $x$-axis be $\frac{2m-1}{2n}$. Then, $\delta_P(p_i, p_j) \ge \frac{2m(\hat{y}-y_{max})}{\frac{2m-1}{2n}} = \frac{2m(2n\hat{y}-2ny_{max})}{2m-1} \ge 2n(\hat{y} - y_{max})$. $\square$

Combining the above lemmas together, we now show that this problem has a lower bound of $\Omega(n \log n)$.

**Theorem 1.** *Computing the spanning ratio of a rectilinear path $P$ in the $L_1$ plane has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model, even if the given rectilinear path is x-monotone.*

*Proof.* By Lemma 2, the spanning ratio must occur at a candidate pair of two high or two low vertices. Substituting $\hat{y} = 3y_{max} + 2n + 1$ into the formulas of Lemma 3 and Lemma 4, we have

$$\begin{aligned}2n(\hat{y} - y_{max}) &= 2n(2y_{max} + 2n + 1) = 2n(\tfrac{2}{3}(\hat{y} - 2n - 1) + 2n + 1) \\ &= 2n(\tfrac{2}{3}\hat{y} + \tfrac{2}{3}n + \tfrac{1}{3}) > 2n(\tfrac{2}{3}\hat{y} + \tfrac{2}{3}n + \tfrac{1}{3}) - \tfrac{4n}{3}(y_{min}) = \tfrac{4n}{3}(\tfrac{1}{2} + \hat{y} + n - y_{min}).\end{aligned}$$

Therefore, if we choose $\hat{y} = 3y_{max}+2n+1$, then the spanning ratio $\delta(P) \ge 2n(2y_{max}+2n + 1)$ if and only if there exists a candidate pair of two low vertices with the same $y$-coordinate. The existence of a candidate pair of two low vertices with the same $y$-coordinate is equivalent to the existence of two numbers $y_i$ and $y_j$ (with $i \ne j$) of the same value in the given instance of the Integer Element Distinctness Problem. $\square$

## 4 Computing the Spanning Ratio of a Rectilinear Path

In this section we compute the spanning ratio of a rectilinear path $P$ in the $L_1$ plane. We define that a vertex $p_i = (p_i.x, p_i.y)$ is *dominated* by another vertex $p_j = (p_j.x, p_j.y)$ if $p_i.x \le p_j.x$ and $p_i.y \le p_j.y$, denoted by $p_i \preceq p_j$. For a vertex $p_i$ in $V$, let $p_i^*$ be the vertex in $V$ such that $\delta_P(p_i, p_i^*) = \max\{\delta_P(p_i, p_j) \mid p_j \in V\}$. We say that $p_i^*$ is the *best partner* of $p_i$ in $V$. Thus if we know the best partner of each vertex, then it is easy to compute the spanning ratio of $P$. It suffices to consider the detours from $p_i$ to the vertices to the right of it, i.e., to find the maximum detour from $p_i$ to the set $P_i = \{p_j \mid p_i.x \le p_j.x\}$. But the size of each $P_i$ could be $O(n)$ and the time complexity might become $O(n^2)$ if we find the best partner of each vertex $p_i$ in a brute force manner. In the following, we give an $O(n \log^2 n)$ time and $O(n)$ space algorithm. We divide the set $P_i$ into two subsets: $D_i^+ = \{p_j \mid p_i \preceq p_j\}$ and $D_i^- = P_i \setminus D_i^+$. We denote the best partner of $p_i$ in $D_i^+$ as $p_i^+$ and in $D_i^-$ as $p_i^-$. We only focus on $D_i^+$ here; the case of $D_i^-$ is similar. That is, we only need to find the $p_i^+$ for each $p_i$ in $D_i^+$. Without loss of generality, we assume that all vertices in $P$ are in the first quadrant.

To solve this problem, we transform the vertices of $V$ from the $L_1$ plane to the $L_2$ plane as follows. We transform each vertex $p_j$ in $V$ into a point $q_j = (q_j.x, q_j.y) = (d_{L_1}(o, p_j), d_P(p_1, p_j))$ in $\mathbb{R}^2$ in a one-to-one manner, where $o$ is the origin. For convenience, we call the original $L_1$ plane the *primal plane* and the transformed space the *dual plane*. In other words, in the dual plane $q_j$ has as its $x$-coordinate the $L_1$ distance between the origin $o$ and $p_j$ and as its $y$-coordinate the path length from $p_1$ to $p_j$. The point set $Q_i^+ = \{q_j \mid p_j \in D_i^+\}$ in the dual plane corresponds to the point set $D_i^+$ in the primal plane. Therefore, we have $\delta_P(p_i, p_i^+) = \max_{q_j \in Q_i^+} |m(i, j)|$, where $m(i, j) = \frac{q_j.y - q_i.y}{q_j.x - q_i.x}$. Thus the spanning ratio $\delta_P(p_i, p_i^+)$ occurs at either maximum $m(i, j)$ or minimum $m(i, j)$ among all $q_j$ in $Q_i^+$. This problem now is equivalent to finding the two tangent lines from $q_i$ to the convex hull of $Q_i^+$. Figure 4-1 shows an example. In Figure 4-1(a), $p_i$ has the dominating set $D_i^+ = \{p_a, p_b, p_c, p_d, p_e\}$. In Figure 4-1(b), we transform $p_i$ and $p_a, p_b, p_c, p_d, p_e$ into the dual plane. The maximum and minimum values of $m(i, j)$ can be found by the two tangent lines from $q_i$ to the convex hull of $Q_i^+ = \{q_a, q_b, q_c, q_d, q_e\}$.

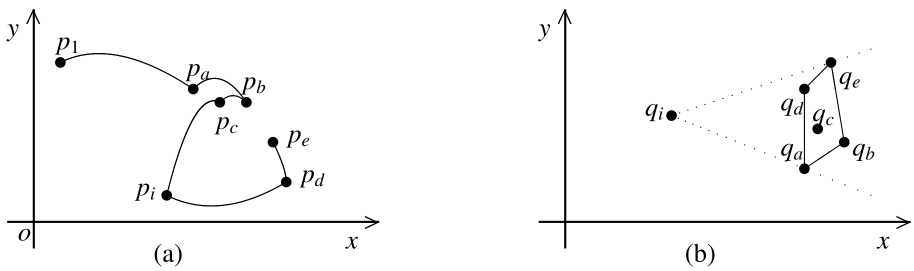



Figure 4-1 (a) A vertex $p_i$ and its $D_i^+ = \{p_a, p_b, p_c, p_d, p_e\}$. (b) Finding $\delta(p_i, p_i^+)$ in the dual plane by the two tangent lines from $q_i$ to the convex hull of $Q_i^+$.

Based on this transformation, if we can find $D_i^+$ for each $p_i$, we can find $p_i^+$ for each $p_i$ by making tangent queries from $q_i$ to the convex hull of $Q_i^+$. We observe that the tangent query is decomposable. A query is called *decomposable* if the answer to the query over the entire set can be obtained by combining the answers to the queries to a suitable collection of subsets of the set. We will partition $D_i^+$ into $\log n$ subsets by the divide-and-conquer method and make the tangent queries from $q_i$ to the convex hulls of the corresponding subsets in the dual plane and choose the one with maximum slope.

Our divide-and-conquer approach works as follows. Let $p_m$ be the vertex in $P$ such that $p_m.x$ is the median of the x-coordinates of all vertices in $P$. We divide the set $P$ into two subsets: $P_L = \{p_i \mid p_i.x \leq p_m.x\}$ and $P_R = \{p_j \mid p_j.x > p_m.x\}$. We then sort the vertices of $P_L$ and $P_R$ in descending $y$-coordinates respectively. We iterate on each vertex in $P_L$ in descending $y$-coordinate order such that we can find its best partner in $P_R$. Then we solve the subproblems $P_L$ and $P_R$ recursively. While iterating on each vertex in $P_L$ in descending $y$-coordinate order, assume that after iterating on the vertex $p_i$ in $P_L$ we have maintained a subset $D_R^+ = \{p_k \mid p_k \in P_R, p_i.y \leq p_k.y\}$ in the primal plane and the convex hull of the corresponding subset $Q_R^+ = \{q_k \mid p_k \in D_R^+\}$ in the dual plane. For the next iterating vertex $p_j$ in $P_L$, we first insert into $D_R^+$ those vertices in $P_R$ whose $y$-coordinates are between $p_i.y$ and $p_j.y$ and their corresponding points in the dual plane into the convex hull of $Q_R^+$ respectively and then make a tangent query from $q_j$ to the convex hull of $Q_R^+$.

Preparata [16] proposed an optimal algorithm for updating the convex hull in $O(\log n)$ time for the insertion only case. Hershberger and Suri [12] obtained an offline version of dynamic convex hull that can process a sequence of $n$ insertion, deletion, and query instructions in total $O(n \log n)$ time and $O(n)$ space. If we implement our convex hull by either of the dynamic convex hull data structures, we can afford tangent query or insertion in $O(\log n)$ time. Therefore, the total time complexity of our algorithm is $T(n) = 2T(\frac{n}{2}) + O(n \log n) = O(n \log^2 n)$.

**Theorem 2.** *The spanning ratio of a rectilinear path in the $L_1$ plane can be found in $O(n \log^2 n)$ time and $O(n)$ space.* □

## 5 Computing the Maximum Detour of a Rectilinear Path

In this section we compute the maximum detour of a rectilinear path $P = (V, E)$ in the $L_1$ plane. The maximum detour can occur on any two distinct points in $U = \bigcup_{e \in E} e$. In a previous work, Grüne et al. [10] presented an $O(n^2)$ algorithm for finding the maximum detour of a simple polygon P. There the detour between two points was defined as the ratio of the minimum length of all connecting paths contained in P, divided by the straight distance. They observed that linear time suffices for monotone rectilinear polygons in $L_1$. We also come to a linear time conclusion for monotone rectilinear paths; but for arbitrary rectilinear paths, we obtain an upper bound of only $O(n \log^2 n)$.

The following lemma is useful and will be used several times.

**Lemma 5.** *For any three points $p, q, r$ in $U = \bigcup_{e \in E} e$ with $p \preceq q$ and $q \preceq r$, we have $\delta_P(p, r) \leq \max\{\delta_P(p, q), \delta_P(q, r)\}$.*

*Proof.* There are three cases to be considered, depending on which one of $\{p, q, r\}$ lies between the two others on $P$, see Figure 5-1: (a) $d_P(p, r) = d_P(p, q) + d_P(q, r)$; (b) $d_P(p, q) = d_P(p, r) + d_P(r, q)$; (c) $d_P(q, r) = d_P(q, p) + d_P(p, r)$.

For case (a), we have $\delta_P(p, r) = \frac{d_P(p,r)}{d_{L_1}(p,r)} = \frac{d_P(p,q)+d_P(q,r)}{d_{L_1}(p,q)+d_{L_1}(q,r)} \leq \max\{\frac{d_P(p,q)}{d_{L_1}(p,q)}, \frac{d_P(q,r)}{d_{L_1}(q,r)}\}$
$= \max\{\delta_P(p, q), \delta_P(q, r)\}$.
For case (b), we have $\delta_P(p, r) = \frac{d_P(p,r)}{d_{L_1}(p,r)} \leq \frac{d_P(p,r)}{d_{L_1}(p,q)} \leq \frac{d_P(p,q)}{d_{L_1}(p,q)} = \delta_P(p, q)$.
For case (c), we have $\delta_P(p, r) = \frac{d_P(p,r)}{d_{L_1}(p,r)} \leq \frac{d_P(p,r)}{d_{L_1}(q,r)} \leq \frac{d_P(q,r)}{d_{L_1}(q,r)} = \delta_P(q, r)$. □

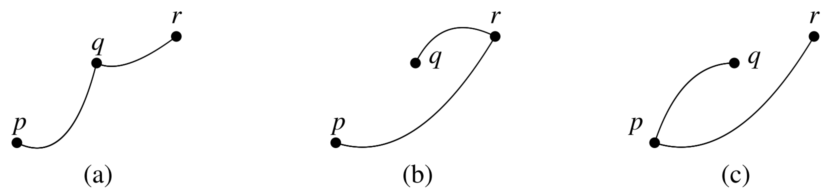



Figure 5-1 (a) $d_P(p, r)$=$d_P(p, q) + d_P(q, r)$ (b) $d_P(p, q)$=$d_P(p, r) + d_P(r, q)$

(c) $d_P(q, r)$=$d_P(q, p) + d_P(p, r)$

First in Section 5.1, we give an $O(n)$ time and $O(n)$ space algorithm to compute the maximum detour when the rectilinear path is monotone. We then present an $O(n \log^2 n)$ time and $O(n)$ space algorithm for the general case in Section 5.2.

### 5.1 Monotone rectilinear paths

Let us assume that $d_P(p_1, p_i)$, for $i = 2, 3, ..., n$, has been computed in $O(n)$ time. For any two distinct points on $P$, if the *open* straight line segment connecting them has no intersection with $P$, we say these two points are *visible* from each other; they form a *visible pair*.

**Lemma 6.** *At least one of the pairs of points on P contributing to the maximum detour must be a visible pair, and these two points must have the same y-coordinate.*

*Proof.* By Lemma 5, if $p, q \in P$ and the open segment $\overline{pq}$ intersects $P$ at $r$, then one of the two detours $\delta_P(p, r)$ and $\delta_P(r, q)$ must be no less than $\delta_P(p, q)$. Thus one of the pairs of points contributing to the maximum detour must be a visible pair.

For a visible pair of points $p, q \in P$, if $p.y \neq q.y$, then we will show that there exists a pair of points such that their detour larger than $\delta_P(p, q)$. Without loss of generality, we assume that the path on $P$ between $p$ and $q$ is below the segment $\overline{pq}$ and $p \preceq q$.

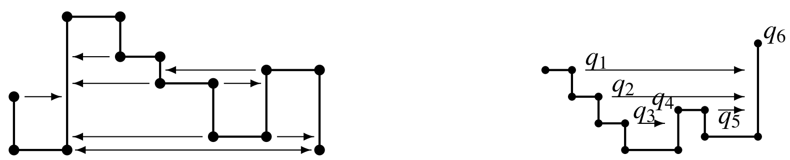
If there is a point $r$ on the path between $p$ and $q$ such that $p \preceq r$ and $r \preceq q$, we have either $\delta_P(p, r) \geq \delta_P(p, q)$ or $\delta_P(r, q) \geq \delta_P(p, q)$ by Lemma 5. We then either replace point $p$ by point $r$ if $\delta_P(r, q) \geq \delta_P(p, q)$ or replace point $q$ by point $r$ if $\delta_P(p, r) \geq \delta_P(p, q)$. If we repeat the above procedure on the path between $p$ and $q$ until there is no point $r$ on the path between $p$ and $q$ such that $p \preceq r$ and $r \preceq q$, we can then move point $q$ downward to a point $q'$ such that $q'.y = p.y$, and we have $\delta_P(p, q') \geq \delta_P(p, q)$. □

Given the lemma above, which says that two points defining the maximum detour must be visible from each other and have the same $y$-coordinate, we shall call such a pair *horizontally visible*.

**Lemma 7.** *For any horizontally visible pair on $P$ contributing to the maximum detour, at least one of these two points must be a vertex.*

*Proof.* We will show that for a horizontally visible pair $p, q \in P$, if both $p$ and $q$ are not a vertex, there exists a pair of points such that their detour larger than $\delta_P(p, q)$. Without loss of generality, we assume that $p$ is to the left of $q$ and the path on $P$ between $p$ and $q$ is below $\overline{pq}$. If we move $p$ and $q$ upward simultaneously while keeping their $L_1$ distance the same, their detour $\delta_P(p, q)$ will increase as the path length from $p$ to $q$ on $P$ increases. Thus we can keep moving $p$ and $q$ upward until one of them coincides with a vertex. □

Thus we can restrict our search of the candidate pairs of points to horizontally visible pairs, with a vertex in each pair. Thus the number of candidate pairs is no more than the number of vertices. Figure 5-2 (a) shows an example of all the candidate pairs on the path $P$. We use a *ray-shooting* method to find all the candidate pairs. We will shoot rays from each vertex to a target point horizontally visible from the vertex. Thus we can divide the valid rays into four types, according to the four types of vertices from which we shoot the rays, i.e., top-right, bottom-right, top-left, and bottom-left corner vertices.




(a) (b)

Figure 5-2 (a) Find all candidate pairs by ray-shooting.
(b) Shoot rays horizontally to the right from top-right vertices.

We only discuss the top-right corner case, as others are similar. Figure 5-2 (b) shows an example in which there are four rays shooting from four top-right corner vertices, $q_1, q_2, q_3$, and $q_5$. We use a stack $S$ to help calculate the detours of this type of candidate pairs. We traverse path $P$ from left to right. When we go downward and encounter a top-right vertex, we push the vertex into $S$. For the example in Figure 5-2 (b), we push $q_1$, $q_2$ and $q_3$ into $S$, respectively. When the path goes upwards and we encounter a vertex $q_i$, we pop the vertices lower than $q_i$ from $S$ and compute the detours associated with the horizontally visible pairs. For example in Figure 5-2 (b), when we encounter the vertex $q_4$, we pop $q_3$ and compute the detour $\delta_P(q_3, q)$ of the horizontally visible pairs $(q_3, q)$, where $q$ is the horizontal projection from $q_3$ on the vertical edge containing $q_4$. Since a vertex can be pushed into and popped from $S$ only once, the total time complexity for finding the maximum detour in a monotone rectilinear path is $O(n)$, and the space complexity is obviously $O(n)$.

**Theorem 3.** *The maximum detour of a n-vertex monotone rectilinear path in the $L_1$ plane can be found in $O(n)$ time and $O(n)$ space.* □

### 5.2 Non-Monotone Rectilinear path

Now we consider the case of a non-monotone rectilinear path $P$. The candidate pairs contributing to the maximum detour can be restricted to the following two cases. It can be proved similarly as in Lemmas 6 and 7.

**Lemma 8.** *Among the pairs of points contributing the maximum detour, there is one satisfies one of the following two properties: (1) it is a pair of visible vertices; (2) it is either a horizontally visible pair of points (with the same y-coordinate) or a vertically visible pair of points (with the same x-coordinate), and at least one of the two points must be a vertex.* □

We can use the algorithm shown in Section 4 to deal with case (1), which takes $O(n \log^2 n)$ time. For case (2), we need to do both vertical and horizontal ray-shooting from $V$ to $P$. The total number of rays is $O(n)$. We roughly describe the algorithm below. It can be done in $O(n \log n)$ time.

Consider shooting rays horizontally to the right from top-right and bottom-right corner vertices. We first sort the vertical edges by their $x$-coordinates, and then use a plane sweep method sweeping a vertical line from left to right. During the sweep, we maintain a binary search tree which consists of *active* corner vertices. An *active* corner vertex is one whose rightward ray has not yet been created. When scanning a new edge $e$, those vertices in the binary search tree whose $y$-coordinates lie between the $y$-coordinates of the two end vertices of $e$ will shoot their rightward rays to $e$, creating horizontally visible pairs of points. We then delete those vertices from the binary search tree and insert the two end vertices of edge $e$, if they are top-right or bottom-right corner vertices, into the binary search tree. Obviously, this algorithm takes time $O(n \log n)$. The other types of rays, horizontally to the left, vertically upward and vertically downward, can be handled in a similar way. Thus we can find all horizontally and vertically visible pairs of points in $O(n \log n)$ time. Therefore, the theorem follows.

**Theorem 4.** *The maximum detour of a n-vertex rectilinear path in the $L_1$ plane can be found in $O(n \log^2 n)$ time and $O(n)$ space.* □

## 6 Conclusion

We have shown that the problem of computing the spanning ratio of a rectilinear path $P$ in the $L_1$ plane has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model and we have given a deterministic $O(n \log^2 n)$ time algorithm. We have also given a deterministic $O(n \log^2 n)$ time algorithm for computing the maximum detour of a rectilinear path $P$ in the $L_1$ plane and have obtained an optimal $O(n)$ time algorithm for the monotone case.

There is still a gap between the lower bound $\Omega(n \log n)$ and upper $O(n \log^2 n)$ for the spanning ratio problem. How to bridge the gap will be of interest. As for the maximum detour problem for non-monotone rectilinear paths, we have not been able to make any use of the property that the maximum detour must be defined by a visible pair of points. Whether one can get a more efficient algorithm exploiting this or any other property is also of interest. Finally whether or not $\Omega(n \log n)$ is a lower bound for computing the maximum detour of a path remains open.

## 7 Acknowledgement

The authors would like to thank the anonymous referees for their careful reading of the paper and for their precise suggestions.

This work was supported in part by the National Science Council, Taiwan, under the Grants NSC 98-2221-E-001-007-MY3, NSC 98-2221-E-001-008-MY3, NSC 97-2221-E-007-054-MY3 and by a DFG-NSC Joint Research Project.

## References

1. P. K. Agarwal, R. Klein, C. Knauer, S. Langerman, P. Morin, M. Sharir, M. Soss: "Computing the Detour and Spanning Ratio of Paths, Trees, and Cycles in 2D and 3D", *Discrete Comput. Geom.*, 39, No. 1–3, pp. 17–37, 2008.
2. P. K. Agarwal, R. Klein, C. Knauer, M. Sharir: "Computing the Detour of Polygonal Curves", *TRB 02-03, Freie Universität Berlin, Fachbereich Mathematik und Informatik*, 2002.
3. O. Aichholzer, F. Aurenhammer, C. Icking, R. Klein, E. Langetepe, G. Rote: "Generalized Self-approaching Curves.", *Discrete Appl. Math.*, 109, pp. 3–24, 2001.
4. S. Alstrup and J. Holm: "Improved Algorithms for Finding Level Ancestors in Dynamic Trees", *27th International Colloquium on Automata Languages and Programming*, Geneva, Switzerland, LNCS 1853, pp. 73–84, 2000.
5. A. Ebbers-Baumann, R. Klein, E. Langetepe, A. Lingas: "A Fast Algorithm for Approximating the Detour of a Polygonal Chain". *Comput. Geom. Theory Appl.* 27, 123–134 (2004).
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: "Introduction to Algorithms, Second Edition", *MIT Press and McGraw-Hill*, 2001.
7. M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf: "Computational Geometry, Second Revised Edition". Springer-Verlag 2000. Section 5.3: Range Trees, pp.105–110.
8. E. W. Dijkstra. "A note on two problems in connexion with graphs". Numerische Mathematik 1: 269–271, 1959.
9. M. L. Fredman and R. E. Tarjan: "Fibonacci heaps and their uses in improved network optimization algorithms". *Journal of the ACM 34(3)*, 596–615, 1987.
10. A. Grüne: "Umwege in Polygonen". *Diploma Thesis*, Institute of Computer Science I, Bonn 2002.
11. J. Gudmundsson and C. Knauer: "Dilation and Detours in Geometric Networks". In *Handbook of Approximation Algorithm and Metaheuristics*, T. F. Gonzalez, Chapman & Hall/CRC, 2007, Section 52.
12. J. Hershberger and S. Suri: "Offline maintenance of planar configurations", *J. Algorithms*, 21, pp. 453–475, 1996.
13. C. Icking, R. Klein, E. Langetepe: "Self-approaching curves", *Math. Proc. Camb. Philos. Soc.*, 125, pp. 441–453, 1999.

14. S. Langerman, P. Morin, M. Soss: "Computing the Maximum Detour and Spanning Ratio of Planar Paths, Trees and Cycles". In: *Alt, H., Ferreira, A. (eds.) STACS 2002*. LNCS, vol. 2285, pp. 250–261. Springer, Heidelberg (2002)
15. G. Narasimhan and M. Smid: "Approximating the Stretch Factor of Euclidean Graphs", *SIAM J. Comput.*, 30(3), pp. 978–989, 2000.
16. F. P. Preparata: "An Optimal Real Time Algorithm for Planar Convex Hulls". *Comm. ACM 22*, 402–405, 1978.
17. F. P. Preparata, M. I. Shamos: "Computational Geometry: An Introducation", *Spriner-Verlag New York Inc.*, 1985.
18. G. Rote: "Curves with increasing chords", *Math. Proc. Camb. Philos. Soc.*, 115, pp. 1–12, 1994.
19. C. Wulff-Nilsen: "Computing the Maximum Detour of a Plane Graph in Subquadratic Time", *Algorithms and Computation, 19th International Symposium, ISAAC 2008*, LNCS 5369, pp. 740–751, 2008.
20. A. C.-C. Yao. Lower Bounds for Algebraic Computation Trees of Functions with Finite Domains. *SIAM Journal on Computing*, Volume 20 , Issue 4, 655–668, 1991.