COMPUTING THE STRETCH FACTOR AND MAXIMUM DETOUR OF PATHS, TREES, AND CYCLES IN THE NORMED SPACE

CHRISTIAN WULFF-NILSEN
Department of Computer Science, University of Copenhagen
koolooz@diku.dk

ANSGAR GRÜNE*, ROLF KLEIN† and ELMAR LANGETEPE‡
Institut für Informatik I, Universität Bonn, Bonn, Germany
*ansgar.gruene@googlemail.com
†rolf.klein@uni-bonn.de
‡elmar.langetepe@informatik.uni-bonn.de

D. T. LEE
Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan
Department of Computer Science and Engineering, National Chung-Hsing University
Taichung, Taiwan
dlee@nchu.edu.tw

TIEN-CHING LIN§
Department of Computer Science and Engineering, National Chung-Hsing University
Taichung, Taiwan
tienchinglin@nchu.edu.tw

SHEUNG-HUNG POON
Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan
spoon@cs.nthu.edu.tw

TENG-KAI YU
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan
tkyu@ntu.edu.tw

Received 14 October 2010
Revised 17 March 2012
Communicated by Otfried Cheong and Yoshio Okamoto, Guest Editors

§Corresponding author.
ABSTRACT

The stretch factor and maximum detour of a graph \( G \) embedded in a metric space measure how well \( G \) approximates the minimum complete graph containing \( G \) and the metric space, respectively. In this paper we show that computing the stretch factor of a rectilinear path in \( L_1 \) plane has a lower bound of \( \Omega(n \log n) \) in the algebraic computation tree model and describe a worst-case \( O(\sigma n \log^2 n) \) time algorithm for computing the stretch factor or maximum detour of a path embedded in the plane with a weighted fixed orientation metric defined by \( \sigma \geq 2 \) vectors and a worst-case \( O(n \log^d n) \) time algorithm to \( d \geq 3 \) dimensions in \( L_1 \)-metric. We generalize the algorithms to compute the stretch factor or maximum detour of trees and cycles in \( \Omega(\sigma n \log^{d+1} n) \) time. We also obtain an optimal \( O(n) \) time algorithm for computing the maximum detour of a monotone rectilinear path in \( L_1 \) plane.

Keywords: Rectilinear path; tree; cycle; stretch factor; maximum detour; spanning ratio; dilation; \( L_1 \) metric; weighted fixed orientation metric.

1. Introduction

Designing modern microchips is a complicated process involving several steps. One of these steps, the so-called routing step, deals with the problem of finding a layout of wires on the chip interconnecting a given set of pins. Many factors need to be taken into consideration when finding such a layout. An important measure is the total wire length. Minimizing this will help reduce heat generation, space on the chip, and signal delay. For this reason, Steiner minimal trees play an important role in VLSI design.

Due to manufacturing limitations, wires are typically restricted to having a finite set of fixed orientations. This has led to an interest in the so-called fixed orientation metrics, initially considered by Widmayer et al.,\(^3^2\) where the distance between a pair of pins is the length of a shortest path between them using a fixed set of orientations. Routing is typically performed in several layers with only one orientation allowed in each layer. Some layers may be more easily congested than others and thus, some orientations of wires may be less desirable than others.\(^3^5\) For this reason, fixed orientation metrics with a weight associated with each orientation have been considered. Steiner minimal trees in the plane equipped with the rectilinear metric and more recently in general weighted fixed orientation metrics have received a great deal of attention.\(^7^,^8^,^1^7^,^1^9^,^2^0^,^2^6^,^3^0\) A disadvantage of using Steiner minimal trees is that wire distance between some pairs of pins may be very large compared to the shortest possible distances. As a consequence, signal delay will be high between such pairs.

To obtain networks with small detours between any pair of pins, spanners have been considered. For \( t \geq 1 \), a \( t \)-spanner for a set of vertices in a network interconnecting the vertices such that the distance in the network between any pair of the given vertices is at most \( t \) times longer than the shortest possible distance between them. Computing networks with small spanner is an active area of research. For more results on spanners, see e.g. [12, 23, 25, 31]. The smallest \( t \) for which the network is a \( t \)-spanner is called the stretch factor of the networks. Two interesting
dual problems are the following: given a network interconnecting a set of vertices (points) in a metric space, what is the stretch factor (maximum detour) of this network?

Suppose we are given a connected graph $G = (V, E)$ embedded in a metric space $M$, the detour between any two distinct points $p, q$ in $U = \bigcup_{e \in E} e$ is defined as

$$\delta_G(p, q) = \frac{d_G(p, q)}{|p, q|_M},$$

where $|p, q|_M$ denotes the distance between $p$ and $q$ in $M$ and $d_G(p, q)$ is the length of a shortest path between $p$ and $q$ 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.,

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

If we restrict the points $p, q$ 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.,

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

Let $V$ denote a finite set of $\sigma \geq 2$ vectors $v_0, v_1, \ldots, v_{\sigma - 1}$. A weighted fixed orientation metric $d_v$ is defined as follows. Letting $v_{\sigma + i} = -v_i$ for $i = 0, 1, \ldots, \sigma - 1$, the unit circle $d_v$ is the boundary of the convex hull of $v_0, v_1, \ldots, v_{\sigma - 1}$. We assume that all vectors, regarded as points, are on the boundary since any vector that is not on the boundary can be discarded without changing the metric. Drawing lines through the $2\sigma$ vectors partitions the plane into $2\sigma$ wedge-shaped regions. For $i = 0, 1, \ldots, 2\sigma - 1$, the region $W_i$ defined by vectors $v_i$ and $v_{(i+1) \mod 2\sigma}$ is called the $i$-th V-cone. For a point $p$, we refer to $p + W_i$ as the $i$-th V-cone of $p$. Let $p, q \in \mathbb{R}^2$. It can be shown that if $q$ belongs to the $i$-th V-cone of $p$ then there exists a shortest path from $p$ to $q$ in the metric $d_v$ that consists of line segments each of which is parallel to either $v_i$ or $v_{(i+1) \mod 2\sigma}$. The $L_1$ metric in the plane is a special type of a fixed orientation metric, defined by vectors $(1, 0)$ and $(0, 1)$. More generally, in $(\mathbb{R}^d, L_1)$, the $L_1$ distance between two points $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ is $L_1(p, q) = \sum_{k=1}^d |q_k - p_k|$.

Surprisingly, the fastest known algorithm for computing the stretch factor of a Euclidean network is a naive one that computes all-pairs shortest paths. If the network is planar, all-pairs shortest paths can be computed in $O(n^2)$ time, giving an $O(n^2)$ time algorithm for computing the stretch factor of the network. For simpler types of graphs, faster algorithms exist. For instance, it has been shown that the stretch factor or maximum detour of a polygonal path in the Euclidean plane can be found in $O(n \log n)$ expected time and that the stretch factor or maximum detour of trees and cycles can be found in $O(n \log^2 n)$ expected time.$^1$ Using parametric search gives rather complicated $O(n \log^c n)$ worst-case time algorithm for computing the stretch factor or maximum detour of a polygonal path in the Euclidean plane for some constant $c > 2$. $^1$
In this paper, we show that computing the stretch factor of a rectilinear path in $L_1$ plane has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model\cite{15}. We consider the stretch factor problem and maximum detour problem in Section 3 and 4 respectively. We obtain an optimal deterministic $O(n)$ time algorithm for computing the maximum detour of a monotone rectilinear path, and describe a worst-case $O(\sigma n \log^2 n)$ time algorithm for computing the stretch factor or maximum detour of a path embedded in the plane with a weighted fixed orientation metric defined by $\sigma \geq 2$ vectors. This is the first sub-quadratic deterministic algorithm for computing the stretch factor or maximum detour of a polygonal path embedded in a metric space avoiding complicated parametric search methods. For the $L_1$-metric, we generalize the algorithm to $d \geq 3$ dimensions with running time $O(n \log^d n)$. At the cost of an extra $\log n$-factor in running time, we also show how to compute the stretch factor or maximum detour of trees and cycles. We conclude with some remarks and future work in Section 5.

2. The Lower Bound

In this section we show that computing the stretch factor 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. The proof follows an idea of Grüne.\cite{15}

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.\cite{15} We will show that we can transform an instance $y_1, y_2, \ldots, y_n$ of integer element distinctness problem into a rectilinear path $P = (V, E)$ in $O(n)$ time. Let $y_{\text{max}} = \max_{1 \leq i \leq n} y_i$ and $y_{\text{min}} = \min_{1 \leq i \leq n} y_i$. If $y_{\text{min}}$ is negative, we add $|y_{\text{min}}| + 1$ to every number to make all numbers positive.

We set the vertex set $V = \{(p_{4i-3} = (\frac{2i-2}{2n}, \hat{y} + i), p_{4i-2} = (\frac{2i-1}{2n}, \hat{y} + i), p_{4i-1} = (\frac{2i-1}{2n}, \hat{y}), p_{4i} = (\frac{2i}{2n}, \hat{y}), \mid i = 1, 2, \ldots, n\}$, where $\hat{y} = 3y_{\text{max}} + 2n + 1$ (the reason will be shown later), and the edge set $E = \{e_j = (p_j, p_{j+1}) \mid j = 1, 2, \ldots, 4n - 1\}$. Then $P = (p_1, p_2, \ldots, p_{4n})$ 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 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 $(\frac{1}{2n}, \hat{y} + 1), (\frac{1}{2n}, 3)$ and $(\frac{1}{2n}, 3)$, 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 $(p_i, p_{i+1})$ and $(p_j, p_{j+1})$ be two pairs of consecutive vertices in $P$ such that $p_i$ and $p_{i+1}$ have the same $y$-coordinate, $p_j$ and $p_{j+1}$ have the same $y$-coordinate, and $i + 1 < j$. Then it holds that

$$\delta p(p_i, p_{j+1}) \leq \delta p(p_{i+1}, p_{j+1}) = \delta p(p_i, p_j) \leq \delta p(p_{i+1}, p_j).$$
\[ p_1 = (0, \hat{y}+1) \]
\[ p_5 = \left(\frac{1}{10}, \hat{y}+2\right) \]
\[ p_6 \]
\[ p_3 = (\frac{1}{10}, 3) \]
\[ p_4 \]
\[ p_7 = \left(\frac{2}{10}, \hat{y}+2\right) \]
\[ p_{20} = (1, 1) \]

Fig. 1. Transforming an instance of the integer element distinctness problem into a rectilinear path.

**Proof.**

\[ \delta_P(p_i, p_j) = \frac{d_P(p_i, p_j)}{d_{L_1}(p_i, p_j)} \]
\[ = \frac{|p_i p_{i+1}| + d_P(p_{i+1}, p_j)}{|p_i p_{i+1}| + d_{L_1}(p_{i+1}, p_j)} \]
\[ \leq \frac{d_P(p_{i+1}, p_j)}{d_{L_1}(p_{i+1}, p_j)} \]
\[ = \delta_P(p_{i+1}, p_j). \]

\[ \delta_P(p_{i+1}, p_{j+1}) = \frac{|p_{i+1} p_{j+1}| + d_P(p_{i+1}, p_{j+1})}{|p_{i+1} p_{j+1}| + d_{L_1}(p_{i+1}, p_{j+1})} \]
\[ = \frac{|p_{i+1} p_{j+1}| + d_P(p_{i+1}, p_{j+1})}{|p_{i+1} p_{j+1}| + d_{L_1}(p_{i+1}, p_{j+1})} \]
\[ = \delta_P(p_{i+1}, p_{j+1}). \]

\[ \delta_P(p_i, p_{j+1}) = \frac{d_P(p_i, p_{j+1})}{d_{L_1}(p_i, p_{j+1})} \]
\[ = \frac{|p_i p_{i+1}| + d_P(p_{i+1}, p_{j+1})}{|p_i p_{i+1}| + d_{L_1}(p_{i+1}, p_{j+1})} \]
\[ \leq \frac{d_P(p_{i+1}, p_{j+1})}{d_{L_1}(p_{i+1}, p_{j+1})} \]
\[ = \delta_P(p_{i+1}, p_{j+1}). \]

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

**Lemma 2.** If a candidate pair \((p_i, p_j)\) has one low vertex and one high vertex, then there exists another candidate pair of two non-consecutive vertices, both high vertices or both low vertices whose 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
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 \((p_i, p_j)\) consists of two non-consecutive high or two non-consecutive low vertices with different \( y \)-coordinates, then

\[
\delta_P(p_i, p_j) \leq \frac{4n}{3} \left( \frac{1}{2} + \hat{y} + n - y_{\text{min}} \right).
\]

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 \( \frac{2m+1}{2n} \). Then

\[
\delta_P(p_i, p_j) = \frac{d_P(p_i, p_j)}{L_1(p_i, p_j)} \leq \frac{(2m-1)/2n + 2m(\hat{y} + n - y_{\text{min}})}{(2m-1)/2n + 1} = \frac{(2 - 1/m)/2n + 2(\hat{y} + n - y_{\text{min}})}{(2 - 1/m)/2n + (1/m)} \leq \frac{1/n + 2(\hat{y} + n - y_{\text{min}})}{3/2n} \leq \frac{4n}{3} \left( \frac{1}{2} + \hat{y} + n - y_{\text{min}} \right).
\]

Since \( L_1(p_i, p_j) \geq \frac{2n-1}{2n} + 1, d_P(p_i, p_j) \leq \frac{2m-1}{2n} + 2m(\hat{y} + n - y_{\text{min}}) \) and \( \frac{2-1/m}{2n} + \frac{1}{m} \geq \frac{1}{2n} + \frac{1}{n} = \frac{1}{2n} \), we have

\[
\delta_P(p_i, p_j) \leq \frac{4n}{3} \left( \frac{1}{2n} + \hat{y} + n - y_{\text{min}} \right) \leq \frac{4n}{3} \left( \frac{1}{2} + \hat{y} + n - y_{\text{min}} \right).
\]

Lemma 4. If a candidate pair \((p_i, p_j)\) consists of two non-consecutive low vertices with the same \( y \)-coordinate, then \( \delta_P(p_i, p_j) \geq 2n(\hat{y} - y_{\text{min}}) \).

Proof. Let the distance between \( p_i \) and \( p_j \) along the \( x \)-axis be \( \frac{2m-1}{2n} \). Then,

\[
\delta_P(p_i, p_j) \geq \frac{2m(\hat{y} - y_{\text{max}})}{2n-1 \frac{2m-1}{2n}} = \frac{2m(2n\hat{y} - 2ny_{\text{max}})}{2n-1} \geq 2n(\hat{y} - y_{\text{max}}).
\]

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

Theorem 1. Computing the stretch factor 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 stretch factor must occur at a candidate pair of two non-consecutive high or two non-consecutive low vertices. Substituting \( \hat{y} \) by
3y_{\text{max}} + 2n + 1 into the formula of Lemma 4 and Lemma 3, we have
\[
2n(y - y_{\text{max}}) = 2n(2y_{\text{max}} + 2n + 1) = 2n\left(\frac{2}{3}(y - 2n - 1) + 2n + 1\right)
\]
\[
= 2n\left(\frac{2}{3}y + \frac{2}{3}n + \frac{1}{3}\right) > 2n\left(\frac{2}{3}y + \frac{2}{3}n + \frac{1}{3}\right) - \frac{4ny_{\text{min}}}{3}
\]
\[
= \frac{4n}{3}\left(\frac{1}{2} + y - n - y_{\text{min}}\right).
\]

Therefore, it holds that \(\delta(P) \geq 2n(2y_{\text{max}} + 2n + 1)\) if and only if there exists a candidate pair of two non-consecutive low vertices with the same \(y\)-coordinates. The existence of such a candidate pair is, in turn, equivalent to the existence of two numbers \(y_i\) and \(y_j\) (with \(i \neq j\)) of the same value in the given instance of the integer element distinctness problem.

\[\square\]

3. Computing the Stretch Factor

In this section we compute the stretch factor of a path, tree and cycle in weighted fixed orientation metric and higher dimensions in \(L_1\)-metric. We first compute the stretch factor of a rectilinear path \(P\) in \(L_1\) plane. We define that a vertex \(p_i = (x_{p_i}, y_{p_i})\) is dominated by another vertex \(p_j = (x_{p_j}, y_{p_j})\) if \(x_{p_i} \leq x_{p_j}\) and \(y_{p_i} \leq y_{p_j}\), 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 we can compute the stretch factor 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 \(P_i = \{p_j \mid x_{p_j} \geq x_{p_i}\}\). But the size of each set \(P_i\) 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 point \(p_i^+\) for each point \(p_i\) 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 \(P\) to the \(L_1\) plane to the \(L_2\) plane as follows. We transform each vertex \(p_j\) in \(P\) to a point \(q_j = (x_{q_j}, y_{q_j}) = (d_{L_1}(o, p_j), d_{L_1}(p_1, p_j))\) 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
\[
m(i, j) = \frac{y_{q_j} - y_{q_i}}{x_{q_j} - x_{q_i}}.
\]

Thus the stretch factor \(\delta_P(p_i, p_i^+)\) occurs at either maximum \(m(i, j)\) or minimum \(m(i, j)\) among all \(q_j\) in \(Q_i^+\). This problem now is equivalent
proposed an optimal algorithm for updating the convex hull in
obtained an offline
September 5, 2012 16:57 WSPC/Guidelines S0218195912600035

Fig. 2. (a) A vertex \( p_i \) and the set \( D_i^+ = \{ p_a, p_b, p_c, p_d, p_e \} \). (b) Finding \( \delta(p_i, p_i^+) \) in the dual plane by the two tangent lines from \( q_i \) to the convex hull of \( Q_i^+ \).

to finding the two tangent lines from \( q_i \) to the convex hull of \( Q_i^+ \). Figure 2 shows an example. In Fig. 2(a), \( p_i \) has the dominating set \( D_i^+ = \{ p_a, p_b, p_c, p_d, p_e \} \). In Fig. 2(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 \( Q_i^+ = \{ q_a, q_b, q_c, q_d, q_e \} \).

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 \( \log n \) canonical 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 \( x_{p_m} \) is the median of the \( x \)-coordinates of all vertices in \( P \). We divide the set \( P \) into two subsets, \( P_L = \{ p_i \mid x_{p_i} \leq x_{p_m} \} \) and \( P_R = \{ p_j \mid x_{p_j} > x_{p_m} \} \). We then sort the vertices of \( P_L \) and \( P_R \) according to \( y \)-coordinates respectively. We iterate on each vertex in \( P_L \) in order of 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 order of descending \( y \)-coordinate, assume that after iterating on the vertex \( p_i \) in \( P_L \) we have maintained a subset \( D_i^+ = \{ p_k \mid p_k \in P_R, y_{p_k} \leq y_{p_i} \} \) in the primal plane and the convex hull of the corresponding subset \( Q_i^+ = \{ q_k \mid p_k \in D_i^+ \} \) in the dual plane. For the next iterating vertex \( p_j \) in \( P_L \), we first insert into \( D_i^+ \) those vertices in \( P_R \) whose \( y \)-coordinates are between \( y_{p_i} \) and \( y_{p_j} \) and their corresponding points in the dual plane into the convex hull of \( Q_i^+ \) and then make a tangent query from \( q_j \) to the convex hull of \( Q_i^+ \).

Hershberger and Suri\textsuperscript{18} obtained an offline version of a dynamic convex-hull data structure 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 these two 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 $T(n) = 2T(\frac{n}{2}) + O(n \log n) = O(n \log^2 n)$.

We now show how to compute the stretch factor of an $n$-vertex rectilinear path $P$ in metric space $(\mathbb{R}^d, L_1)$, $d \geq 3$, in $O(n \log^d n)$ time. We generalize the above algorithm to higher dimensions. To find the dominating set $D^+_i$ for each vertex $p_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ in $\mathbb{R}^d$, it is equivalent to making an orthogonal range query of the form $R_i = [x_{i1}, \infty) \times [x_{i2}, \infty) \times \ldots \times [x_{id}, \infty)$ on vertex set $P$. Without loss of generality, we assume that all vertices in $P$ are in the first hyperoctant. We again transform each vertex $p_j = (x_{j1}, x_{j2}, \ldots, x_{jd})$ in primal space $(\mathbb{R}^d, L_1)$ into a point $q_j = (d_{L_1}(o, p_j), d_P(p_1, p_j))$ in dual space $\mathbb{R}^2$ where $o$ is the origin. We will use a higher-dimensional range tree in primal space augmented by a dynamic convex-hull data structure of corresponding points in dual space. Just like constructing a higher-dimensional range tree, we divide the set $P$ into two $(d - 1)$-dimensional subsets by its median of the first coordinate. These $(d - 1)$-dimensional subsets are constructed in the same way. The recursion stops when we are left with vertices restricted to their last coordinate; these are stored in a balanced binary search tree in descending order of their last coordinate. When we make a range query on region $R_i$ in a higher-dimensional range tree, it is divided into $\log^{d-1} n$ canonical subsets. We then can make a tangent query from $q_i$ to its corresponding convex hull for each canonical subset and choose the one with maximum slope. The following theorem summarizes our discussion.

**Theorem 2.** The stretch factor of a rectilinear path in $(\mathbb{R}^d, L_1)$ can be computed in $O(n \log^d n)$ time and $O(n)$ space.

We generalize the above algorithm to weighted fixed orientation metric $d_w$. The idea is simple: we apply a certain linear transformation to the vertices of $P$ so that $i$th $V$-cones are mapped to first quadrants and then use the algorithm. This is done for all $V$-cones, giving an algorithm in $O(\sigma n \log^2 n)$ time.

**Theorem 3.** The stretch factor of a path in the $V$-cones can be computed in $O(\sigma n \log^2 n)$ time and $O(n)$ space.

We now generalize our algorithms for paths to more complicated graphs, trees, and cycles. We will show how to compute the stretch factor of an $n$-vertex tree or cycle in $O(\sigma n \log^3 n)$ time for the weighted fixed orientation metric $d_w$ and in $O(n \log^{d+1} n)$ time for higher dimensions in the $L_1$-metric. We will restrict our attention to the space $(\mathbb{R}^2, L_1)$, since both the weighted fixed orientation metric and higher dimension can be handled in a similar way. We extend the definition of the detour from vertices to any two subgraphs $G_1$ and $G_2$ of a graph $G$, by putting

$$\delta_G(G_1, G_2) = \max_{p \in G_1, q \in G_2, p \neq q} \delta_{G}(p, q).$$
Let $T$ be an $n$-vertex tree embedded in $(\mathbb{R}^2, L_1)$. We use the idea of Grimm\cite{Grimm} to compute the stretch factor of $T$. First, $T$ is partitioned into two subtrees $T_1$ and $T_2$ sharing a single vertex $p$ such that each subtree contains between $n/4$ and $3n/4$ vertices. As shown by Langemer et al.,\cite{Langemer} this can be done in $O(n)$ time. Then the stretch factor $\delta(T)$ of $T$ can be computed by computing $\delta(T_1, T_2)$, $\delta(T_1)$ and $\delta(T_2)$ respectively and $\delta(T_1)$ and $\delta(T_2)$ can be found recursively. To compute $\delta(T_1, T_2)$, we can view each subtree $T_i$ as a subpath $P_i$ from its subroot to each vertex of $T_i$, and the whole tree as a path connected by a pseudo edge of the two subroots with edge weight defined by path length of the two subroots in $T$. Then, $\delta(T_1, T_2)$ can be computed by our algorithms for paths.

Let us now consider an $n$-vertex cycle $C$ embedded in $(\mathbb{R}^2, L_1)$. This case is more difficult because there are two paths along $C$ between any pair of vertices. For two points $a, b \in C$, let $C[a, b]$ denote the subsets of $C$ from $a$ to $b$ in counterclockwise direction and let $d_C(a, b)$ denote the length of $C[a, b]$; thus, in general, $d_C(a, b) \neq d_C(b, a)$. For a point $s \in C$, let $\pi(s)$ denote the point on $C$ such that $d_C(s, \pi(s)) = d_C(\pi(s), s) = |C|/2$. Let $s$ be a point on $C$, and let $A, B$ be two subsets of $C[s, \pi(s)]$, then it is easy to see that $\delta_C(A, B) = \delta_{C[s, \pi(s)]}(A, B)$. Now the detoure between two vertices $p, q \in C$ is defined as

$$\delta_C(p, q) = \min\{d_C(p, q), d_C(q, p)\}.$$ 

To compute the stretch factor of $C$, we first find a point $r_1$ such that $C[r_1, \pi(r_1)]$ and $C[\pi(r_1), r_1]$ both contain between $n/4$ and $3n/4$ vertices. Then, the stretch factor $\delta(C)$ of $C$ can be found by computing $\delta(C[r_1, \pi(r_1)])$, $\delta(C[\pi(r_1), r_1])$ and $\delta(C[r_1, \pi(r_1)], C[\pi(r_1), r_1])$ respectively. The two values $\delta(C[r_1, \pi(r_1)])$, $\delta(C[\pi(r_1), r_1])$ can be computed directly by our algorithms for paths. Without loss of generality we may assume that $C[r_1, \pi(r_1)]$ contains more vertices than $C[\pi(r_1), r_1]$. Then, we can find a point $r_2$ on $C[r_1, \pi(r_1)]$ such that $C[r_1, r_2]$ and $C[r_2, \pi(r_1)]$ both contain between $1/4$ and $3/4$ vertices of $C[r_1, \pi(r_1)]$. Then, $\delta(C[r_1, \pi(r_1)], C[\pi(r_1), r_1])$ can be found recursively by computing $\delta(C_1, C_2)$, $\delta(C_3, C_4)$, $\delta(C_1, C_3)$ and $\delta(C_2, C_4)$, where $C_1 = C[r_1, r_2]$, $C_2 = C[r_2, \pi(r_1)]$, $C_3 = C[\pi(r_1), r_1]$, and $C_4 = C[\pi(r_2), r_1]$. $\delta(C_2, C_3)$, $\delta(C_4, C_1)$ can be computed directly by our algorithms for paths by connecting $C_2$, $C_3$ as a path and connecting $C_4$, $C_1$ as a path respectively. $\delta(C_1, C_3)$ and $\delta(C_2, C_4)$ can be found recursively.

**Theorem 4.** The stretch factor of a tree or cycle in the $V$-cones can be computed in $O(n \log^3 n)$ time and $O(n)$ space.

**Theorem 5.** The stretch factor of a rectilinear tree or cycle in $(\mathbb{R}^d, L_1)$ can be computed in $O(n \log^{d+1} n)$ time and $O(n)$ space.

4. Computing the Maximum Detour

In this section we show how to compute the maximum detour of an $n$-vertex path, tree and cycle for the weighted fixed orientation metric $d_\nu$ and higher dimensions in
Let us assume that \( d_P(p_i, p_i) \) 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 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 4(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.
We only discuss the top-right corner case, as others are similar. Figure 4(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 Fig. 4(b), we push $q_1$, $q_2$ and $q_3$ into $S$. 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 Fig. 4(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 pair $(q_3, q)$, 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 computing the maximum detour in a monotone rectilinear path is $O(n)$, and the space complexity is obviously $O(n)$. Hence, we have the following result.

**Theorem 6.** The maximum detour of an $n$-vertex monotone rectilinear path in the $L_1$ plane can be computed in $O(n)$ time and $O(n)$ space.

### 4.2. 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 defining the maximum detour, there is one that 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 stretch factor algorithm shown in Section 3 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 implemented to run 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, going horizontally to the left, vertically upward or 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, we can conclude as follows.

Theorem 7. The maximum detour of a path in the $V$-cones can be computed in $O(\sigma n \log^2 n)$ time and $O(n)$ space.

Theorem 8. The maximum detour of a path in $(\mathbb{R}^d, L_1)$ can be computed in $O(n \log^d n)$ time and $O(n)$ space.

Theorem 9. The maximum detour of a tree or cycle in the $V$-cones can be computed in $O(\sigma n \log^3 n)$ time and $O(n)$ space.

Theorem 10. The maximum detour of a tree or cycle in $(\mathbb{R}^d, L_1)$ can be computed in $O(n \log^{d+1} n)$ time and $O(n)$ space.

5. Conclusion

We have shown that the problem of computing the stretch factor of a rectilinear path in the $L_1$ plane has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model, and we compute the stretch factor and maximum detour of paths, trees and cycles in weighted fixed orientation metric and $(\mathbb{R}^{d+1}, L_1)$. Moreover, we obtain an optimal $O(n)$ time algorithm for computing the maximum detour of a monotone rectilinear path in $L_1$ plane.

There is still a gap between the lower bound of $\Omega(n \log n)$ and upper bound of $O(n \log^2 n)$ for the stretch factor problem in $L_1$ plane. 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 in $L_1$ plane remains open.
Acknowledgments

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


