APPROXIMATING THE STRETCH FACTOR OF EUCLIDEAN GRAPHS*

GIRI NARASIMHAN† AND MICHIEL SMID‡

Abstract. There are several results available in the literature dealing with efficient construction of t-spanners for a given set S of n points in $\mathbb{R}^d$. t-spanners are Euclidean graphs in which distances between vertices in G are at most t times the Euclidean distances between them; in other words, distances in G are "stretched" by a factor of at most t. We consider the interesting dual problem: given a Euclidean graph G whose vertex set corresponds to the set S, compute the stretch factor of G, i.e., the maximum ratio between distances in G and the corresponding Euclidean distances. It can trivially be solved by solving the All-Pairs-Shortest-Path problem. However, if an approximation to the stretch factor is sufficient, then we show it can be efficiently computed by making only $O(n)$ approximate shortest path queries in the graph G. We apply this surprising result to obtain efficient algorithms for approximating the stretch factor of Euclidean graphs such as paths, cycles, trees, planar graphs, and general graphs. The main idea behind the algorithm is to use Callahan and Kosaraju's well-separated pair decomposition.

Key words. computational geometry, spanners, approximate shortest paths, well-separated pairs

AMS subject classifications. 68U05

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*. The length of a path in G is defined as the sum of the lengths of all edges on this path. For any two vertices p and q of G, we denote by $|pq|_G$ the distance in G between them, i.e., the length of a shortest path connecting p and q.

Let $t > 1$ be a real number. We say that G is a *t-spanner* for S, if for each pair of points $p, q \in S$, we have $|pq|_G \leq t \cdot |pq|$, i.e., there exists a path in G between p and q of length at most t times the Euclidean distance between these two points.

The smallest t such that G is a t-spanner for S is called the *stretch factor* of G (also referred to as *dilation* [21] or *distortion* [19] in the literature). We will denote the stretch factor by t^*. Note that

$$t^* = \max \left\{ \frac{|pq|_G}{|pq|} : p, q \in S, p \neq q \right\}.$$

Most of the earlier research considered the problem of constructing or analyzing geometric t-spanners for a given set of points. In this paper, we consider the interesting dual problem:

Problem: *Given a set S of n points in $\mathbb{R}^d$, and a connected Euclidean graph with vertices from S, design an efficient algorithm to compute (exactly or approximately) its stretch factor.*

*The work of GN was supported by NSF Grant CCR-940-9752, and a grant by Cadence Design Systems.

†Department of Mathematical Sciences, The University of Memphis, Memphis TN 38152 (`giri@msci.memphis.edu`).

‡Department of Computer Science, University of Magdeburg, D-39106 Magdeburg, Germany (`michiel@isg.cs.uni-magdeburg.de`).

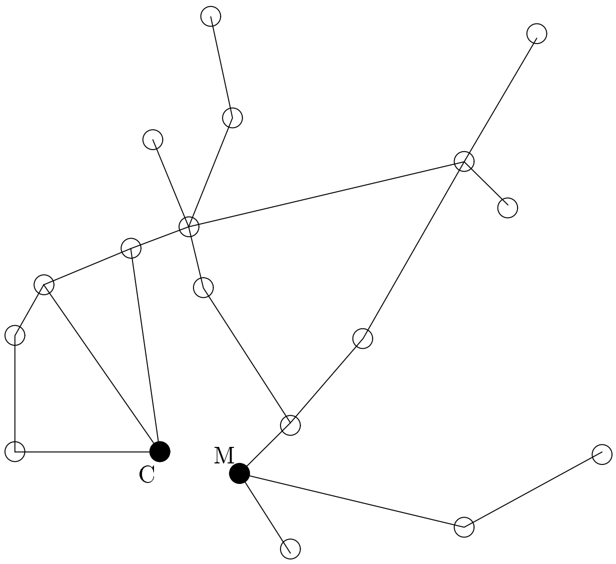

FIG. 1.1. *A section of the Scandinavian rail network; C denotes Copenhagen, and M denotes Malmö.*

Spanners have applications in network design, robotics, distributed algorithms, and many other areas, and have been the subject of considerable research [1, 4, 8, 11, 14, 18, 24]. More recently, spanners have received a lot of attention by researchers with the discovery of new applications for them in the design of approximation algorithms for geometric optimization problems such as the Euclidean traveling salesperson problem [3, 23].

If the graph represents, say, a network of highways, then the stretch factor is a measure of the maximum percentage increase in driving distance for using the network of highways over the direct "as-the-crow-flies" distance. Thus the stretch factor of a network is an important parameter to be considered when evaluating and analyzing networks. Furthermore, determining the two vertices in the network for which this increase is maximized helps to identify the "weakest" part of the network in terms of distances.

Figure 1.1 shows a section of the Scandinavian rail network. It is clear that the stretch factor in this network is determined by Copenhagen (marked by a C) and Malmö (marked by a M). A link between these two cities would drastically reduce the stretch factor[1].

Let $G = (S, E)$ be a Euclidean graph, and let $n := |S|$ and $m := |E|$. Clearly, the time complexity of solving the *All-Pairs-Shortest-Path* problem for G is an upper bound on the time complexity of computing the stretch factor of G. Hence, running Dijkstra's algorithm—implemented with Fibonacci heaps—from each vertex of G, gives the stretch factor of G, in $O(n^2 \log n + nm)$ time (c.f., [10]). For some classes of graphs, better running times can be obtained. For example, if G is a planar Euclidean graph, Frederickson [15] has shown that the distances in G between all pairs of vertices can be computed in $O(n^2)$ total time. Therefore, the stretch factor of a planar Euclidean graph can be computed in $O(n^2)$ time.

[1] A 16-kilometer bridge across the Øresund connecting the two cities is currently being built.

We are not aware of any algorithms that compute the stretch factor in subquadratic time, for any class of connected Euclidean graphs. (Exceptions are trivial classes of graphs, such as complete graphs, which have stretch factor one.) For example, we do not even know if the stretch factor of a Euclidean path can be computed in $o(n^2)$ time. This leads to the question whether there are faster algorithms that *approximate* stretch factors.

Let G be a connected Euclidean graph with stretch factor t^*, and let $c_1 \geq 1$, $c_2 \geq 1$, and $t \geq 1$ be real numbers. We say that t is a (c_1, c_2)-*approximate stretch factor* of G, if

$$\frac{1}{c_1} t \leq t^* \leq c_2 t.$$

1.1. Our results. The results of this paper are as follows.

1. Using the *well-separated pair decomposition* of Callahan and Kosaraju [7], we reduce the problem of approximating the stretch factor of any Euclidean graph G to a sequence of $O(n)$ approximate shortest path queries in G.
2. We prove that, in the algebraic computation tree model, any algorithm that takes as input any connected Euclidean graph G with n vertices, and computes an approximation to the stretch factor of G, takes $\Omega(n \log n)$ time in the worst case.
3. For any real constant $\epsilon > 0$, we can compute in $O(n \log n)$ time, a $(1, 1+\epsilon)$-approximate stretch factor of any Euclidean path, cycle, or tree with n vertices. By the previous result, this is optimal in the algebraic computation tree model.
4. For any real constant $\epsilon > 0$, we can compute in $O(n\sqrt{n})$ time, a $(1, 1+\epsilon)$-approximate stretch factor of any planar Euclidean graph with n vertices.
5. For any integer constant $\beta \geq 1$, and real constant $\epsilon > 0$, we can compute in $O(mn^{1/\beta} \log^2 n)$ expected time, a $(2\beta(1+\epsilon), 1+\epsilon)$-approximate stretch factor of any Euclidean graph with n vertices and m edges.

In our first algorithm (Algorithm $\mathcal{A}$), the stretch factor is approximated by making farthest pair queries on $O(n)$ pairs of *sets of points*. Except for graphs such as paths, cycles, trees and planar graphs, it is not clear how such queries can be solved efficiently. Our second algorithm (Algorithm $\mathcal{B}$) is much simpler; it makes shortest path queries for $O(n)$ specific pairs of *points*. The time complexity of algorithm $\mathcal{B}$ is consequently improved over the corresponding one for algorithm $\mathcal{A}$.

It is interesting that $O(n)$ approximate shortest path queries are sufficient to approximate the stretch factor. It is also interesting to note the pairs of points on which these $O(n)$ shortest path queries are made. In Algorithm $\mathcal{B}$, these linear number of queries depend *only* on the positions of the vertices; they do *not* depend on the edges of the graph G. Finally, our algorithms also determine two vertices for which the stretch factor is approximately maximized.

1.2. Related work. As mentioned already, our reduction uses the well-separated pair decomposition of [7], thus adding to the list of applications of this powerful method. For other applications of this decomposition, see [4, 6, 7].

Some related research in the general direction of approximating stretch factors include papers by Dobkin et al. [12], and Keil and Gutwin [17], which showed that the Delaunay triangulation has a stretch factor bounded by a small constant, and a paper by Eppstein [13], which showed that a certain class of Euclidean graphs (called beta-skeletons) can have arbitrarily large stretch factors. Note that these papers analyze

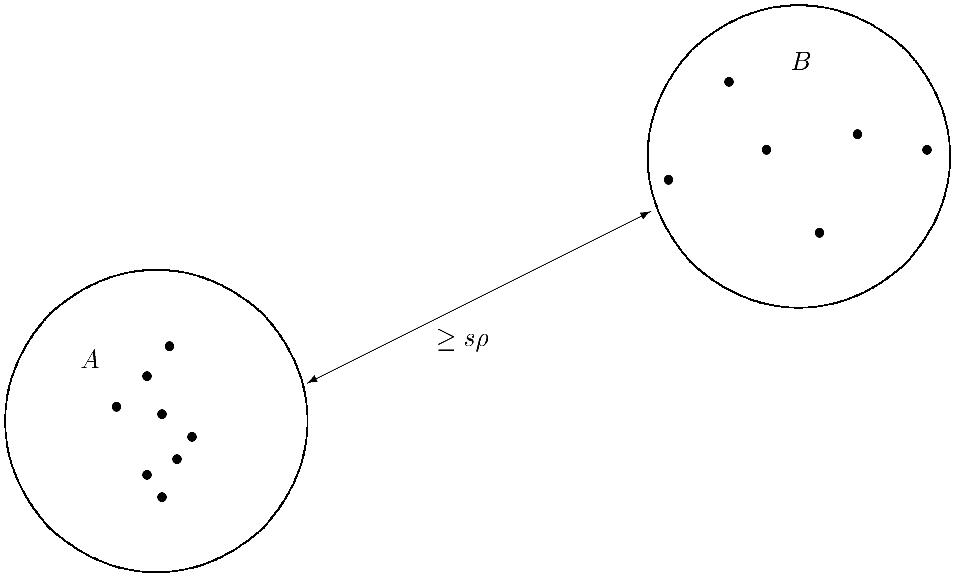

FIG. 2.1. *Two planar point sets A and B that are well-separated w.r.t. s. Both circles have radius ρ; their distance is at least $s\rho$.*

the largest possible stretch factor of *any* Delaunay triangulation or beta-skeleton. For example, Keil and Gutwin showed that for any finite set of points in the plane, the stretch factor of its Delaunay triangulation is bounded from above by $\frac{2\pi}{3\cos\pi/6}$. Clearly, for some sets of points, the stretch factor can be much smaller. The current paper represents the first attempt at devising algorithms to efficiently approximate the stretch factor of a given Euclidean graph.

2. Well-separated pairs. Our algorithms use the *well-separated pair decomposition* devised by Callahan and Kosaraju [7]. We briefly review this decomposition and some of its relevant properties.

DEFINITION 2.1. *Let $s > 0$ be a real number, and let A and B be two finite sets of points in $\mathbb{R}^d$. We say that A and B are* well-separated w.r.t. s, *if there are two disjoint d-dimensional balls C_A and C_B, having the same radius, such that (i) C_A contains all points of A, (ii) C_B contains all points of B, and (iii) the distance between C_A and C_B is at least equal to s times the radius of C_A.*

See Figure 2.1 for an illustration. In this paper, s will always be a constant, called the *separation constant.* The following lemma follows easily from Definition 2.1.

LEMMA 2.2. *Let A and B be two finite sets of points that are well-separated w.r.t. s, let a and p be points of A, and let b and q be points of B. Then*

1. $|ab| \leq (1 + 4/s)|pq|$,
2. $|pa| \leq (2/s)|pq|$.

DEFINITION 2.3 ([7]). *Let S be a set of n points in $\mathbb{R}^d$, and $s > 0$ a real number. A* well-separated pair decomposition *(WSPD) for S (w.r.t. s) is a sequence of pairs of non-empty subsets of S,*

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

General Algorithm $\mathcal{A}$
The algorithm takes as input a Euclidean graph G on a set S of points in $\mathbb{R}^d$, and a real constant $\epsilon > 0$.
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.
Step 2: For each i, $1 \leq i \leq k$, compute two points a_i and b_i, where $a_i \in A_i$ and $b_i \in B_i$, such that

$$|a_i b_i|_G = \max\{|pq|_G : p \in A_i, q \in B_i\},$$

and compute $t_i := |a_i b_i|_G / |a_i b_i|$.
Step 3: Report the value of t, defined as $t := \max(t_1, t_2, \ldots, t_k)$. Also report points a_i and b_i for which $t = t_i$.

FIG. 3.1. *The first general algorithm for approximating the stretch factor of a Euclidean graph.*

such that
1. $A_i \cap B_i = \emptyset$, *for all* $i = 1, 2, \ldots, k$,
2. *for any two distinct points* p *and* q *of* S, *there is exactly one pair* $\{A_i, B_i\}$ *in the sequence, such that*
   (a) $p \in A_i$ *and* $q \in B_i$, *or*
   (b) $p \in B_i$ *and* $q \in A_i$,
3. A_i *and* B_i *are well-separated w.r.t.* s, *for all* $i = 1, 2, \ldots, k$.

The integer k *is called the* size *of the WSPD.*

Callahan and Kosaraju showed how such a WSPD of size $k = O(n)$ can be computed using a binary tree, called the *fair split tree.*

THEOREM 2.4 ([7]). *Let* S *be a set of* n *points in* $\mathbb{R}^d$, *and* $s > 0$ *a separation constant. In* $O(n \log n + \alpha_{ds} n)$ *time, we can compute a WSPD for* S *of size at most* $\alpha_{ds} n$. *The constant in the Big-Oh bound does not depend on* s. *Moreover, for a large separation constant* s, *the value of* α_{ds} *is proportional to* $2^d d^{d/2} s^d$.

3. The first general algorithm. Let S be a set of n points in $\mathbb{R}^d$, and let G be a connected Euclidean graph having the points of S as its vertices. Recall that the stretch factor t^* of G is equal to

$$t^* = \max\left\{\frac{|pq|_G}{|pq|} : p, q \in S, p \neq q\right\}.$$

Consider a WSPD

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

for S. It follows from Lemma 2.2 that all Euclidean distances between a point of A_i and a point of B_i are roughly equal. Hence, if we compute for each i, $1 \leq i \leq k$, a point $a_i \in A_i$ and a point $b_i \in B_i$ whose distance $|a_i b_i|_G$ in G is maximum, then the largest value of $|a_i b_i|_G / |a_i b_i|$ should be a good approximation to the stretch factor t^* of G. This observation leads to our first general algorithm, which we denote by $\mathcal{A}$. See Figure 3.1. The following lemma proves the correctness of algorithm $\mathcal{A}$.

LEMMA 3.1. *The value of t reported by algorithm $\mathcal{A}$ is a $(1, 1+\epsilon)$-approximate stretch factor of G.*

Proof. Let t^* be the stretch factor of the graph G. We have to show that $t \leq t^* \leq (1+\epsilon)t$. Since t can be written as $|pq|_G/|pq|$ for some points p and q in S, $p \neq q$, it is clear that $t \leq t^*$.

To prove the second inequality, let x and y be two points of S such that $t^* = |xy|_G/|xy|$. Let i be the (unique) index such that (i) $x \in A_i$ and $y \in B_i$, or (ii) $x \in B_i$ and $y \in A_i$. Assume w.l.o.g. that (i) holds.

Consider the points $a_i \in A_i$ and $b_i \in B_i$ that were chosen in Step 2 of the algorithm. Clearly, $|xy|_G \leq |a_i b_i|_G$. By Lemma 2.2, we have $|a_i b_i| \leq (1 + 4/s)|xy| = (1+\epsilon)|xy|$. This gives

$$t^* = \frac{|xy|_G}{|xy|} \leq \frac{|a_i b_i|_G}{|xy|} \leq (1+\epsilon)\frac{|a_i b_i|_G}{|a_i b_i|} = (1+\epsilon)t_i \leq (1+\epsilon)t.$$

This completes the proof. □

4. An improved algorithm. The main problem with the general algorithm $\mathcal{A}$ presented in the previous section is that Step 2 is hard to implement efficiently. In a preliminary version of this paper ([20]), we showed that algorithm $\mathcal{A}$ can be used to compute a $(1, 1+\epsilon)$-approximation to the stretch factor of Euclidean paths, cycles, and trees, in $O(n \log n)$, $O(n \log n)$, and $O(n \log^2 n)$ time, respectively. Using results from Arikati *et al.* [2], we were able to design an $O(n^{5/3} polylog(n))$-time algorithm for computing a $(2, 1+\epsilon)$-approximation to the stretch factor of planar Euclidean graphs.

In this section, we give a much simpler approximation algorithm. Recall that in algorithm $\mathcal{A}$, we compute for each pair $\{A_i, B_i\}$ in a WSPD for S, a point $a_i \in A_i$ and a point $b_i \in B_i$ for which $|a_i b_i|_G$ is maximum, and use $|a_i b_i|_G/|a_i b_i|$ as a candidate for the approximate stretch factor. Below, we prove that we can take *arbitrary* points $a_i \in A_i$ and $b_i \in B_i$, and use $|a_i b_i|_G/|a_i b_i|$, or an approximation to this quantity, as a candidate. Note that this is counter-intuitive, because the distances in the graph G between points of A_i and points of B_i can vary greatly.

Hence, the problem of approximating the stretch factor of a Euclidean graph can be reduced to the problem of making $O(n)$ (approximate) shortest path query computations. Shortest path query computations can, in general, be implemented more efficiently than the *farthest pair* (between sets of vertices) computations that were required when implementing algorithm $\mathcal{A}$.

This improved algorithm, which we denote by $\mathcal{B}$, will be given in Section 4.1 below. In Section 5, we show that algorithm $\mathcal{B}$ achieves, in subquadratic time, comparable approximation ratios for various classes of Euclidean graphs, as compared to algorithm $\mathcal{A}$.

4.1. The reduction. Let p and q be two distinct vertices of a connected Euclidean graph G, and let $c \geq 1$ be a real number. We say that the real number $L(p,q)$ is a *c-approximation* to the length of a shortest path in G between p and q, if

$$|pq|_G \leq L(p,q) \leq c \cdot |pq|_G.$$

Let $\mathcal{G}$ be a class of connected Euclidean graphs. We assume that we are given an algorithm ASP_c that takes as input (i) any graph G from the class $\mathcal{G}$, and (ii) any sequence of pairs of vertices of G; the algorithm ASP_c computes for each pair (a, b) in this sequence, a c-approximation to $|ab|_G$.

Improved Algorithm $\mathcal{B}$
The algorithm takes as input a Euclidean graph G from the class $\mathcal{G}$, on a set S of points in $\mathbb{R}^d$, and a real constant $\epsilon > 0$.
Step 1: Using separation constant $s = 4(1+\epsilon)/\epsilon$, compute a WSPD

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

for S. For each i, $1 \le i \le k$, take an arbitrary point $a_i \in A_i$, and an arbitrary point $b_i \in B_i$.
Step 2: Use algorithm ASP_c to compute, for each i, $1 \le i \le k$, a c-approximation $L(a_i, b_i)$ to the length $|a_i b_i|_G$ of a shortest path in G between a_i and b_i. For each i, $1 \le i \le k$, let

$$t_i := \frac{L(a_i, b_i)}{|a_i b_i|}.$$

Step 3: Report the value of t, defined as $t := \max(t_1, t_2, \ldots, t_k)$. Also report points a_i and b_i for which $t = t_i$.

FIG. 4.1. *The improved algorithm for approximating the stretch factor of a Euclidean graph.*

Algorithm $\mathcal{B}$ is given in Figure 4.1. The following theorem bounds the performance ratio of the output of algorithm $\mathcal{B}$.

THEOREM 4.1. *Let G be a Euclidean graph from the class $\mathcal{G}$ on a set S of points in $\mathbb{R}^d$, let t^* be the stretch factor of G, and let t be the value that is reported by algorithm $\mathcal{B}$. Then*

$$\frac{1}{c} t \le t^* \le (1+\epsilon)^2 t,$$

i.e., t is a $(c, (1+\epsilon)^2)$-approximate stretch factor of G.

Proof. Let i be the index such that $t = t_i = L(a_i, b_i)/|a_i b_i|$. Then,

$$t \le c \frac{|a_i b_i|_G}{|a_i b_i|} \le c t^*.$$

In the rest of the proof, we will prove the second inequality. To be more precise, we will show that for all points $p, q \in S$, $p \neq q$,

$$\frac{|pq|_G}{|pq|} \le (1+\epsilon)^2 t.$$

This will prove that $t^* \le (1+\epsilon)^2 t$.

The proof is by induction on the rank of the distance $|pq|$ in the sorted sequence of $\binom{n}{2}$ distances in S. To start the induction, assume that p, q is a closest pair in S. Let i be the index such that (i) $p \in A_i$ and $q \in B_i$, or (ii) $p \in B_i$ and $q \in A_i$. Assume w.l.o.g. that (i) holds. Since $s > 2$, it follows from Lemma 2.2 that $A_i = \{p\}$ and $B_i = \{q\}$. Hence, in Step 2 of the algorithm, we have computed the value $t_i = L(p,q)/|pq|$. It follows that

$$\frac{|pq|_G}{|pq|} \le \frac{L(p,q)}{|pq|} = t_i \le t < (1+\epsilon)^2 t.$$

Now assume that p, q is not a closest pair in S and, moreover, assume that $|xy|_G/|xy| \leq (1+\epsilon)^2 t$ for all pairs x, y of points of S such that $x \neq y$ and $|xy| < |pq|$. Let i be the index such that (i) $p \in A_i$ and $q \in B_i$, or (ii) $p \in B_i$ and $q \in A_i$. Again, assume w.l.o.g. that (i) holds. Consider the point $a_i \in A_i$, and the point $b_i \in B_i$ that were chosen in Step 2 of the algorithm. By the triangle inequality, we have

$$|pq|_G \leq |pa_i|_G + |a_i b_i|_G + |b_i q|_G. \tag{4.1}$$

We distinguish three cases, depending on whether $|pa_i|_G$ and $|b_i q|_G$ are smaller or larger than $(\epsilon/2)|a_i b_i|_G$, respectively.

Case 1: $|pa_i|_G > (\epsilon/2)|a_i b_i|_G$ and $|b_i q|_G > (\epsilon/2)|a_i b_i|_G$.

First note that $p \neq a_i$, because $|pa_i|_G > 0$. We may assume w.l.o.g. that $|b_i q|_G \leq |pa_i|_G$. It follows from (4.1) that

$$|pq|_G < \frac{2(1+\epsilon)}{\epsilon} |pa_i|_G.$$

Since $s = 4(1+\epsilon)/\epsilon$, Lemma 2.2 implies that $|pa_i| \leq \frac{\epsilon}{2(1+\epsilon)}|pq|$. Therefore,

$$\frac{|pq|_G}{|pq|} < \frac{2(1+\epsilon)}{\epsilon} \frac{|pa_i|_G}{|pq|} \leq \frac{|pa_i|_G}{|pa_i|}.$$

Since $|pa_i| < |pq|$, the induction hypothesis implies that

$$\frac{|pq|_G}{|pq|} < \frac{|pa_i|_G}{|pa_i|} \leq (1+\epsilon)^2 t.$$

Case 2: $|pa_i|_G \leq (\epsilon/2)|a_i b_i|_G$.

In this case, (4.1) implies that

$$|pq|_G \leq (1+\epsilon/2)|a_i b_i|_G + |b_i q|_G.$$

Moreover, Lemma 2.2 implies that $|b_i q| \leq \frac{\epsilon}{2(1+\epsilon)}|pq|$, and $|a_i b_i| < (1+\epsilon)|pq|$.

First assume that $b_i \neq q$. Then,

$$\begin{aligned}
\frac{|pq|_G}{|pq|} &\leq (1+\epsilon/2) \frac{|a_i b_i|_G}{|pq|} + \frac{|b_i q|_G}{|pq|} \\
&< (1+\epsilon/2)(1+\epsilon) \frac{|a_i b_i|_G}{|a_i b_i|} + \frac{\epsilon}{2(1+\epsilon)} \frac{|b_i q|_G}{|b_i q|} \\
&\leq (1+\epsilon/2)(1+\epsilon) \frac{L(a_i, b_i)}{|a_i b_i|} + \frac{\epsilon}{2(1+\epsilon)} \frac{|b_i q|_G}{|b_i q|}.
\end{aligned}$$

Since $|b_i q| < |pq|$, the induction hypothesis implies that $|b_i q|_G/|b_i q| \leq (1+\epsilon)^2 t$. Hence,

$$\frac{|pq|_G}{|pq|} < (1+\epsilon/2)(1+\epsilon)\, t_i + \frac{\epsilon(1+\epsilon)}{2} t \leq (1+\epsilon)^2 t,$$

where the last inequality follows from the fact that $t_i \leq t$.

If $b_i = q$, then basically the same calculation shows that

$$\frac{|pq|_G}{|pq|} < (1+\epsilon/2)(1+\epsilon)\, t_i < (1+\epsilon)^2 t,$$

Case 3: $|b_i q|_G \leq (\epsilon/2)|a_i b_i|_G$.
This case is symmetric to Case 2. □

In the following theorem, we summarize the result of this section. We denote by $T(n,m,k)$, the worst running time of algorithm ASP_c, when given (i) a graph $G \in \mathcal{G}$ having n vertices and m edges, and (ii) a sequence of c-approximate shortest path queries, consisting of k pairs of vertices of G.

THEOREM 4.2. *Let S be a set of n points in $\mathbb{R}^d$, let G be a Euclidean graph from the class $\mathcal{G}$, having the points of S as its vertices, and let ϵ be a real constant such that $0 < \epsilon \leq 3$. We can compute a $(c, 1+\epsilon)$-approximate stretch factor of G, in time*

$$O(n \log n) + T(n, m, \beta_{d\epsilon} n).$$

Here, $\beta_{d\epsilon}$ is a constant which is proportional to $24^d d^{d/2} (1/\epsilon)^d$ if $\epsilon \downarrow 0$.

Proof. Run algorithm $\mathcal{B}$, with ϵ replaced by $\epsilon/3$. Let t be the value that is computed by this algorithm. By Theorem 4.1, we have $t \leq ct^*$, and

$$t^* \leq (1+\epsilon/3)^2 t \leq (1+\epsilon)t.$$

The bound on the running time follows immediately from the algorithm and Theorem 2.4. □

5. Applications of Algorithm $\mathcal{B}$.

5.1. A lower bound. Before we start with the applications, we prove a lower bound for approximating the stretch factor in the algebraic computation tree model. (See Ben-Or [5] or Preparata and Shamos [22] for a description of this model.)

THEOREM 5.1. *Any algebraic computation tree algorithm that takes as input (i) a Euclidean path or cycle on a set of n points in $\mathbb{R}^d$, and (ii) real numbers $c_1 \geq 1$ and $c_2 \geq 1$, and that computes a (c_1, c_2)-approximate stretch factor of this graph, has worst-case running time $\Omega(n \log n)$.*

Proof. We give the lower bound proof for the case when the graph is a path. The lower bound proof for the cycle is similar.

Let $\mathcal{C}$ be any algorithm that satisfies the hypothesis. We will show that $\mathcal{C}$ can be used to solve the *Element-Uniqueness-Problem*, which is known to have an $\Omega(n \log n)$ lower bound in the algebraic computation tree model. (See [5, 22].)

Let $x_1, x_2, \ldots, x_n$ be a sequence of n real numbers. We consider these numbers as points on the x_1-axis in $\mathbb{R}^d$. Let M be the maximal element in the input sequence. Define the path P by

$$P := (x_1, M+1, x_2, M+2, x_3, M+3, \ldots, x_{n-1}, M+n-1, x_n).$$

Note that each edge of P has a non-zero length. We choose arbitrary real numbers $c_1 \geq 1$ and $c_2 \geq 1$, and run algorithm $\mathcal{C}$ on the path P. Let t be the (c_1, c_2)-approximate stretch factor of P that is computed. Then it is easy to see that t is finite if and only if the input numbers $x_1, x_2, \ldots, x_n$ are pairwise distinct.

Since the reduction takes $O(n)$ time, it follows that algorithm $\mathcal{C}$ has a worst-case running time of $\Omega(n \log n)$. □

5.2. Paths, cycles and trees. Let $\mathcal{G}$ be the class of Euclidean paths, cycles, or trees. For any graph G in this class, we can, after an $O(n)$-time preprocessing, answer exact shortest path queries in $O(1)$ time, if G is a path or cycle, and in $O(\log n)$ time, if G is a tree. (If we allow non-algebraic operations, then we can even answer shortest path queries in a tree in $O(1)$ time. See [16].) Hence, we can apply Theorem 4.2, with $c = 1$ and $T(n, m, k) = O(n + k)$, if G is a path or cycle, and $T(n, m, k) = O(n + k \log n)$, if G is a tree, and get the following result.

THEOREM 5.2. *Let S be a set of n points in $\mathbb{R}^d$, let G be a Euclidean path, cycle, or tree, having the points of S as its vertices, and let ϵ be a real constant, such that $0 < \epsilon \leq 3$. In $O(n \log n)$ time, we can compute a $(1, 1 + \epsilon)$-approximate stretch factor of G.*

It follows from Theorem 5.1 that the above result is optimal in the algebraic computation tree model.

5.3. Planar graphs. For the next application, let $\mathcal{G}$ be the class of planar connected Euclidean graphs. Let G be a graph in this class, on a set of n points in $\mathbb{R}^d$. Arikati *et al.* [2] have shown that we can build a data structure, in $O(n\sqrt{n})$ time, that allows us to solve exact shortest path queries, in $O(\sqrt{n})$ time per query. Hence, we can apply Theorem 4.2 with $c = 1$ and $T(n, m, k) = O(n\sqrt{n} + k\sqrt{n})$. This gives the following theorem.

THEOREM 5.3. *Let S be a set of n points in $\mathbb{R}^d$, let G be a planar connected Euclidean graph having the points of S as its vertices, and let ϵ be a real constant, such that $0 < \epsilon \leq 3$. In $O(n\sqrt{n})$ time, we can compute a $(1, 1+\epsilon)$-approximate stretch factor of G.*

5.4. General graphs. In our final application, we let $\mathcal{G}$ be the general class of connected Euclidean graphs. Let $G \in \mathcal{G}$ be any graph with n vertices and m edges. Note that $m \geq n - 1$. Cohen [9] has shown that for any integer $\beta \geq 1$, and any constant ϵ such that $0 < \epsilon \leq 1/2$, any sequence of $(2\beta(1 + \epsilon))$-approximate shortest path queries can be answered in *expected* time

$$O((m + k)n^{1/\beta}\beta \log^2 n).$$

where k is the number of queries. Applying Theorem 4.2 gives the following result.

THEOREM 5.4. *Let S be a set of n points in $\mathbb{R}^d$, let G be a connected Euclidean graph having the points of S as its vertices, and having m edges, let $\beta \geq 1$ be an integer constant, and let ϵ be a real constant, such that $0 < \epsilon \leq 1/2$. In*

$$O(mn^{1/\beta} \log^2 n)$$

expected time, we can compute a $(2\beta(1 + \epsilon), 1 + \epsilon)$-approximate stretch factor of G.

By choosing different values for the integer constant β, Theorem 5.4 gives an interesting trade-off between the running time and the approximation factor. For example, by choosing β large enough, the running time in Theorem 5.4 is almost linear in m, but then the approximation of the stretch factor is very weak (although it is still bounded by a constant).

Theorem 5.4 implies the following result for *sparse* graphs, i.e., graphs having $O(n)$ edges.

COROLLARY 5.5. *Let S be a set of n points in $\mathbb{R}^d$, let G be a sparse connected Euclidean graph having the points of S as its vertices, let $\beta \geq 1$ be an integer constant, and let ϵ be a real constant, such that $0 < \epsilon \leq 1/2$. In*

$$O(n^{1+1/\beta} \log^2 n)$$

expected time, we can compute a $(2\beta(1 + \epsilon), 1 + \epsilon)$-approximate stretch factor of G.

6. Conclusions. In this paper we showed how to efficiently compute a close approximation to the stretch factors of Euclidean graphs. We showed that the problem can be reduced either to a sequence of farthest pair queries on $O(n)$ pairs of sets of points, or to a sequence of (approximate) shortest path queries for $O(n)$ specific pairs of points. It would be interesting to know whether $o(n)$ shortest path queries are sufficient for determining stretch factors approximately.

Except for trivial classes such as complete graphs, it is not known how to compute the exact stretch factor of *any* class of Euclidean graphs in time less than that required to solve the *All-Pairs-Shortest-Path* problem. Hence, it is not even known if the exact stretch factor of simple Euclidean graphs, such as paths, can be computed in subquadratic time.

Stretch factors can be thought of as a quantitative measure to compare distances in two different metrics. In this paper, we demonstrated techniques to compute stretch factors in order to compare a "graph metric" with the Euclidean metric. It would be interesting to study stretch factors as a measure to compare two non-Euclidean metrics. Our techniques cannot be used then, since no equivalent of the well-separated pair decomposition is known for non-Euclidean metrics.

REFERENCES

[1] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete Comput. Geom.*, 9:81–100, 1993.

[2] S. R. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Algorithms—ESA '96, Fourth Annual European Symposium*, volume 1136 of *Lecture Notes Comput. Sci.*, pages 514–528. Springer-Verlag, 1996.

[3] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, pages 33–41, 1998.

[4] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 489–498, 1995.

[5] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[6] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 291–300, 1993.

[7] 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*, 42:67–90, 1995.

[8] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. *Internat. J. Comput. Geom. Appl.*, 5:125–144, 1995.

[9] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28:210–236, 1998.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[11] G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *Internat. J. Comput. Geom. Appl.*, 7:297–315, 1997.

[12] D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete Comput. Geom.*, 5:399–407, 1990.

[13] D. Eppstein. Beta-skeletons have unbounded dilation. Technical Report 96-15, Univ. of California, Irvine, Dept. of Information & Computer Science, Irvine, CA, 92697-3425, USA, 1996.

[14] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science, Amsterdam, 1999.

[15] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16:1004–1022, 1987.

[16] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13:338–355, 1984.

[17] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete Comput. Geom.*, 7:13–28, 1992.

[18] C. Levcopoulos, G. Narasimhan, and M. Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 186–195, 1998.

[19] Linial, London, and Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15, 1995.

[20] G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean paths, cycles and trees. Report 9, Department of Computer Science, University of Magdeburg, Magdeburg, Germany, 1999.

[21] D. Peleg and A. Schäffer. Graph spanners. *J. Graph Theory*, 13:99–116, 1989.

[22] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, Berlin, 1988.

[23] S. Rao and W. D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 540–550, 1998.

[24] M. Smid. Closest-point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science, Amsterdam, 1999.