On the Dilation Spectrum of Paths, Cycles, and Trees[*]

Rolf Klein[†] Christian Knauer[‡] Giri Narasimhan[§] Michiel Smid[¶]

March 11, 2009

Abstract

Let G be a graph with n vertices which is embedded in Euclidean space $\mathbb{R}^d$. For any two vertices of G, their *dilation* is defined to be the ratio of the length of a shortest connecting path in G to the Euclidean distance between them. In this paper, we study the spectrum of the dilation, over all pairs of vertices of G. For paths, cycles, and trees in $\mathbb{R}^2$, we present $O(n^{3/2+\epsilon})$–time randomized algorithms that compute, for a given value $\kappa > 1$, the exact number of vertex pairs of dilation at most κ. Then we present deterministic algorithms that approximate the number of vertex pairs of dilation at most κ to within a factor of $1 + \epsilon$. They run in $O(n \log^2 n)$ time for paths and cycles, and in $O(n \log^3 n)$ time for trees, in any constant dimension d.

Keywords: Computational geometry, dilation, distribution, geometric graph, network, spectrum, stretch factor.

[*]A preliminary version of this paper was presented at ISAAC'05 [11]. RK was supported by DFG Kl 655/14; MS was supported by NSERC.

[†]Institute of Computer Science I, Universität Bonn, Bonn, Germany.

[‡]Institute of Computer Science, Freie Universität Berlin, Berlin, Germany.

[§]School of Computing and Information Sciences, Florida International University, Miami, Florida, U.S.A.

[¶]School of Computer Science, Carleton University, Ottawa, Ontario, Canada.

1 Introduction

Let S be a set of n points in $\mathbb{R}^d$, where $d \geq 1$ is a constant, and let G be an undirected connected graph having the points of S as its vertices. The length of any edge (p, q) of G is defined to be the Euclidean distance $|pq|$ between the two vertices p and q. We say that G is a *Euclidean graph.* The length of a path in G is defined to be the sum of the lengths of its edges. For two vertices p and q, let $d_G(p, q)$ denote the minimum length of any path in G that connects p and q. For two distinct vertices p and q, their *dilation* in G is defined to be the quantity

$$\delta_G(p, q) := \frac{d_G(p, q)}{|pq|}.$$

Euclidean graphs are frequently used for modeling traffic or transportation networks. One measure for their performance is the *dilation* of G, as introduced in [17], which is defined to be the maximum dilation over all pairs of distinct vertices in G, i.e.,

$$\sigma(G) := \max\{\delta_G(p, q) : p, q \in S, p \neq q\}. \tag{1}$$

This value is also called the stretch factor, the spanning ratio, or the distortion [13] of G.

A lot of work has been done on the construction of good spanners, i.e., sparse graphs of low dilation that connect a given vertex set and enjoy other desirable properties; see the handbook chapter [7] or the monograph [16].

The problem of computing the dilation of a given Euclidean graph has first been addressed in [15]. They gave an $O(n \log n)$–time algorithm for approximating, up to a $1 - \epsilon$ factor, the dilation of paths, trees, and cycles in Euclidean space of constant dimension. In [1], exact randomized algorithms were given that run in $O(n \log n)$ expected time for paths in $\mathbb{R}^2$, and in $O(n \log^2 n)$ expected time for trees or cycles in $\mathbb{R}^2$. In $\mathbb{R}^3$, $O(n^{4/3+\epsilon})$ expected time is sufficient for either type of graph. Recent progress has been made by the second author for the dilation of Euclidean graphs having bounded treewidth; see [2]: For any fixed constant $k \geq 2$, the dilation of a Euclidean graph in $\mathbb{R}^2$ with n vertices and treewidth at most k can be computed in $O(n \log^{k+1} n)$ expected time.

In [19], randomized algorithm were presented for computing the *detour* of plane graphs and graphs of bounded treewidth; the detour of a graph G is the maximum of the quantity $\delta_G(p, q)$, where p and q are vertices or interior points of edges. For general Euclidean graphs, the best algorithm to compute the dilation seems to be running Dijkstra's algorithm for each vertex of G, which leads to an $O(mn + n^2 \log n)$–time algorithm [6]; here, n and m denote the number of vertices and edges of G, respectively.

In the recent papers [8, 14], the authors considered the problem of inserting an edge that maximizes the reduction in the dilation of a Euclidean graph. Interestingly, it was observed in [12] that this problem becomes NP-hard if $k \geq 1$ extra edges can be inserted; see also [5].

In this paper, we study the vertex-to-vertex dilation of Euclidean graphs from a different perspective. The dilation, as defined in (1), only gives the pair of vertices for which the dilation is maximized; it says nothing about how the rest of the network behaves with respect to dilation. In real networks, one may tolerate a high dilation for a limited number of pairs of vertices, as long as the dilation is bounded for a majority of the vertex-pairs. Therefore, we focus our attention on computing the *dilation spectrum* of a graph G. That is, for a given threshold value $\kappa > 1$, we are interested in the number

$$\pi_G(\kappa) := |\{\{p, q\} \in \binom{S}{2} : \delta_G(p, q) \leq \kappa\}|,$$

where $\binom{S}{2}$ is the set of all 2-element subsets of S. Thus, $\pi_G(\kappa)$ is the number of pairs of distinct vertices whose dilation does not exceed κ. The corresponding distribution of the dilation, i.e., the sequence $\pi_G(\kappa)$ for different values of κ, could be helpful in understanding structural properties of the given geometric graph G.

Clearly, the cost $O(mn + n^2 \log n)$ of running Dijkstra's algorithm from each vertex of G is an upper bound on the time complexity of computing the dilation spectrum of G. For some classes of graphs, better running times can be obtained. For example, it has been shown in [9] that, for any plane graph G, the shortest-path distances in G between all pairs of vertices can be computed in $O(n^2)$ total time. The same upper bound holds for the dilation spectrum. In this paper, we present several subquadratic algorithms that compute the value $\pi_G(\kappa)$ (either exactly or approximately) for the cases when G is a path, tree, or cycle.

In Section 2, we provide randomized algorithms for paths, trees, and cycles in $\mathbb{R}^2$, that allow $\pi_G(\kappa)$ to be computed in $O(n^{3/2+\epsilon})$ expected time. To this end, we first use a geometric transformation scheme introduced in [1].

This reduces the problem of computing $\pi_G(\kappa)$ to a counting problem involving points and cones in $\mathbb{R}^3$. By applying range counting techniques, we obtain the value of $\pi_G(\kappa)$.

In Section 3, faster algorithms will be presented for *approximating* the dilation spectrum in $\mathbb{R}^d$. More precisely, for any given reals $\kappa > 1$ and $\epsilon > 0$, we show how to compute an integer M that satisfies

$$\pi_G(\kappa) \leq M \leq \pi_G((1+\epsilon)\kappa).$$

Thus, this number M approximates the number of vertex-pairs having dilation at most κ. The running time of these deterministic algorithms is $O(n \log^2 n)$ for paths and cycles, and $O(n \log^3 n)$ for trees. Our approach is based on the well-separated pair decomposition [4].

We conclude the paper in Section 4 with some open problems.

2 Computing the Exact Dilation Spectrum

In this section, we present randomized algorithms that, for any given threshold $\kappa > 1$, compute the exact dilation spectrum $\pi_G(\kappa)$, for paths, cycles, and trees that are embedded in $\mathbb{R}^2$.

2.1 Paths in $\mathbb{R}^2$

In this subsection, we describe a randomized algorithm for computing $\pi_G(\kappa)$ for a polygonal path G in the plane. First, we describe a reduction from [1] that rephrases the problem of computing $\pi_G(\kappa)$ as a counting problem in three-dimensional space. Then we apply range counting algorithms to solve the latter problem. Throughout this subsection, G denotes a path, whose vertex set is a set of n points in $\mathbb{R}^2$, and $\kappa > 1$ denotes a real number.

2.1.1 Reduction to point-cone counting in $\mathbb{R}^3$

We start by considering the following problem, which is related to the problem of computing $\pi_G(\kappa)$.

Let p_0 be one of the end-vertices of the path G. Define the following order $<_G$ on the vertices of G: For two vertices p and q, we have $p <_G q$ if p is encountered before q when traversing G starting at p_0.

Let A and B be two vertex-sets of the path G such that $p <_G q$ holds for all p in A and all q in B. We will show how to compute

$$\pi_G(\kappa, A, B) := |\{(p,q) \in A \times B : \delta_G(p,q) \leq \kappa\}|,$$

i.e., the number of vertex-pairs (p,q) with $p \in A$ and $q \in B$ whose dilation does not exceed κ. Later, we will see how this result can be used to compute $\pi_G(\kappa)$.

For any vertex $p \in A$, we define the *weight* $\omega(p)$ of p to be the value

$$\omega(p) := \frac{d_G(p_0, p)}{\kappa}.$$

Let $\check{C}$ denote the cone

$$\check{C} : z = \sqrt{x^2 + y^2}$$

in $\mathbb{R}^3$. We map each vertex $p = (p_x, p_y)$ of A to the cone

$$C_p := \check{C} + (p_x, p_y, \omega(p)).$$

Thus, if we regard C_p as the graph of a bivariate function, then for any point $x \in \mathbb{R}^2$, we have

$$C_p(x) = |px| + \omega(p).$$

We define

$$\mathcal{C}(A) := \{C_p : p \in A\}.$$

We map each vertex $q = (q_x, q_y)$ of B to the point

$$\hat{q} := (q_x, q_y, \omega(q))$$

in $\mathbb{R}^3$, and define

$$\hat{B} := \{\hat{q} : q \in B\}.$$

The following lemma explains the relationship between the dilation of p and q (where $p \in A$ and $q \in B$) and the location of the point $\hat{q}$ with respect to the cone C_p:

Lemma 1 *Let p be a point of A and let q be a point of B. Then $\delta_G(p,q) \leq \kappa$ if and only if $\hat{q}$ lies below C_p, i.e., $\omega(q) \leq C_p(q)$.*

Proof. The proof follows by a straightforward algebraic manipulation:

$$\begin{aligned}
\delta_G(p,q) \le \kappa \quad &\Longleftrightarrow \quad \frac{d_G(p,q)}{|pq|} \le \kappa \\
&\Longleftrightarrow \quad \frac{d_G(p_0,q) - d_G(p_0,p)}{|pq|} \le \kappa \\
&\Longleftrightarrow \quad \frac{d_G(p_0,q)}{\kappa} \le |pq| + \frac{d_G(p_0,p)}{\kappa} \\
&\Longleftrightarrow \quad \omega(q) \le |pq| + \omega(p) \\
&\Longleftrightarrow \quad \omega(q) \le C_p(q).
\end{aligned}$$

∎

Thus, this lemma reduces the problem of counting all pairs $(p,q) \in A \times B$ with $\delta_G(p,q) \le \kappa$ to the problem of counting all pairs $(p,q) \in A \times B$ for which $\hat{q}$ lies below C_p.

2.1.2 Solving the point-cone counting problem: A preliminary algorithm

We have seen that the problem of computing $\pi_G(\kappa, A, B)$ amounts to counting the number of point-cone pairs $(\hat{q}, C_p) \in \hat{B} \times \mathcal{C}(A)$ for which $\hat{q}$ lies below C_p.

Suppose we are given a set P of n points in $\mathbb{R}^3$ and a set $\mathcal{C}$ of m cones in $\mathbb{R}^3$ whose axes are vertical and whose apices are their bottommost points. We describe a randomized algorithm that computes the value

$$\underline{\mu}(P,\mathcal{C}) := |\{(p,C) \in P \times \mathcal{C} : \ p \text{ lies below } C\}|.$$

In order to compute $\underline{\mu}(P,\mathcal{C})$, we fix a sufficiently large constant r, choose a random sample $\mathcal{R}$ of $r \log r$ cones in $\mathcal{C}$, and compute the vertical decomposition $\mathcal{A}_{\|}$ of the arrangement $\mathcal{A}$ of the cones in $\mathcal{R}$. By Theorem 8.21 in [18], $\mathcal{A}_{\|}$ has $O(r^3 \log^4 r)$ cells. For each cell Δ in $\mathcal{A}_{\|}$, let $P_\Delta = \{p \in P : p \in \Delta\}$, let $\mathcal{C}_{\not\Delta}$ be the set of cones in $\mathcal{C}$ that cross Δ, and let $\mathcal{C}_{\underline{\Delta}}$ be the set of cones in $\mathcal{C}$ that lie completely above Δ. Then,

$$\underline{\mu}(P,\mathcal{C}) = \sum_{\Delta \in \mathcal{A}_{\|}} (|P_\Delta||\mathcal{C}_{\underline{\Delta}}| + \underline{\mu}(P_\Delta, \mathcal{C}_{\not\Delta})).$$

Set $n_\Delta := |P_\Delta|$ and $m_\Delta := |\mathcal{C}_{\not\Delta}|$. Obviously, $\sum_\Delta n_\Delta = n$. It follows from the theory of random sampling (see [10]) that, with high probability,

$m_\Delta \leq m/r$ for all Δ. If this condition is not satisfied for the sample $\mathcal{R}$, then we pick a new random sample. The expected number of trials until we get a "good" sample is bounded from above by a constant.

If m_Δ or n_Δ is less than some prespecified constant, then we use a brute-force procedure to compute $\underline{\mu}(P_\Delta, \mathcal{C}_{\not\Delta})$ in $O(m_\Delta + n_\Delta)$ time. Otherwise, we compute $\underline{\mu}(P_\Delta, \mathcal{C}_{\not\Delta})$ recursively.

For $n, m > 0$, let $T(n,m)$ denote the expected running time of the algorithm on a set of n points and a set of m cones. We get the following probabilistic recurrence relation:

$$T(n,m) = \sum_{\Delta \in \mathcal{A}_{\|}} T\left(n_\Delta, \frac{m}{r}\right) + O(m+n). \tag{2}$$

We claim that the solution to this recurrence relation is, for any $\epsilon > 0$,

$$T(n,m) = O(m^{3+\epsilon} + n \log m). \tag{3}$$

In order to prove this claim, first recall that $\sum_\Delta n_\Delta = n$. It follows that the total number of points at any level of the recursion is n. Moreover, since the number of points in any recursive call contributes linearly to the cost of the divide step, and since the depth of the recursion is $O(\log m)$, the total contribution of the points to $T(n,m)$ is $O(n \log m)$.

It remains to analyze the contribution of the cones to the total cost. Since this is also linear in each divide step, it obeys the following recurrence relation:

$$t(m) = \sum_{\Delta \in \mathcal{A}_{\|}} t\left(\frac{m}{r}\right) + O(m).$$

Recall that the number of cells Δ in $\mathcal{A}_{\|}$ is $O(r^3 \log^4 r)$. It follows that

$$t(m) = O(r^3 \log^4 r) \cdot t\left(\frac{m}{r}\right) + O(m).$$

Consider the function t' defined by

$$t'(m) = r^3 \log^4 r \cdot t'\left(\frac{m}{r}\right) + O(m).$$

According to the Master Theorem [6], we have

$$t'(m) = O\left(m^{\log_r \left(r^3 \log^4 r\right)}\right) = O\left(m^{3 + 4 \log\log r / \log r}\right).$$

Therefore, by choosing the constant r sufficiently large, we have $t'(m) = O(m^{3+\epsilon})$. Since $t(m) = O(t'(m))$, the claim in (3) follows.

2.1.3 Solving the point-cone counting problem: A faster algorithm

Consider again a set P of n points in $\mathbb{R}^3$ and a set $\mathcal{C}$ of m cones in $\mathbb{R}^3$ whose axes are vertical and whose apices are their bottommost points. We saw above how the quantity $\underline{\mu}(P, \mathcal{C})$ can be computed in $O(m^{3+\epsilon} + n \log m)$ expected time. Observe that the number m of cones has a much larger effect on the running time than the number n of points. We will show below that the problem of computing $\underline{\mu}(P, \mathcal{C})$ is, in fact, symmetric in P and $\mathcal{C}$ (and, thus, in n and m as well). Because of this, we can interchange the roles of P and $\mathcal{C}$ and, as we will see, obtain a faster algorithm for computing $\underline{\mu}(P, \mathcal{C})$.

The key idea is to consider the following dualization step: Let $\hat{C}$ denote the cone

$$\hat{C} : z = -\sqrt{x^2 + y^2}$$

in $\mathbb{R}^3$. For any point $p = (p_x, p_y, p_z)$ in $\mathbb{R}^3$, define the cone

$$D(p) := \hat{C} + (p_x, p_y, p_z),$$

and for any cone $C = \check{C} + (p_x, p_y, p_z)$, define the point

$$D(C) := (p_x, p_y, p_z).$$

Observe that p lies below C if and only if $D(C)$ lies above $D(p)$. Thus, if we define

$$D(P) := \{D(p) : p \in P\}$$

and

$$D(\mathcal{C}) := \{D(C) : C \in \mathcal{C}\},$$

then we have

$$\underline{\mu}(P, \mathcal{C}) = \bar{\mu}(D(\mathcal{C}), D(P)),$$

where $\bar{\mu}(D(\mathcal{C}), D(P))$ is defined to be the number of pairs $(D(C), D(p))$ in $D(\mathcal{C}) \times D(P)$ for which $D(C)$ lies above $D(p)$.

We now show how to use this dualization to improve the running time obtained in (3). The improved algorithm does the following.

First assume that $m > n^3$. Then we use the duality transformation D to switch the roles of P and $\mathcal{C}$, and compute $\bar{\mu}(D(\mathcal{C}), D(P))$ (which is equal to the value $\underline{\mu}(P, \mathcal{C})$ that we want to compute) using the algorithm just described. Thus, by (3), the expected running time $T'(m,n)$ of the improved algorithm satisfies

$$T'(m,n) = O\left(n^{3+\epsilon} + m\log n\right) = O\left(m^{1+\epsilon}\right) \qquad \text{if } m > n^3.$$

Now assume that $m \le n^3$. As above, we fix a sufficiently large constant r, choose a random sample $\mathcal{R}$ of $r\log r$ cones in $\mathcal{C}$, and compute the vertical decomposition $\mathcal{A}_{\|}$ of the arrangement $\mathcal{A}$ of the cones in $\mathcal{R}$. As mentioned above, $\mathcal{A}_{\|}$ has $O(r^3\log^4 r)$ cells. For each cell $\Delta \in \mathcal{A}_{\|}$, we define P_Δ, $\mathcal{C}_{\not\Delta}$, and $\mathcal{C}_{\underline{\Delta}}$ as before. If P_Δ is too big, then we further subdivide it. To be more precise, if $|P_\Delta| > n/r^3$, we partition it into disjoint sets $P_\Delta^{(1)}, \ldots, P_\Delta^{(k_\Delta)}$, where $|P_\Delta^{(1)}| = \ldots = |P_\Delta^{(k_\Delta - 1)}| = n/r^3$ and $|P_\Delta^{(k_\Delta)}| \le n/r^3$. If $|P_\Delta| \le n/r^3$, then we define $k_\Delta = 1$ and $P_\Delta^{(1)} = P_\Delta$.

Observe that $\sum_{\Delta\in\mathcal{A}_{\|}} k_\Delta$, i.e., the total number of sets $P_\Delta^{(i)}$, is at most r^3 plus the number of cells in $\mathcal{A}_{\|}$. Therefore, $\sum_{\Delta\in\mathcal{A}_{\|}} k_\Delta$ is still $O(r^3\log^4 r)$. We have

$$\underline{\mu}(P,\mathcal{C}) = \sum_{\Delta\in\mathcal{A}_{\|}} \sum_{i=1}^{k_\Delta} \left(|P_\Delta^{(i)}||\mathcal{C}_{\underline{\Delta}}| + \underline{\mu}(P_\Delta^{(i)}, \mathcal{C}_{\not\Delta}) \right).$$

Set $n_\Delta^{(i)} = |P_\Delta^{(i)}|$ and $m_\Delta = |\mathcal{C}_{\not\Delta}|$. By construction, we have $n_\Delta^{(i)} \le n/r^3$. By the theory of random sampling (see [10]), we have, with high probability, $m_\Delta \le m/r$ for all Δ. If this condition is not satisfied for the sample $\mathcal{R}$, then we pick a new random sample. The expected number of trials until we get a "good" sample is bounded from above by a constant.

If m_Δ or $n_\Delta^{(i)}$ is less than a prespecified constant, then we use a brute-force procedure to compute $\underline{\mu}(P_\Delta^{(i)}, \mathcal{C}_{\not\Delta})$ in $O(m_\Delta + n_\Delta^{(i)})$ time. Otherwise, we compute $\underline{\mu}(P_\Delta^{(i)}, \mathcal{C}_{\not\Delta})$ recursively.

In this way, we obtain the following recurrence relation for the expected running time $T'(n,m)$ of the new algorithm:

$$T'(n,m) = \begin{cases} \sum_{\Delta\in\mathcal{A}_{\|}} \sum_{i=1}^{k_\Delta} T'\left(n_\Delta^{(i)}, m_\Delta\right) + O(m+n) & \text{if } m \le n^3, \\ O(m^{1+\epsilon}) & \text{if } m > n^3. \end{cases}$$

By the discussion above, this simplifies to

$$T'(n,m) = \begin{cases} O(r^3\log^4 r)\cdot T'\left(\frac{n}{r^3}, \frac{m}{r}\right) + O(m+n) & \text{if } m \le n^3, \\ O(m^{1+\epsilon}) & \text{if } m > n^3. \end{cases}$$

We claim that the solution to this recurrence relation satisfies

$$T'(n,m) = O\left((mn)^{3/4+\epsilon} + m^{1+\epsilon} + n^{1+\epsilon}\right), \tag{4}$$

for any constant $\epsilon > 0$.

The claim is clearly true in the case when $m > n^3$. Let us therefore focus on the case when $m \leq n^3$. In the k-th step of the recursion, the first variable n_k in T' is equal to $n_k = n/r^{3k}$, whereas the second variable m_k is equal to $m_k = m/r^k$. The recursion stops at the first k for which $m_k > n_k^3$; this is the case when $k = \lceil \log_r((n^3/m)^{1/8}) \rceil$. At that point, we have $m_k \leq (m^3/n)^{3/8}$, and the total number of subproblems created at depth k is $O((r^3 \log^4 r)^k) = O((r^{3+\epsilon})^k) = O((n^3/m)^{3/8+\epsilon})$. Since each of these subproblems can be solved in $O(m_k^{1+\epsilon}) = O((m^3/n)^{3/8+\epsilon})$ time, the overall time to solve all subproblems at the k-th level of the recursion amounts to $O((mn)^{3/4+\epsilon})$.

For each j with $1 \leq j \leq k$, the cost for the divide step at the j-th level of the recursion is $O(m_j + n_j) = O(n/r^{3j} + m/r^j)$ per subproblem and the number of subproblems at that level is $O(r^{(3+\epsilon)j})$. Thus, the total time for all divide steps at all levels of the recusion is equal to a quantity that is proportional to

$$\begin{aligned}
\sum_{j=1}^{k} r^{(3+\epsilon)j}\left(\frac{n}{r^{3j}} + \frac{m}{r^j}\right) &= \sum_{j=1}^{k}\left(r^{\epsilon j} n + r^{(2+\epsilon)j} m\right) \\
&= O\left(r^{\epsilon k} n + r^{(2+\epsilon)k} m\right) \\
&= O\left(\left(\frac{n^3}{m}\right)^{\epsilon/8} n + \left(\frac{n^3}{m}\right)^{(2+\epsilon)/8} m\right).
\end{aligned}$$

Since

$$\left(\frac{n^3}{m}\right)^{\epsilon/8} n \leq n^{1+3\epsilon/8} \leq n^{1+\epsilon},$$

and

$$\left(\frac{n^3}{m}\right)^{(2+\epsilon)/8} m = n^{3/4+3\epsilon/8} m^{3/4-\epsilon/8} \leq (mn)^{3/4+\epsilon},$$

the total time for all divide steps at all levels of the recusion is

$$O\left((mn)^{3/4+\epsilon}\right).$$

Therefore, we have shown that (4) holds for $m \leq n^3$.

2.1.4 Computing the dilation spectrum of a path

We are now ready to combine the results obtained above to compute the value $\pi_G(\kappa)$ for the case when G is a path in $\mathbb{R}^2$. The approach is to apply the divide-and-conquer technique and use the algorithm of Section 2.1.3 in the merge-step.

Theorem 1 *Let G be a path on n vertices in $\mathbb{R}^2$, and let $\kappa > 1$ be a real number. Then we can compute $\pi_G(\kappa)$, i.e., the number of vertex-pairs of G that attain dilation at most κ, in $O(n^{3/2+\epsilon})$ expected time, for any constant $\epsilon > 0$.*

Proof. Assume that n is larger than some prespecified constant. We assume for simplicity that n is a power of two. Let $p_0, p_1, \ldots, p_{n-1}$ be the vertices of G, when G is traversed from one end-vertex to the other end-vertex. Let $A = \{p_0, \ldots, p_{n/2-1}\}$ and $B = \{p_{n/2}, \ldots, p_{n-1}\}$, and let G_A and G_B be the paths with vertex sets A and B, respectively. Observe that

$$\pi_G(\kappa) = \pi_{G_A}(\kappa) + \pi_{G_B}(\kappa) + \pi_G(\kappa, A, B).$$

We have seen in Section 2.1.3 that $\pi_G(\kappa, A, B)$ can be computed in $O(n^{3/2+\epsilon})$ expected time. By recursively computing the values $\pi_{G_A}(\kappa)$ and $\pi_{G_B}(\kappa)$, we obtain the value $\pi_G(\kappa)$. The expected running time $T(n)$ of this algorithm satisfies the recurrence

$$T(n) = 2 \cdot T(n/2) + O(n^{3/2+\epsilon}).$$

By the Master Theorem [6], this solves to $T(n) = O(n^{3/2+\epsilon})$. ∎

We can use the same approach to actually report all pairs of vertices for which the dilation does not exceed κ, in additional time that is proportional to the size of the output.

2.2 Cycles in $\mathbb{R}^2$

In this subsection, we consider the case when G is a polygonal cycle on n points in $\mathbb{R}^2$. This case is more difficult than that of paths, because any two vertices p and q are connected by two paths, the shorter of which determines the dilation $\delta_G(p, q)$.

We denote the total length of all edges of G by $|G|$. For any two vertices p and q, we denote by $G[p,q]$ the portion of G from p to q in clockwise direction, and let $d'_G(p,q)$ denote its length. Then the shortest-path length $d_G(p,q)$ between p and q is given by

$$d_G(p,q) = \min(d'_G(p,q), |G| - d'_G(p,q)).$$

We can preprocess G in $O(n)$ time such that $d_G(p,q)$, for any two vertices p and q, can be computed in $O(1)$ time.

Our algorithm for computing $\pi_G(\kappa)$ uses the divide-and-conquer strategy. Therefore, we start by considering the problem that arises in the merge-step.

For any vertex p of G, let $\nu(p)$ denote the last vertex of G, in clockwise direction from p, for which $d'_G(p,\nu(p)) \leq |G|/2$.

Consider two vertices t_1 and t_2, and assume that the four vertices $t_1, t_2, b_1 = \nu(t_1), b_2 = \nu(t_2)$ appear in clockwise order along G. We will present a divide-and-conquer algorithm (which uses the algorithm of Section 2.1.3 in the merge-step) that computes

$$\begin{aligned} \pi_G(t_1,t_2,b_1,b_2) &:= \pi_G(\kappa, G[t_1,t_2], G[b_1,b_2]) \\ &= |\{(p,q) \in G[t_1,t_2] \times G[b_1,b_2]) : \delta_G(p,q) \leq \kappa\}|. \end{aligned}$$

First observe that

$$d'_G(b_1,b_2) \leq d'_G(t_2,b_2) \leq |G|/2.$$

Let m and n be the number of edges in $G[b_1,b_2]$ and $G[t_1,t_2]$, respectively. If $\min(m,n) = 1$, then we compute $\pi_G(t_1,t_2,b_1,b_2)$ in $O(m+n)$ time, by brute force. Otherwise, assume that $n \geq m > 1$. Let t be a median vertex of $G[t_1,t_2]$, and let $b = \nu(t)$. Then, $b \in G[b_1,b_2]$ and

$$\begin{aligned} &\pi_G(t_1,t_2,b_1,b_2) \\ &\quad = \pi_G(t_1,t,b,b_2) + \pi_G(t,t_2,b_1,b) + \pi_G(t_1,t,b_1,b) + \pi_G(t,t_2,b,b_2) - K, \end{aligned}$$

where

$$K = \begin{cases} 3 & \text{if } \delta_G(t,b) \leq \kappa, \\ 0 & \text{otherwise.} \end{cases}$$

The quantities $\pi_G(t_1,t,b_1,b)$ and $\pi_G(t,t_2,b,b_2)$ are computed recursively. Since the paths $G[t,t_2]$ and $G[b_1,b]$ are in $G[t,\nu(t)]$, we can compute $\pi_G(t,t_2,b_1,b)$, according to the results in Section 2.1.3, in $O((n+m)^{3/2+\epsilon})$ expected time. An analogous argument applies to $\pi_G(t_1,t,b,b_2)$.

Let m_1 be the number of edges in $G[b_1, b]$. Then $G[b, b_2]$ contains $m - m_1$ edges. Let $T(n, m)$ denote the expected time for computing $\pi_G(t_1, t_2, b_1, b_2)$. Then we obtain the following recurrence, for any constant $\epsilon > 0$:

$$T(n,m) \leq T(n/2, m_1) + T(n/2, m - m_1) + O\left((n+m)^{3/2+\epsilon}\right), \quad \text{if } n \geq m > 1,$$

with a symmetric inequality for $m > n$, and $T(n, 1) = O(n)$, $T(1, m) = O(m)$. The solution to this recurrence is

$$T(n,m) = O\left((n+m)^{3/2+\epsilon}\right).$$

Now consider the problem of computing $\pi_G(\kappa)$ for the cycle G. We choose a vertex v and let $G_1 := G[v, \nu(v)]$. If $d'_G(v, \nu(v)) < |G|/2$, then we let v' be the vertex clockwise next to $\nu(v)$. If $d'_G(v, \nu(v)) = |G|/2$, then we let $v' := \nu(v)$. In either case, let $G_2 := G[v', v]$. Observe that

$$\pi_G(\kappa) = \pi_{G_1}(\kappa) + \pi_{G_2}(\kappa) + \pi_G(v, \nu(v), v', v).$$

The values $\pi_{G_1}(\kappa)$ and $\pi_{G_2}(\kappa)$ can be computed in $O(n^{3/2+\epsilon})$ expected time using Theorem 1, because both $d'_G(v, \nu(v))$ and $d'_G(v', v)$ are less than or equal to $|G|/2$. The value of $\pi_G(v, \nu(v), v', v)$ can be computed within the same time bound by the recursive algorithm just described. We thus obtain the following result.

Theorem 2 *Let G be a cycle on n vertices in $\mathbb{R}^2$, and let $\kappa > 1$ be a real number. Then we can compute $\pi_G(\kappa)$, i.e., the number of vertex-pairs of G that attain dilation at most κ, in $O(n^{3/2+\epsilon})$ expected time, for any constant $\epsilon > 0$.*

2.3 Trees in $\mathbb{R}^2$

Let G be a tree whose vertex set is a set of n points in $\mathbb{R}^2$. It is well known that G contains a vertex v whose removal leaves two graphs G'_1 and G'_2, having at most $2n/3$ vertices each. Moreover, such a *centroid vertex* v can be computed in $O(n)$ time. Each of the two graphs G'_1 and G'_2 is a forest of trees, while each of the graphs $G_1 := G'_1 \cup \{v\}$ and $G_2 := G'_2 \cup \{v\}$ is connected and, hence, a tree again.

We will, again, apply the divide-and-conquer technique to compute the value $\pi_G(\kappa)$. Consider two distinct vertices p and q in the tree G. If both p

and q belong to the same tree G_i, then the path connecting p and q in G is also contained in G_i; thus, the pair (p, q) will be inspected if we recursively process G_1 and G_2. Otherwise, let us assume that p is a vertex of G_1 and q is a vertex of G_2. The path connecting p and q in G leads through the centroid vertex v. Thus, using the notations

$$\omega_1(p) := \frac{d_{G_1}(p, v)}{\kappa} \text{ and } \omega_2(q) := \frac{d_{G_2}(v, q)}{\kappa},$$

we obtain

$$\begin{aligned}
\delta_G(p, q) \leq \kappa \quad & \Longleftrightarrow \quad \frac{d_G(p, q)}{|pq|} \leq \kappa \\
& \Longleftrightarrow \quad \frac{d_{G_1}(p, v) + d_{G_2}(v, q)}{|pq|} \leq \kappa \\
& \Longleftrightarrow \quad \frac{d_{G_1}(p, v)}{\kappa} \leq |pq| - \frac{d_{G_2}(v, q)}{\kappa} \\
& \Longleftrightarrow \quad \omega_1(p) \leq |pq| - \omega_2(q) \\
& \Longleftrightarrow \quad \omega_1(p) \leq C_q^2(p),
\end{aligned}$$

where C_q^2 denotes the upwards oriented cone $\check{C}$ in $\mathbb{R}^3$, translated such that its apex is at a distance of $\omega_2(q)$ *below* the point $(q, 0)$. As in Section 2.1.3, we can count the vertex-pairs (p, q) for which p is a vertex in G_1, q is a vertex in G_2, and $\delta_G(p, q) \leq \kappa$, in $O(n^{3/2+\epsilon})$ expected time.

Consequently, we obtain a divide-and-conquer algorithm for computing $\pi_G(\kappa)$. The expected running time $T(n)$ of this algorithm satisfies the recurrence

$$T(n) = T(n - k + 1) + T(k) + O(n^{3/2+\epsilon}),$$

where $n/3 \leq k \leq 2n/3$. This recurrence solves to $O(n^{3/2+\epsilon})$. Thus, we obtain the following result.

Theorem 3 *Let G be a tree on n vertices in $\mathbb{R}^2$, and let $\kappa > 1$ be a real number. Then we can compute $\pi_G(\kappa)$, i.e., the number of vertex-pairs of G that attain dilation at most κ, in $O(n^{3/2+\epsilon})$ expected time, for any constant $\epsilon > 0$.*

3 Computing the Approximate Dilation Spectrum

Now we set out to give faster algorithms for computing an approximation of the dilation spectrum. The results of this section hold true for paths, cycles, and trees on point sets in in $\mathbb{R}^d$. Our reduction uses the well-separated pair decomposition, thus adding to the list of applications of this powerful method introduced in Callahan's Ph.D. thesis [3].

3.1 Well-separated pairs

We briefly review this decomposition and some of the relevant properties that we are going to use. We assume that the vertices of the input graph are in $\mathbb{R}^d$, where $d \geq 2$ is a constant. Let $s > 0$ denote a real number, called the *separation constant*. Two point sets A and B in $\mathbb{R}^d$ are said to be *well-separated* with respect to s, if they can be circumscribed by two disjoint balls of the same radius, say ρ, which are at least $s \cdot \rho$ apart.

For two well-separated sets A and B, the following two claims are easy to verify:

1. If a, a', and a'' are points in A and b is a point in B, then $|a'a''| \leq \frac{2}{s}|ab|$.
2. If a and a' are points in A and b and b' are points in B, then $|ab| \leq (1 + 4/s)|a'b'|$.

Given a set S of n points in $\mathbb{R}^d$, a *well-separated pair decomposition (WSPD)* consists of a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_k, B_k\}$ of well-separated pairs of subsets of S, such that, for any two distinct points p and q in S, there is a unique index i such that $p \in A_i$ and $q \in B_i$ holds, or vice versa.

Such a WSPD can be constructed from the *split-tree* $T(S)$ of the point set S. This tree is defined as follows. The root of $T(S)$ is associated with the bounding box of S, denoted by $\mathrm{BB}(S)$. The subtrees are created recursively by halving the longest edge of $\mathrm{BB}(S)$, creating two subsets S_1 and S_2 of S, whose split-trees $T(S_1)$ and $T(S_2)$ become the two subtrees of the root of $T(S)$. Note that there is no bound on the ratio of the sizes of S_1 and S_2, implying that the tree $T(S)$ may have height $\Omega(n)$.

Given a separation constant $s > 0$, the pairs $\{A_i, B_i\}$ of a WSPD of S can be obtained from $T(S)$ by recursively inspecting the offsprings of each internal node. As was shown by Callahan and Kosaraju [3, 4], this process produces a set of well-separated pairs from each internal node of $T(S)$. Each subset A_i (and, similarly, each subset B_i) of a WSPD pair corresponds to a node v of the split-tree, in the sense that A_i equals the set of all points stored at the leaves of the subtree that is rooted at v.

Clearly, for a given point set S, if the separation constant s is sufficiently large, a WSPD of S must consist of all $\binom{n}{2}$ singleton pairs. The surprising fact shown by Callahan and Kosaraju [3, 4] is the following. If the dimension d and the separation constant s are constants, then the number k of pairs in a WSPD depends only linearly on the size n of S, and the WSPD can be constructed in $O(n \log n)$ time.

A modified version of the above result (see Chapter 4.5 of Callahan's thesis [3]) will be used in the following subsections. Callahan showed how to compute a WSPD $\{A_i, B_i\}$, $1 \leq i \leq k$, such that at least one of A_i and B_i is a singleton set, and the number k of pairs is $O(n \log n)$. Again, each subset A_i and B_i corresponds to the set of points stored in the subtree rooted at some node of the split-tree. Also, as for the standard construction, this WSPD can be constructed in $O(n \log n)$ time.

3.2 A general algorithm

We start by describing a general algorithm for approximating the dilation spectrum of an arbitrary Euclidean graph. In the subsequent subsections, we then show how to efficiently implement this algorithm for paths, cycles, and trees.

Given a Euclidean graph G, a real number $\kappa > 1$, and a real constant $\epsilon > 0$, our general algorithm will output an integer M for which

$$\pi_G(\kappa) \leq M \leq \pi_G((1+\epsilon)).$$

The idea is as follows. Consider a WSPD

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_k, B_k\}$$

for the vertex set of G, where $k = O(n \log n)$ and each set A_i is a singleton set $A_i = \{a_i\}$. We know that the Euclidean distances between a_i and all points b_i in B_i are approximately equal. Therefore, by only considering the shortest-path distances $d_G(a_i, b_i)$, we obtain approximations to the dilations $\delta_G(a_i, b_i)$ for all points b_i in B_i. This observation leads to the general algorithm $\mathcal{A}$ presented below.

General Algorithm $\mathcal{A}$:

Input: A geometric graph G on a set S of n points in $\mathbb{R}^d$, a real number $\kappa > 1$, and a real constant $\epsilon > 0$.
Output: A number M satisfying $\pi_G(\kappa) \leq M \leq \pi_G((1+\epsilon)\kappa)$.

Step 1: Let $\epsilon' = \sqrt{1+\epsilon} - 1$. Using separation constant $s = 4/\epsilon'$, compute a WSPD

$$\{A_1, B_1\}, \{A_2, B_2\}, \dots, \{A_k, B_k\}$$

for the set S, with the added conditions that $|A_i| = 1$ for each $i = 1, \dots, k$, and $k = O(n \log n)$.

Step 2: For each $i = 1, \dots, k$, let $A_i = \{a_i\}$, and let D_i denote the Euclidean distance $|a_i b|$, where b is an arbitrary element of B_i. Compute

$$m_i = |\{b_i \in B_i : d_G(a_i, b_i) \leq (1+\epsilon')\kappa D_i\}|.$$

Step 3: Return $M = \sum_{i=1}^{k} m_i$.

Observe that algorithm $\mathcal{A}$ can be easily modified to actually output the M pairs of points counted. The following lemma proves the correctness of the algorithm.

Lemma 2 *Consider the output M of algorithm $\mathcal{A}$. Then*

$$\pi_G(\kappa) \leq M \leq \pi_G((1+\epsilon)\kappa).$$

Proof. Consider an index i with $1 \leq i \leq k$. In Step 2 of the algorithm, D_i is equal to $|a_i b|$, where b is an arbitrary element of B_i.

Let b_i be a point in B_i such that $\delta_G(a_i, b_i) \leq \kappa$. Then $d_G(a_i, b_i) \leq \kappa |a_i b_i|$ and, by the properties of well-separated pairs (see Section 3.1),

$$d_G(a_i, b_i) \leq \kappa(1 + 4/s) \cdot |a_i b| = (1+\epsilon')\kappa D_i.$$

Consequently, the pair (a_i, b_i) is counted in the variable m_i. This proves that $\pi_G(\kappa) \leq M$.

To prove the second inequality, let b_i be a point in B_i such that the pair (a_i, b_i) is counted in Step 2. Then, $d_G(a_i, b_i) \leq (1+\epsilon')\kappa D_i$ holds. But then, by the same property of well-separated pairs,

$$d_G(a_i, b_i) \leq (1+\epsilon')^2 \kappa |a_i b_i| = (1+\epsilon)\kappa |a_i b_i|,$$

implying that this pair has a dilation $\delta_G(a_i, b_i)$ of at most $(1+\epsilon)\kappa$. It follows that $M \leq \pi_G((1+\epsilon)\kappa)$, because each pair (a_i, b_i) is counted at most once. ■

In the next subsections, we show how the general algorithm $\mathcal{A}$ introduced above can be implemented to run efficiently on paths, cycles, and trees.

3.3 Paths

Let the graph G be a path $(p_0, p_1, \ldots, p_{n-1})$ on the points of the set S, let $\kappa > 1$ be a real number, and let $\epsilon > 0$ be a real constant. Following our general algorithm $\mathcal{A}$ of Section 3.2, we first compute a split-tree $T = T(S)$ and a corresponding WSPD

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_k, B_k\}$$

for S, with separation constant $s = 4/\epsilon'$, where $A_i = \{a_i\}$, for all $1 \leq i \leq k$, and $k = O(n \log n)$. As mentioned in Section 3.1, this can be done in $O(n \log n)$ time.

Before we continue with the general algorithm $\mathcal{A}$, we perform two preprocessing steps on the paht G.

In the first preprocessing step, we traverse the path G, starting from vertex p_0, and compute for each vertex p_j its path-distance $d_G(p_0, p_j)$. Using this information, we can compute, for any two indices j and k with $0 \leq j < k < n$, the path-distance $d_G(p_j, p_k)$ in $O(1)$ time, as the difference between $d_G(p_0, p_k)$ and $d_G(p_0, p_j)$.

In the second preprocessing step, we traverse the split-tree T in postorder and store, with each node u of T, the list S_u of all points contained in the leaves of the subtree of u, sorted by their order along the path G. This involves merging two sorted sublists at each internal node u of the tree. If v and w are the two children of u, and if their subtrees store m and m' points, where $m \leq m'$, then S_u can be obtained from S_v and S_w in $O(m \log m')$ time. In fact, we store these lists as balanced binary search trees, because we need to perform binary search on them. The merge is performed by repeatedly inserting the elements of the shorter list into the longer one. Thus, all lists S_u can be computed in $O(n \log^2 n)$ total time.

We now consider the implementation of Step 2 of our general algorithm $\mathcal{A}$. Let i be an integer with $1 \leq i \leq k$, and consider the pair $(\{a_i\}, B_i)$ of our WSPD. Let v_i be the node of T such that B_i is equal to the set of points stored in the subtree of v_i. Thus, S_{v_i} is the list containing the points of B_i, sorted by their order along the path G. Following algorithm $\mathcal{A}$, we choose an arbitrary element b in B_i and set $D_i = |a_i b|$. We use binary search with a_i to divide the list S_{v_i} into two sublists $S^1_{v_i}$ and $S^2_{v_i}$: The sublist $S^1_{v_i}$ consists of all elements in S_{v_i} that come before a_i in the path G, whereas $S^1_{v_i}$ consists of all elements in S_{v_i} that come after a_i in G. Our goal is to compute the number of elements b_i in B for which the path-distance $d_G(a_i, b_i)$ is at most $(1+\epsilon')\kappa D_i$. Each such element b_i is either in $S^1_{v_i}$ or in $S^2_{v_i}$. Since both sublists are sorted by path-distance from a_i, we can use two binary searches, one on each of the two sublists, in order to identify the number of points b_i in S_{v_i} for which $d_G(a_i, b_i) \leq (1+\epsilon')\kappa D_i$ holds.

It is easy to see that this correctly implements Step 2. Since it involves $3k = O(n \log n)$ binary searches altogether, the running time of the entire algorithm is $O(n \log^2 n)$. Thus, we obtain the following result.

Theorem 4 *Let G be a path on n vertices in $\mathbb{R}^d$, let $\kappa > 1$ be a real number, and let $\epsilon > 0$ be a real constant. In $O(n \log^2 n)$ time, we can compute an integer M that lies between $\pi_G(\kappa)$ and $\pi_G((1+\epsilon)\kappa)$.*

3.4 Cycles

If the graph G is a cycle $(p_0, p_1, \ldots, p_{n-1}, p_0)$ on the points of the set S, then we can apply quite the same approach as in Subsection 3.3. The only difference is in the list S_u associated with each node u of the split-tree. In fact, these lists are now cyclic, making it necessary to adapt the binary search routines. This does not affect the running time, and we obtain, in analogy to Theorem 4, the following result.

Theorem 5 *Let G be a cycle on n vertices in $\mathbb{R}^d$, let $\kappa > 1$ be a real number, and let $\epsilon > 0$ be a real constant. In $O(n \log^2 n)$ time, we can compute an integer M that lies between $\pi_G(\kappa)$ and $\pi_G((1+\epsilon)\kappa)$.*

3.5 Trees

We now turn to the case of trees. Let S be a set of n points in $\mathbb{R}^d$ that are the vertices of a Euclidean tree G, let $\kappa > 1$ be a real number, and let $\epsilon > 0$ be a real constant. Our algorithm for approximating $\pi_G(\kappa)$ looks as follows.

Step 1: As in Section 2.3, we determine, in time $O(n)$, a centroid vertex v of G whose removal splits G into forests G_1' and G_2', having at most $2n/3$ vertices each. The graphs $G_1 := G_1' \cup \{v\}$ and $G_2 := G_2' \cup \{v\}$ are connected and, hence, trees again. Traverse each of these two trees in preorder (starting at the root v), and store with each vertex p the tree-distance $d_G(p, v)$ between p and the centroid vertex v.

Step 2: Run the same algorithm recursively, once on the tree G_1, and once on the tree G_2.

Step 3: In this step, the algorithm considers the pairs (p, q), where p is a vertex of G_1 and q is a vertex of G_2. Compute a split-tree T, and a corresponding WSPD

$$\{\{a_1\}, B_1\}, \{\{a_2\}, B_2\}, \ldots, \{\{a_k\}, B_k\},$$

where $k = O(n \log n)$, for separation constant $s := 4/\epsilon'$, where $\epsilon' = \sqrt{1+\epsilon} - 1$.

Step 4: For each node u of the split-tree, denote by S_u the set of points of S that are stored in the subtree of u. Traverse T in postorder, and compute, for each of its nodes u, the sorted sequence S_u^1 of nodes, consisting of all nodes in G_1 sorted according to their three-distance from the centroid v. Similarly, compute the sorted sequence S_u^2 of nodes, consisting of all nodes in G_2 sorted according to their tree-distance from the centroid v.

Step 5: For each i with $1 \leq i \leq k$, we do the following. Consider the pair $\{\{a_i\}, B_i\}$ in our WSPD and the node u_i in the split tree such that $B_i = S_{u_i}$. Let the two sets computed in Step 4 for u_i be $S_{u_i}^1$ and $S_{u_i}^2$. If $a_i \in G_1$, then use binary search to identify the number of points $b_i \in S_{u_i}^2$ such that $d_G(a_i, b_i) \leq (1 + \epsilon')\kappa D_i$. Otherwise the search is performed in $S_{u_i}^2$.

The correctness of the above algorithm is proved by induction and results from the fact that Step 2 counts all relevant pairs of nodes that involve a_i and another node in G_1 (assuming that a_i is a node in G_1), whereas Step 5 counts all relevant pairs of nodes that involve a_i and another node in G_2.

Note that the distance in G between a point a_i in G_1 and a point b_i in G_2 is obtained by adding their distances to the centroid vertex v.

To prove the time complexity, let $\mathcal{T}(n)$ denote the running time of the entire algorithm on an input tree having n vertices. Then,

$$\mathcal{T}(n) = O(n \log^2 n) + \mathcal{T}(n') + \mathcal{T}(n''),$$

where n' and n'' are positive integers such that $n' \leq 2n/3$, $n'' \leq 2n/3$, and $n' + n'' = n + 1$. The $O(n \log^2 n)$ term comes about because the binary search spends $O(\log n)$ time on each of the $O(n \log n)$ well-separated pairs. The above recurrence relation solves to $\mathcal{T}(n) = O(n \log^3 n)$.

Theorem 6 *Let G be a tree on n vertices in $\mathbb{R}^d$, let $\kappa > 1$ be a real number, and let $\epsilon > 0$ be a real constant. In $O(n \log^3 n)$ time, we can compute an integer M that lies between $\pi_G(\kappa)$ and $\pi_G((1+\epsilon)\kappa)$.*

4 Open Problems

Besides the obvious question about improving the running times of our algorithms, an interesting question is how to efficiently compute the dilation spectrum for planar and general Euclidean graphs.

Another interesting question is to what extent the dilation spectrum of a Euclidean graph can be used for reconstructing the original graph. The structure of a graph is certainly not uniquely characterized by its spectrum (not even for the simple case of two adjacent edges). However, the dilation spectrum could highlight important structural characteristics of the graph, especially in combination with other parameters and network quality measures.

Acknowledgement: The authors would like to thank the organizers and all participants of the Korean workshop 2004 at Schloß Dagstuhl for establishing the nice and stimulating atmosphere that gave rise to this work. The authors also thank the two anonymous referees for their comments.

References

[1] P. K. Agarwal, R. Klein, C. Knauer, S. Langerman, P. Morin, M. Sharir, and M. Soss. Computing the detour and spanning ratio of paths, trees,

and cycles in 2D and 3D. *Discrete & Computational Geometry*, 39:17–37, 2008.

[2] S. Cabello and C. Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry: Theory and Applications*, 2009. Article in press.

[3] P. B. Callahan. *Dealing With Higher Dimensions: The Well-Separated Pair Decomposition and Its Applications.* Ph.D. thesis, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, 1995.

[4] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42:67–90, 1995.

[5] O. Cheong, H. Haverkort, and M. Lee. Computing a minimum-dilation spanning tree is NP-hard. *Computational Geometry: Theory and Applications*, 41:188–205, 2008.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 2nd edition, 2001.

[7] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science, Amsterdam, 2000.

[8] M. Farshi, P. Giannopoulos, and J. Gudmundsson. Improving the stretch factor of a geometric network by edge augmentation. *SIAM Journal on Computing*, 38:226–240, 2008.

[9] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.

[10] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

[11] R. Klein, C. Knauer, G. Narasimhan, and M. Smid. Exact and approximation algorithms for computing the dilation spectrum of paths, trees, and cycles. In *Proceedings of the 16th International Symposium on Algorithms and Computation*, volume 3827 of *Lecture Notes in Computer Science*, pages 849–858, Berlin, 2005. Springer-Verlag.

[12] R. Klein and M. Kutz. Computing geometric minimum-dilation graphs is NP-hard. In *Proceedings of the 14th International Symposium on Graph Drawing (GD 2006)*, volume 4372 of *Lecture Notes in Computer Science*, pages 196–207, Berlin, 2007. Springer-Verlag.

[13] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[14] J. Luo and C. Wulff-Nilsen. Computing best and worst shortcuts of graphs embedded in metric spaces. In *Proceedings of the 19th International Symposium on Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 764–775, Berlin, 2008. Springer-Verlag.

[15] G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM Journal on Computing*, 30:978–989, 2000.

[16] G. Narasimhan and M. Smid. *Geometric Spanner Networks.* Cambridge University Press, Cambridge, UK, 2007.

[17] D. Peleg and A. A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.

[18] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, Cambridge, UK, 1995.

[19] C. Wulff-Nilsen. Computing the maximum detour of a plane graph in subquadratic time. In *Proceedings of the 19th International Symposium on Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 740–751, Berlin, 2008. Springer-Verlag.