Exact and Approximation Algorithms for Computing the Dilation Spectrum of Paths, Trees, and Cycles

Rolf Klein[1], Christian Knauer[2], Giri Narasimhan[3], and Michiel Smid[4]

[1] Institute of Computer Science I, Universität Bonn, Römerstraße 164, D-53117 Bonn, Germany, rolf.klein@uni-bonn.de; supported by DFG Kl 655/14.
[2] Freie Universität Berlin, Institute of Computer Science, Berlin, Germany, christian.knauer@inf.fu-berlin.de.
[3] School of Computer Science, Florida, International University, ECS389, University Park, Miami, FL 33199, U.S.A., giri@cs.fiu.edu.
[4] School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario K1S 5B6, Canada, michiel@scs.carleton.ca; supported by NSERC.

Abstract. Let G be a graph embedded in Euclidean space. For any two vertices of G their *dilation* denotes the length of a shortest connecting path in G, divided by their Euclidean distance. In this paper we study the spectrum of the dilation, over all pairs of vertices of G. For paths, trees, and cycles in 2D we present $O(n^{3/2+\epsilon})$ randomized algorithms that compute, for a given value $\kappa \geq 1$, the exact number of vertex pairs of dilation $> \kappa$. Then we present deterministic algorithms that approximate the number of vertex pairs of dilation $> \kappa$ up to an $1 + \eta$ factor. They run in time $O(n \log^2 n)$ for chains and cycles, and in time $O(n \log^3 n)$ for trees, in any constant dimension.

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

1 Introduction

Let S be a set of n points in $\mathbb{R}^d$, where $d \geq 1$ is a small 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 as the Euclidean distance $|pq|$ between the two vertices p and q. Such graphs are called *Euclidean graphs*. Let $d_G(x, y)$ denote the length of a shortest path in G that connects x and y. Then the *dilation* between x and y in G is defined as

$$\delta_G(x, y) = \frac{d_G(x, y)}{|xy|}.$$

Euclidean graphs are frequently used for modeling traffic or transportation networks. In order to measure their performance, the *dilation* of G [13] has been

used, which is defined as the maximum dilation over all pairs of vertices in G, i.e.,

$$\sigma(G) = \max_{p \neq q \in V} \delta_G(p, q). \tag{1}$$

This value is also called the stretch factor, the spanning ratio, or the distortion [10] of G. A lot of work has been done on the construction of good spanners, i. e., of sparse graphs of low dilation that connect a given vertex set and enjoy other desirable properties; see the handbook chapter [5] or the forthcoming monograph [12]. How to compute the dilation of a given Euclidean graph has first been addressed in [11]. They gave an $O(n \log n)$ algorithm for approximating, up to an $1 - \epsilon$ factor, the dilation of paths, trees, and cycles in Euclidean space of constant dimension. In [1] exact randomized algorithms were given that run in time $O(n \log n)$ for paths in the plane, and in time $O(n \log^2 n)$ for computing the dilation of a plane tree or cycle. In 3D, time $O(n^{4/3+\epsilon})$ is sufficient for either type of graph; see also [2, 9]. For general graphs, nothing better seems to be known than running Dijkstra's algorithm for each vertex of G, which leads to an $O(mn + n^2 \log n)$ algorithm [4]; here n denotes the number of vertices, while m is the number of edges of G. In the recent paper [6], the interesting question has been addressed how to decrease the dilation of a graph as much as possible by inserting another edge.

In this paper we are studying the vertex-to-vertex dilation of graphs from a different perspective. Computing the dilation, as defined in (1), simply points to the pair of vertices for which the dilation is maximized. In real networks, one may wish to tolerate some number of pairs of vertices with high dilations, if the network provides good connections to the majority of vertex pairs. Therefore, we are interested in computing (exactly or approximately) the *dilation spectrum* of a graph G. That is, for a given threshold $\kappa > 1$ we are interested in the number

$$\pi_G(\kappa) := |\{\ (p, q) \in\ S^2;\ \delta_G(p, q) > \kappa\ \}| \tag{2}$$

of all vertex pairs whose dilation exceeds κ. This distribution of the dilation could also be helpful in understanding structural properties of a given geometric graph.

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 [7] that the distances in G between all pairs of vertices of a plane geometric graph can be computed in $O(n^2)$ total time. The same upper bound holds for the dilation spectrum.

This paper is organized as follows. In Section 2 we provide randomized algorithms for polygonal paths, trees, and cycles in the plane, that allow $\pi_G(\kappa)$ to be computed in time $O(n^{3/2+\epsilon})$. To this end, we first use a geometric transformation scheme introduced in [1], in order to reduce the problem of computing $\pi_G(\kappa)$ to a counting problem, and then apply suitable range counting techniques. Then, in Section 3, faster algorithms will be presented for *approximating* the dilation spectrum. More precisely, for given reals $\kappa \geq 1$ and $\epsilon > 0$ we shall compute a

number M satisfying

$$\pi_G(\kappa\ (1+\epsilon)^2) \leq M \leq \pi_G(\kappa) \tag{3}$$

that approximates the number of vertex pairs of dilation $> \kappa$. The run time of these deterministic algorithms is in $O(n \log^2 n)$ for paths and cycles, and in $O(n \log^3 n)$ for trees. Our approach is based on the well-separated pair decomposition [3]; hence, it works in any constant dimension. Finally, in Section 4 we mention some open problems.

2 Computing the Exact Dilation Spectrum

In this section we derive exact algorithms for computing the dilation distribution $\pi_G(\kappa)$ of certain types of plane graphs, given a threshold $\kappa > 1$.

2.1 Paths

We start with a randomized algorithm for computing $\pi_P(\kappa)$ for a simple polygonal path P in the plane. First, we describe a reduction that rephrases the problem of computing $\pi_P(\kappa)$ as a counting problem in three-dimensional space. Then we apply range counting algorithms to solve that problem.

Reduction to range counting We shall consider a slightly more generally version of our problem. Namely, let A, B be two disjoint vertex sets of G; then we are going to compute

$$\pi_G(\kappa, A, B) = |\{\ (x,y) \in A \times B \mid \delta_G(x,y) > \kappa\ \}|,$$

the number of vertex pairs in $A \times B$ whose dilation exceeds κ.

Assume that some orientation $<_P$ of P is given, and let p_0 be the first vertex of P. For a vertex $p \in A$, we define the *weight* of p to be $\omega(p) = d_P(p_0, p)/\kappa$. Let $\check{C}$ denote the cone $z = \sqrt{x^2 + y^2}$ in $\mathbb{R}^3$. We map each vertex $p = (p_x, p_y) \in A$ to the cone $C_p = \check{C} + (p_x, p_y, \omega(p))$. If we regard C_p as the graph of a bivariate function then for any point $x \in \mathbb{R}^2$, $C_p(x) = |xp| + \omega(p)$. Set $\mathcal{C}(A) = \{C_p \mid p \in A\}$. We map a vertex $q = (q_x, q_y) \in B$ to the point $\hat{q} = (q_x, q_y, \omega(q)) \in \mathbb{R}^3$. Let $\hat{B} = \{\hat{q} \mid q \in B\}$.

Lemma 1. *For any vertex pair $(p,q) \in A \times B$ such that $p <_P q$, we have that $\delta(q,p) \leq \kappa$ if and only if $\hat{q}$ lies below C_p, i.e., $\omega(q) \leq C_p(q)$.*

Proof.

$$\begin{aligned}
\delta(p,q) \leq \kappa &\iff \frac{d_P(p,q)}{|qp|} \leq \kappa \iff \frac{d_P(p_0,q) - d_P(p_0,p)}{|qp|} \leq \kappa \\
&\iff \frac{d_P(p_0,q)}{\kappa} \leq |qp| + \frac{d_P(p_0,p)}{\kappa} \\
&\iff \omega(q) \leq |qp| + \omega(p) \iff \omega(q) \leq C_p(q).
\end{aligned}$$

If there is a vertex pair $(p,q) \in A \times B$ with $\delta(p,q) > \kappa$ we cannot be sure that p that lies before q on P. Hence, we must also consider the symmetric situation with the orientation of P reversed.

Counting points above cones In the previous section we have seen that the problem of computing $\pi(\kappa, A, B)$ amounts to count the number of point-cone pairs $(p, C) \in \hat{B} \times \mathcal{C}(A)$ such that p lies above C.

Suppose we are given a set P of n points 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 to compute $\underline{\mu}(P, \mathcal{C})$, the number of point-cone pairs $(p, C) \in P \times \mathcal{C}$ such that p lies above C.

We fix a sufficiently large constant r, choose a random sample $\mathcal{R}$ of $O(r)$ cones in $\mathcal{C}$, and compute the vertical decomposition $\mathcal{A}_\parallel$ of the arrangement $\mathcal{A}$ of the cones in $\mathcal{R}$. As is known (c.f. [14], Theorem 8.21), $\mathcal{A}_\parallel$ has $O(r^3 \log^4 r)$ cells. For each cell $\Delta \in \mathcal{A}_\parallel$, let $P_\Delta = \{p \in P \mid p \in \Delta\}$, let $\mathcal{C}_{\not\Delta} \subseteq \mathcal{C}$ be the set of cones crossing Δ, and let $\mathcal{C}_{\underline{\Delta}} \subseteq \mathcal{C}$ be the set of cones which lie completely below Δ. Clearly $\underline{\mu}(P, \mathcal{C}) = \sum_{\Delta \in \mathcal{A}_\parallel} |P_\Delta||\mathcal{C}_{\underline{\Delta}}| + \underline{\mu}(P_\Delta, \mathcal{C}_{\not\Delta})$.

Set $n_\Delta = |P_\Delta|$ and $m_\Delta = |\mathcal{C}_{\not\Delta}|$. Obviously, $\sum_\Delta n_\Delta = n$ and by the theory of random sampling [8], $m_\Delta \leq m/r$ with high probability, for all Δ. If this condition is not satisfied for the sample $\mathcal{R}$, we pick a new random sample. The expected number of trials until we get a 'good' sample is constant. If m_Δ or n_Δ is less than a prespecified constant, then we use a naive procedure to determine $\underline{\mu}(P_\Delta, \mathcal{C}_\Delta)$. Otherwise, we recursively compute $\underline{\mu}(P_\Delta, \mathcal{C}_\Delta)$. For $m, n > 0$, 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:

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

where $\sum_\Delta n_\Delta \leq n$. The solution to the above recurrence is, for any $\epsilon > 0$,

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

To improve the running time of the algorithm we can perform the following dualization step: Let $\hat{C}$ denote the cone $z = -\sqrt{x^2 + y^2}$ in $\mathbb{R}^3$. If we map each point $p = (p_x, p_y, p_z) \in P$ to the cone $\delta(p) = C_p = \hat{C} + (p_x, p_y, p_z)$ and each cone $C = \hat{C} + (p_x, p_y, p_z)$ to the point $\delta(C_p) = p_C = (p_x, p_y, p_z)$ we have that p lies above C if and only if p_C lies below C_p, in other words $\underline{\mu}(P, \mathcal{C}) = \bar{\mu}(\delta(\mathcal{C}), \delta(P))$.

We can use this to tune our algorithm as follows. The recursion proceeds as earlier except that when $m_\Delta > n_\Delta^3$, we switch the roles of P_Δ and $\mathcal{C}_\Delta$ using the duality transformation δ, and compute $\underline{\mu}(P_\Delta, \mathcal{C}_\Delta) = \bar{\mu}(\delta(\mathcal{C}_\Delta), \delta(P_\Delta))$ in $T(m_\Delta, n_\Delta) = O(n_\Delta^{3+\epsilon} + m_\Delta \log n_\Delta) = O(m_\Delta^{1+\epsilon})$ time with the algorithm just described. With these modifications, the recurrence becomes

$$T(n,m) = \begin{cases} \displaystyle\sum_{\Delta \in \mathcal{A}_\parallel} T\left(n_\Delta, \frac{m}{r}\right) + O(m+n) & \text{if } m \leq n^3 \\ O(m^{1+\epsilon}) & \text{if } m > n^3, \end{cases}$$

where $\sum_{\Delta} n_{\Delta} \leq n$. It can be shown that, for any $\epsilon > 0$,

$$T(n, m) = O((mn)^{3/4+\epsilon} + m^{1+\epsilon} + n^{1+\epsilon}).$$

Thus, we obtain the following result.

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

Note that we can use the same approach to report all pairs of vertices for which the dilation is larger than κ, in additional time that is proportional to the size of the output.

The same result can be obtained for trees, using a decomposition technique that will be presented in Section 3.4 for computing the approximate dilation spectrum.

2.2 Cycles

Let us now consider the case in which P is a simple closed polygonal curve. This case is more difficult because there are two paths between any two points x, y of P. Let $P[x, y]$ denote the portion of P from x to y in clockwise direction, and let $d_P(x, y)$ denote its length. Moreover, for a vertex p of P, let $\nu(p)$ denote the last vertex on P in clockwise direction such that $d_P(p, \nu(p))$ does not exceed half the perimeter of P.

Suppose we are given four vertices $t_1, t_2, b_1 = v(t_1), b_2 = v(t_2)$ of P in clockwise order, and we want to compute

$$\pi(t_1, t_2, b_1, b_2) := \pi_P(\kappa, P[t_1, t_2], P[b_1, b_2]).$$

First observe that $d_P(b_1, b_2) \leq |P|/2$ holds. Let m, n be the number of edges in $P[b_1, b_2]$ and $P[t_1, t_2]$, respectively. If $\min\{m, n\} = 1$, then we compute $\pi(t_1, t_2, b_1, b_2)$ in $O(m + n)$ time, by brute force. Otherwise, suppose that $n \geq m$ holds. Let t be the median vertex of $P[t_1, t_2]$, and let $b = \nu(t)$. Clearly, $b \in P[b_1, b_2]$ and

$$\pi(t_1, t_2, b_1, b_2) = \pi(t_1, t, b, b_2) + \pi(t, t_2, b_1, b) + \pi(t_1, t, b_1, b) + \pi(t, t_2, b, b_2).$$

The quantities $\pi(t_1, t, b_1, b)$ and $\pi(t, t_2, b, b_2)$ are computed recursively. Since $P[t, t_2]$ and $P[b_1, b]$ lie in $P[t, \nu(t)]$, we can compute $\pi(t, t_2, b_1, b)$ according to Theorem 1 in $O((n+m)^{3/2+\epsilon})$ randomized expected time for any $\epsilon > 0$. Almost the same argument applies to $\pi(t_1, t, b, b_2)$.

Let m_1 be the number of edges in $P[b_1, b]$. Then $P[b, b_2]$ contains at most $m - m_1 + 1$ edges. Let $T(n, m)$ denote the maximum expected time of computing $\pi(t_1, t_2, b_1, b_2)$. Then we obtain the following recurrence for any $\epsilon > 0$:

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

with a symmetric inequality for $m \geq n$, and $T(n,1) = O(n), T(1,m) = O(m)$. The solution to the above recurrence is

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

To compute $\pi_P(\kappa)$, we choose a vertex $v \in P$ and let $P_1 = P[v, \nu(v)]$ and $P_2 = P[\nu(v), \nu(\nu(v))]$. In

$$\pi_P(\kappa) = \pi_{P_1}(\kappa) + \pi_{P_2}(\kappa) + \pi(v, \nu(v), \nu(v), \nu(\nu(v)))$$

the values π_{P_1}, π_{P_2} can be computed in $O(n^{3/2+\epsilon})$ expected time using Theorem 1, and $\pi(v, \nu(v), \nu(v), v)$ can be computed within the same time by the recursion just described. We obtain

Theorem 2. *Let P be a closed polygonal path on n vertices in $\mathbb{R}^2$, and let $\kappa \geq 1$. Then we can compute $\pi_P(\kappa)$ in $O(n^{3/2+\epsilon})$ randomized expected time.*

3 Computing the Approximate Dilation Spectrum

Now we set out to give faster algorithms for computing an approximation of the dilation spectrum. Our reduction uses the well-separated pair decomposition, thus adding to the list of applications of this powerful method.

3.1 Well-separated pairs

We briefly review this decomposition and some of its relevant properties. Let d be a fixed dimension, and let s denote a fixed constant, called the *separation constant*. Two point sets A, B in $\mathbb{R}^d$ are *well separated* with respect to s if they can be circumscribed by two disjoint d-dimensional balls of some radius r that are at least $s \cdot r$ apart. Then the distance between any two points of the same set is at most $1 + 4/s$ times the distance between any two points of different sets, while point pairs of different sets differ in distance by at most a factor $2/s$; these basic WSPD properties will be used in the sequel.

Given a set S of n points, 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 points $p \neq q$ of S, there is a unique index i such that $p \in A_i$ and $q \in B_i$ holds, or vice versa.

Callahan and Kosaraju [3] showed that a WSPD of size $k = O(n)$ always exists, and that it can be efficiently computed in time $O(n \log n)$ using a split tree. We are going to use a modified version of their result where each pair $\{A_i, B_i\}$ contains a singleton set and size k is allowed to be in $O(n \log n)$.

3.2 A general algorithm

Given a real number $\kappa > 1$ and a geometric graph G, we show how to compute the number $\rho_G(\kappa)$ of all pairs of vertices in the graph for which the dilation is

at most κ.[5] Now consider a WSPD

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

for the set of vertices of G. We may assume that each A_i is a singleton set, $\{a_i\}$ and that $k = O(n \log n)$. We know that all geometric distances between a_i and a point of B_i are roughly equal. Let the distance between a_i and any representative point in B_i be denoted by D_i. Hence, if we compute for each i, $1 \leq i \leq k$, all points $b_i \in B_i$ whose distance $d_G(a_i, b_i)$ in G is at most $\kappa(1+\epsilon) \cdot D_i$, then we would have effectively computed all pairs of points in the pair $\{A_i, B_i\}$ for which their approximate dilation is at most κ. This observation leads to the general algorithm, $\mathcal{A}$, presented below in Figure 3.2; it could be easily modified

General Algorithm $\mathcal{A}$
Input: A geometric graph G on a set S of points in $\mathbb{R}^d$ and a constant $\epsilon > 0$.
Output: The number of pairs of vertices of G of dilation at most $\kappa(1+\epsilon)$.
Step 1: Using separation constant $s = 4/\epsilon$, compute a WSPD

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

for the set S, with the added condition that $|A_i| = 1$, for each $i = 1, \ldots, k$.
Step 2: For each i, $1 \leq i \leq k$, let $A_i = \{a_i\}$, and D_i denote the length $|a_i b|$, where b is an arbitrary element of B_i. For $i = 1, \ldots, k$. compute n_i, the number of points $b_i \in B_i$, such that $d_G(a_i, b_i) \leq \kappa(1+\epsilon) \cdot D_i$.
Step 3: Report $N = \sum_{i=1}^{k} n_i$.

to output the pairs of points of dilation $\leq \kappa(1+\epsilon)$.

The following lemma compares N, the output of Algorithm $\mathcal{A}$, with the true value of $\rho_G(\kappa)$.

Lemma 2. *The number of pairs of vertices in G with stretch factor at most κ is at most equal to N. The number of pairs of vertices in G with stretch factor at most $\kappa(1+\epsilon)^2$ is at least equal to N. In other words,*

$$\rho_G(\kappa) \leq N \leq \rho_G(\kappa(1+\epsilon)^2)$$

Proof. In step 2 of algorithm $\mathcal{A}$, D_i is equal to $|a_i b|$, where b is an arbitrary element of B_i. Step 2 of the algorithm counts all pairs of points $b_i \in B_i$ for which $|a_i b_i| \leq \kappa(1+\epsilon) \cdot D_i$.

If $d_G(a_i, b_i) \leq \kappa |a_i b_i|$, then by the WSPD properties, $d_G(a_i, b_i) \leq \kappa(1+\epsilon) \cdot D_i$, and thus the pair (a_i, b_i) is counted in step 2 of the algorithm, proving that $\rho_G(\kappa) \leq N$.

If $d_G(a_i, b_i) \leq \kappa(1+\epsilon) D_i$, then it is counted in step 2 of the algorithm. But then, by the WSPD property, $d_G(a_i, b_i) \leq \kappa(1+\epsilon)^2 \cdot |a_i b_i|$, implying that this pair has a dilation of at most $\kappa(1+\epsilon)^2$. Thus, $N \leq \rho_G(\kappa(1+\epsilon)^2)$.

This completes the proof.

[5] Clearly, we can obtain the number $\pi_G(\kappa)$ of vertex pairs whose dilation exceeds κ by subtracting $\rho_G(\kappa)$ from $\binom{n}{2}$.

3.3 Paths

Let the graph G be a simple path $(p_0, p_1, \ldots, p_{n-1})$ on the points of the set S, and let $\epsilon > 0$ be a constant.

Following our general algorithm $\mathcal{A}$ of Section 3.2, we start by computing a split tree T and a corresponding WSPD

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}$$

for S, with separation constant $s = 4/\epsilon$, where $|A_i| = 1$, for $1 \leq i \leq m$, and $m = O(n \log n)$. This can be done in time $O(n \log n)$.

By a simple postorder traversal of T, we store with each node u of T the sorted list of indices of all points stored at the leaves of the subtree of u. Note that this involves merging two sorted sublists at each node of the tree. Each set B_i is represented by a node v_i in the split tree T such that their subtrees contain exactly the points of B_i in their leaves. Let the sorted list of the points for node v_i be denoted by V_i. Also since $|A_i| = 1$, it is represented by a leaf node u_i in the split tree.

Step 2 of our algorithm is implemented as follows. First, we traverse the path, and compute for each vertex p_j, $0 < j < n$, the distance $d_G(p_0, p_j)$. Using this information, we can compute for any $0 \leq j < k < n$, the distance $d_G(p_j, p_k)$ in constant time, as the difference between $d_G(p_0, p_k)$ and $d_G(p_0, p_j)$.

Then, for $1 \leq i \leq m$, use binary search to identify the two sublists of points in V_i that lie before and after point a_i. It is easy to observe that the two sublists are effectively sorted by their distance from a_i. Finally use two binary searches to search the two sublists separately to identify the number of points $b_i \in V_i$ such that $d_G(a_i, b_i) \leq \kappa(1 + \epsilon)D_i$.

It is easy to see that this correctly implements Step 2. Since it involves $2m = O(n \log n)$ binary searches, the running time of the entire algorithm is bounded by $O(n \log^2 n)$.

Theorem 3. *Let S be a set of n points in $\mathbb{R}^d$, let G be a simple path on the points of S, and let ϵ be a positive constant. In $O(n \log^2 n)$ time, we can compute a number N that lies between $\rho_G(\kappa)$ and $\rho_G(\kappa(1 + \epsilon)^2)$.*

Due to space limitations we can only mention that the same result can be obtained for cycles.

3.4 Trees

It is well known that in any tree G having n vertices, there is a vertex v, whose removal gives two graphs G'_1 and G'_2, each having at most $2n/3$ vertices. Moreover, such a vertex v can be found in $O(n)$ time. Each of the two graphs G'_i is a forest of trees. We will call v a *centroid vertex* of G. Each of the graphs $G_1 := G'_1 \cup \{v\}$ and $G_2 := G'_2 \cup \{v\}$ is connected and, hence, a tree again.

Let S be a set of n points in $\mathbb{R}^d$, and let G be an arbitrary tree having the points of S as its vertices. We will identify the vertices of G with the points of S.

Let $\epsilon > 0$ be a constant. The following recursive algorithm, which is inspired by the general algorithm $\mathcal{A}$, solves the problem when the input graph G is a tree.

Step 1: Compute a centroid vertex v of G, and the corresponding decomposition into trees G_1 and G_2. Note that v is a vertex of both these trees. Traverse each tree in preorder, and store with each vertex p the distance $d_G(p, v)$ between p and the centroid vertex v.

Step 2: Use the same algorithm to recursively solve the problem on graph G_1 and on graph G_2.

Step 3: Let $s := 4/\epsilon$. Compute a split tree T and a corresponding WSPD

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}$$

for the points of S, with separation constant s, having size $m = O(n \log n)$.

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 of nodes, S_u^1, consisting of nodes in G_1 sorted according to their distance from the centroid v. Similarly, compute the sorted sequence of nodes, S_u^2, consisting of nodes in G_2 sorted according to their distance from the centroid v.

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

The correctness of the above algorithm is proved by induction and results from the fact that Step 2 counts all pairs of nodes that involve a_i and another node in G_1 (assuming that $a_i \in G_1$), while Step 5 counts all pairs of nodes that involve a_i and another node in G_2. Note that the distance in G from point $a_i \in G_1$ to point $b_i \in G_2$ is given by adding their distances to the centroid vertex v.

To prove the time complexity, let $\mathcal{T}(n)$ denote the running time of our 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 4. *Let S be a set of n points in $\mathbb{R}^d$, let G be a tree on the points of S, and let ϵ be a positive constant. In $O(n \log^3 n)$ time, we can compute a number N such that it lies between $\rho_G(\kappa)$ and $\rho_G(\kappa(1+\epsilon)^2)$.*

4 Open Problems

Besides the obvious questions about possible improvement or generalization of our algorithms, there seem to be some interesting structural problems one may

want to study. To what extend does the dilation spectrum of a graph, that is, the function $\pi_G(\kappa)$, reflect the graph's structure? And, what types of functions $\pi_G(\kappa)$ can occur?

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.

References

1. P. Agarwal, R. Klein, Ch. 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*, to appear in Discrete and Computational Geometry.
2. P. Agarwal, R. Klein, Ch. Knauer, and M. Sharir, *Computing the detour of polygonal curves*, Technical Report B 02-03, Freie Universität Berlin, Fachbereich Mathematik und Informatik, 2002.
3. P. B. Callahan and S. R. Kosaraju, *A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields*, J. ACM, 42 (1995), pp. 67–90.
4. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
5. D. Eppstein, *Spanning trees and spanners*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier Science, Amsterdam, 1999, pp. 425–461.
6. M. Farshi, P. Giannopoulos, and J. Gudmundsson, *Finding the best shortcut in a geometric network*, 21st Ann. ACM Symp. Comput. Geom. (2005), pp. 327–335.
7. G. N. Frederickson, *Fast algorithms for shortest paths in planar graphs, with applications*, SIAM J. Comput., 16 (1987), pp. 1004–1022.
8. D. Haussler and E. Welzl, *Epsilon-nets and simplex range queries*, Discrete Comput. Geom. 2 (1987), pp. 127–151.
9. S. Langerman, P. Morin, and M. Soss, *Computing the maximum detour and spanning ratio of planar chains, trees and cycles*, In *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS 2002)*, volume 2285 of *LNCS*, pages 250–261. Springer-Verlag, 2002.
10. N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245.
11. G. Narasimhan and M. Smid, *Approximating the stretch factor of Euclidean Graphs*, SIAM J. Comput., 30(3) (2000), pp. 978–989.
12. G. Narasimhan and M. Smid, *Geometric Spanner Networks*, Cambridge University Press, to appear.
13. D. Peleg and A. Schäffer, *Graph spanners*, J. Graph Theory, 13 (1989), pp. 99–116.
14. M. Sharir and P. Agarwal, *Davenport-Schinzel sequences and their geometric applications*, Cambridge University Press, New York, 1995.
15. M. Smid, *Closest-point problems in computational geometry*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier Science, Amsterdam, 1999, pp. 877–935.