Approximating the stretch factor of Euclidean paths, cycles and trees

Giri Narasimhan* Michiel Smid†

April 23, 1999

Abstract

Given a set S of n points in $\mathbb{R}^d$, and a graph G having the points of S as its vertices, the stretch factor t^* of G is defined as the maximal value $|pq|_G/|pq|$, where $p, q \in S$, $p \neq q$, $|pq|_G$ is the length of a shortest path in G between p and q, and $|pq|$ is the Euclidean distance between p and q. We consider the problem of designing algorithms that, for an arbitrary constant $\epsilon > 0$, compute an ϵ-approximation to this stretch factor, i.e., a value t such that $t \leq t^* \leq (1+\epsilon)t$. We give efficient solutions for the cases when G is a path, cycle, or tree. The main idea used in all the algorithms is to use well-separated pair decompositions to speed up the computations.

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. 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

*Department of Mathematical Sciences, The University of Memphis, Memphis TN 38152. E-mail: giri@msci.memphis.edu. Research supported by NSF Grant CCR-940-9752, and a grant by Cadence Design Systems.

†Department of Computer Science, University of Magdeburg, D-39106 Magdeburg, Germany. E-mail: michiel@isg.cs.uni-magdeburg.de.

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 constant. We say that G is a *t-spanner* for S, if for each pair of points $p, q \in S$, $p \neq q$, 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. We will denote the stretch factor by t^*. Note that

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

Let $\epsilon > 0$. We call t an *ϵ-approximate stretch factor* if $t \leq t^* \leq (1+\epsilon)t$.

Let G be a planar graph with n vertices. Frederickson [11] has shown that the distances in G between all pairs p and q of vertices can be computed in $O(n^2)$ total time. Therefore, the *exact* stretch factor of a planar graph can be computed in $O(n^2)$ time. In general, the time complexity of solving the *All-Pairs-Shortest-Path* problem is an upper bound on the time complexity of computing the exact stretch factor of a graph. We are not aware of any algorithms that compute the exact stretch factor in subquadratic time, for any class of connected Euclidean graphs.

Spanners have applications in network design, robotics, distributed algorithms, and many other areas, and have been the subject of considerable research [1, 3, 7, 8, 14]. 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 problems such as the traveling salesperson problem [2, 15]. Most of the earlier research considered the problem of constructing or analyzing geometric t-spanners. In this paper, we consider the interesting dual problem, i.e., the problem of finding the value of t for which a given geometric graph is a t-spanner. There has been some research in this general direction. Some interesting papers in this direction include papers by Dobkin et al. [9], and Keil and Gutwin [13], who showed that the Delaunay triangulation has a stretch factor bounded by a small constant, and a paper by Eppstein [10] who showed that a certain class of geometric graphs (called beta-skeletons) can have arbitrarily large stretch factors. The current paper represents the first attempt at devising algorithms to efficiently compute the stretch factors for larger classes of graphs.

If the graph represents, say, a network of highways, then the stretch factor (also referred to as *dilation* or *distortion* in the literature) 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. Our algorithm also determines the two vertices for which this increase is (approximately) maximized. In this sense, our algorithm helps to identify the "weakest" part of the network in terms of distances.

The techniques used in this paper are particularly interesting. Here we present an elegant way to use the *well-separated pair decomposition* devised by Callahan and Kosaraju [4, 6], thus adding to the list of applications of this powerful method. For other applications of the decomposition, see [3, 5, 6]. Although we have not been successful so far, we believe that our techniques can be generalized to solve the same problem for more general classes of graphs. We also believe that these techniques can be used to solve related problems such as approximating the stretch factors for arbitrary pairs of vertices, and efficiently computing approximate shortest path lengths between arbitrary pair of vertices.

The results of this paper are as follows. Let S be a set of n points in $\mathbb{R}^d$, let G be a connected graph on the points of S, and let $\epsilon > 0$ be a constant. We show that an ϵ-approximate stretch factor of G can be computed

1. in $O(n \log n)$ time, if G is a path,
2. in $O(n \log n)$ time, if G is a cycle,
3. in $O(n \log^2 n)$ time, if G is a tree.

2 Well-separated pairs

Our algorithms use the *well-separated pair decomposition* devised by Callahan and Kosaraju [4, 6].

Definition 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 the points of A, (ii) C_B contains 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 1 for an illustration. In this paper, s will always be a constant, called the *separation constant.*

The following lemma follows easily from Definition 1.

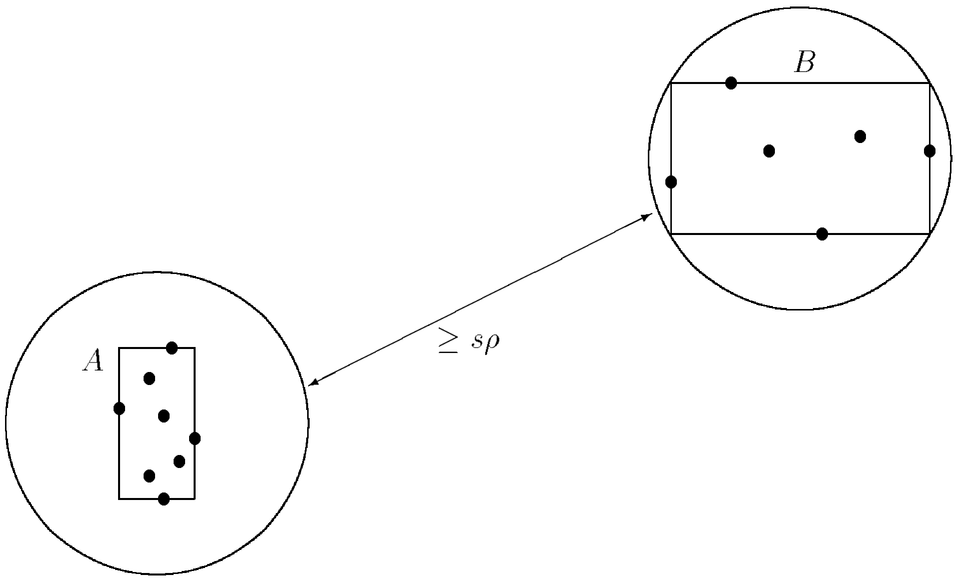

Figure 1: *Two planar point sets A and B that are well-separated w.r.t. s. Both circles have radius ρ; their distance is at least $s\rho$.*

Lemma 1 *Let A and B be two sets of points that are well-separated w.r.t. s, let a and x be points of A, and let b and y be points of B. Then*

1. $|ab| \leq (1 + 4/s)|xy|$, *and*
2. $|ab| \leq (1 + 2/s)|ay|$.

Definition 2 ([4, 6]) *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,*

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

such that

1. $A_i \cap B_i = \emptyset$, *for all $i = 1, 2, \ldots, m$,*
2. *for each unordered pair $\{p, q\}$ of distinct points 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* $i = 1, 2, \ldots, m$.

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

Callahan and Kosaraju show how such a WSPD of size $m = O(n)$ can be computed using a binary tree T, called the *split tree*. We briefly describe the main ideas. The split tree is similar to a kd-tree. They start by computing the bounding box of the points of S, which is successively split by d-dimensional hyperplanes, each of which is orthogonal to one of the axes. If a box is split, they take care that each of the two resulting boxes contains at least one point of S. As soon as a box contains exactly one point, the process stops (for this box).

The resulting binary tree T stores the points of S at its leaves; one leaf per point. Also, each node of T is associated with a subset of S. We denote this subset by S_u; it is the set of all points of S that are stored in the subtree of u.

The split tree T can be computed in $O(n \log n)$ time. Callahan and Kosaraju show that, given T, a WSPD of size $m = O(n)$ can be computed in $O(n)$ time. Each pair $\{A_i, B_i\}$ in this WSPD is represented by two nodes u_i and v_i of T. That is, we have $A_i = S_{u_i}$ and $B_i = S_{v_i}$.

In [4], Callahan also showed that, given the split tree T, a WSPD of size $m = O(n \log n)$ can be computed, such that each pair $\{A_i, B_i\}$ contains at least one singleton set. Again, each pair is represented by two nodes of T. The running time of this algorithm is bounded by $O(n \log n)$.

Theorem 1 ([4, 6]) *Let* S *be a set of* n *points in* $\mathbb{R}^d$, *and* $s > 0$ *a separation constant.*

1. *In* $O(n \log n)$ *time, we can compute a WSPD for* S *of size* $O(n)$.

2. *In* $O(n \log n)$ *time, we can compute a WSPD for* S *of size* $O(n \log n)$, *in which each pair* $\{A_i, B_i\}$ *contains at least one singleton set.*

General Algorithm $\mathcal{A}$

Step 1: Using separation constant $s = 4/\epsilon$, compute a split tree for S and compute a corresponding WSPD

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

Step 2: For each i, $1 \leq i \leq m$, compute two points a_i and b_i, where $a_i \in A_i$ and $b_i \in B_i$, such that

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

Let $t_i := |a_i b_i|_G / |a_i b_i|$.

Step 3: Report the value of t, defined as $t := \max(t_1, t_2, \ldots, t_m)$.

Figure 2: *The general algorithm for approximating the stretch factor.*

3 The general approach

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 let ϵ be a positive constant. The algorithms in this paper are based on the general algorithm $\mathcal{A}$ described in Figure 2. The following lemma proves the correctness of this algorithm.

Lemma 2 *The value of t reported by algorithm $\mathcal{A}$ is an ϵ-approximate stretch factor of G.*

Proof. Let t^* be the stretch factor of the graph G. We will 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 1, we have $|a_i b_i| \leq (1 + 4/s)|xy| = (1+\epsilon)|xy|$. This gives

$$t^* = \frac{|xy|_G}{|xy|} \leq \frac{|a_i b_i|_G}{|xy|}$$

$$
\begin{aligned}
&\leq (1+\epsilon)\frac{|a_i b_i|_G}{|a_i b_i|} = (1+\epsilon)t_i \\
&\leq (1+\epsilon)t.
\end{aligned}
$$

This completes the proof. ∎

Remark 1 If the WSPD has the property that each pair $\{A_i, B_i\}$ contains at least one singleton set, then we can take the separation constant $s = 2/\epsilon$. See Lemma 1.

4 Approximating the stretch factor of a simple path

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, 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 $s = 4/\epsilon$. By Theorem 1, we can compute such a WSPD with $m = O(n)$, in $O(n \log n)$ time.

Recall that each pair $\{A_i, B_i\}$ is represented by two nodes u_i and v_i in the split tree T such that their subtrees contain exactly the points of A_i and B_i in their leaves, respectively.

By a simple post order traversal of T, we store with each node u of T the smallest and largest indices of all points that are stored at the leaves of the subtree of u.

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 $|p_0 p_j|_G$. Using this information, we can compute for any $0 \leq j < k < n$, the distance $|p_j p_k|_G$ in constant time, as the difference between $|p_0 p_k|_G$ and $|p_0 p_j|_G$.

Then, for each i, $1 \leq i \leq m$, consider the nodes u_i and v_i. Let j and j' be the smallest and largest indices that are stored with u_i. Similarly, let k and k' be the smallest and largest indices that are stored with v_i. We compute $|p_j p_{k'}|_G$ and $|p_{j'} p_k|_G$. Finally, we compute

$$t_i := \max\left\{\frac{|p_j p_{k'}|_G}{|p_j p_{k'}|}, \frac{|p_{j'} p_k|_G}{|p_{j'} p_k|}\right\}.$$

It is easy to see that this correctly implements Step 2. Clearly, the running time of the entire algorithm is bounded by $O(n \log n)$.

Theorem 2 *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 $O(n \log n)$ time, we can compute an ϵ-approximate stretch factor of G.*

5 Approximating the stretch factor of a cycle

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

We preprocess G in $O(n)$ time, such that the distance in G between any two points of S can be computed in $O(1)$ time.

We compute 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 $s = 2/\epsilon$, such that each set A_i is a singleton, say $A_i = \{a_i\}$. Note that $m = O(n \log n)$. With each point p_j, we store all indices i, $1 \leq i \leq m$, such that $a_i = p_j$.

Let L be the total length of the cycle, i.e., $L = \sum_{j=0}^{n-1} |p_j p_{j+1}|$, where indices are to be read modulo n.

By walking around the cycle, we compute for each j, $0 \leq j < n$, the index k such that

$$|p_j p_{j+1}| + |p_{j+1} p_{j+2}| + \ldots + |p_{k-2} p_{k-1}| < L/2,$$

and

$$|p_j p_{j+1}| + |p_{j+1} p_{j+2}| + \ldots + |p_{k-1} p_k| \geq L/2.$$

Then define $h_j := p_k$.

Let $1 \leq i \leq m$, and consider the point a_i. Let j be the index such that $a_i = p_j$. Let b_i be the point in B_i such that

$$|a_i b_i|_G = \max\{|a_i q|_G : q \in B_i\}.$$

Let x be the first point of B_i that is encountered when walking along G from h_j towards p_j in one direction. Similarly, let y be the first point of B_i that is encountered when walking along G from h_j towards p_j in the other direction. It is easy to see that $b_i \in \{x, y\}$.

Hence, we can solve the problem of computing an ϵ-approximate stretch of G, if we can solve m queries of the form: given a point h and an index i, find the first point of B_i that is encountered when walking from h along the cycle in a given direction.

We will show how a batch of m such queries can be solved in $O(m)$ total time. Consider the split tree T. Recall that each set B_i is represented by a node, say v_i, of T, i.e., $B_i = S_{v_i}$.

The algorithm that solves the batch of queries traverses the nodes of T in postorder. If u is the node of T that is currently visited, then the algorithm has a data structure $DS(u)$ that stores all points of S that are in the subtree of u. This data structure supports the following two operations. First, we can insert a point into $DS(u)$. Second, we can answer queries of the form: given a point h of S and a direction ($+$ or $-$), find the first point of the set S_u that is encountered when walking from h along the cycle G in the given direction. This data structure will be specified below. The algorithm is given below.

TRAVERSE(u, T): (* u is a node of T *)
if u is a leaf
then build a data structure $DS(u)$ storing the index
    of the point stored at u;
    $x :=$ point stored at u;
    for each i such that $B_i = \{x\}$
    do $t_i := |a_i x|_G / |a_i x|$
    endfor;
else $u' :=$ left child of u;
  $u'' :=$ right child of u;
  TRAVERSE(u', T);
  TRAVERSE(u'', T);
  if $|S_{u'}| \leq |S_{u''}|$
  then insert each element of $DS(u')$ into $DS(u'')$;
      $DS(u) := DS(u'')$;
      discard $DS(u')$
  else insert each element of $DS(u'')$ into $DS(u')$;
      $DS(u) := DS(u')$;
      discard $DS(u'')$
  endif;
  for each i such that $B_i = S_u$

do $p_j := a_i$
$x :=$ point returned by querying $DS(u)$ with the point h_j and direction $+$;
$y :=$ point returned by querying $DS(u)$ with the point h_j and direction $-$;
if $|a_i x|_G > |a_i y|_G$
then $t_i := |a_i x|_G / |a_i x|$
else $t_i := |a_i y|_G / |a_i y|$
endif
endfor
endif

How do we implement the data structure DS? This structure stores a subset of the points of S. Clearly, it suffices to store the indices of these points. These indices define a set of pairwise disjoint intervals whose union is $[0 \ldots n-1]$. An insertion can be viewed as splitting such an interval into two disjoint intervals. In a query, we basically want to find the endpoints of the interval that contains the query point. Hence, we have to solve the *interval split-find problem.* Imai and Asano [12] have shown that this problem can be solved in $O(1)$ amortized time per operation. Since the number of queries is $m = O(n \log n)$, we spend $O(n \log n)$ time for solving them all. How many insertions are performed? Note that $DS(u)$ is obtained by inserting the points from the child's structure whose subtree is smaller, into the structure of the other child. The following lemma shows that each point of S is inserted at most $\log n$ times. Since there are n points, the total number of insertions is therefore bounded by $O(n \log n)$. This will prove that the total time for the insertions is $O(n \log n)$.

Lemma 3 *Let T be a binary tree with n leaves, and let ℓ be an arbitrary leaf of T. Let k be the number of proper ancestors u of ℓ such that the number of leaves in the subtree that contains ℓ and that is rooted at a child of u is less than or equal to the number of leaves in the subtree that is rooted at the other child of u. Then $k \leq \log n$.*

Proof. Let $u_1, u_2, \ldots, u_k$ be the nodes that satisfy the hypothesis, sorted along the path from the root to ℓ. (Hence, u_1 is closest to the root.) For each i, $1 \leq i \leq k$, let m_i be the number of leaves in the subtree that contains ℓ and that is rooted at a child of u_i.

Clearly, $m_1 \leq n/2$, and $m_k \geq 1$. Let i be such that $1 < i \leq k$. Since (i) the number of leaves in the subtree of u_i is at least $2m_i$, and (ii) u_i is in the subtree that contains ℓ and that is rooted at a child of u_{i-1}, we have $m_{i-1} \geq 2m_i$. Therefore,

$$n/2 \geq m_1 \geq 2m_2 \geq 2^2 m_3 \geq \ldots \geq 2^{k-1} m_k \geq 2^{k-1}.$$

This implies that $k \leq \log n$. ∎

We summarize the result of this section.

Theorem 3 *Let S be a set of n points in $\mathbb{R}^d$, let G be a cycle on the points of S, and let ϵ be a positive constant. In $O(n \log n)$ time, we can compute an ϵ-approximate stretch factor of G.*

6 Approximating the stretch factor of a tree

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 $\mathcal{A}$, computes an ϵ-approximate stretch factor of G.

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 $|pv|_G$ between p and the centroid vertex v.

Step 2: Use the same algorithm to compute an ϵ-approximate stretch factor t_1 of G_1, and an ϵ-approximate stretch factor t_2 of 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 $m = O(n)$.

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 values

$$dist_1(u) = \max\{|qv|_G : q \in S_u \cap G_1, q \neq v\}, \quad (1)$$

$$dist_2(u) = \max\{|qv|_G : q \in S_u \cap G_2, q \neq v\}. \quad (2)$$

(Here, we define the maximal element of the empty set as $-\infty$.) If $dist_1(u) \neq -\infty$, then we store with u a point $q_1 \in S_u \cap G_1$, $q_1 \neq v$, for which $dist_1(u) = |q_1 v|_G$. Similarly, if $dist_2(u) \neq -\infty$, then we store with u a point $q_2 \in S_u \cap G_2$, $q_2 \neq v$, for which $dist_2(u) = |q_2 v|_G$. In other words, each node of T also stores the vertices in $S_u \cap G_1$ and $S_u \cap G_2$ that are farthest from the centroid vertex, along with the corresponding distances; these vertices will help determine the approximate stretch factor between vertices in a well-separated pair of the decomposition.

Step 5: For each i, $1 \leq i \leq m$, we do the following. Consider the pair $\{A_i, B_i\}$ in our WSPD, and the nodes u_i and v_i in the split tree such that $A_i = S_{u_i}$ and $B_i = S_{v_i}$, respectively.

If $dist_1(u_i) = -\infty$ or $dist_2(v_i) = -\infty$, then set $t'_i := -\infty$. Otherwise, consider the point $q_1 \in A_i \cap G_1$, $q_1 \neq v$, for which $dist_1(u_i) = |q_1 v|_G$, and the point $q_2 \in B_i \cap G_2$, $q_2 \neq v$, for which $dist_2(v_i) = |q_2 v|_G$. Set $t'_i := |q_1 q_2|_G / |q_1 q_2|$.

Symmetrically, if $dist_2(u_i) = -\infty$ or $dist_1(v_i) = -\infty$, then set $t''_i := -\infty$. Otherwise, consider the point $q_1 \in B_i \cap G_1$, $q_1 \neq v$, for which $dist_1(v_i) = |q_1 v|_G$, and the point $q_2 \in A_i \cap G_2$, $q_2 \neq v$, for which $dist_2(u_i) = |q_2 v|_G$. Set $t''_i := |q_1 q_2|_G / |q_1 q_2|$.

Note that $|q_1 q_2|_G$ can be easily computed as the sum of $|q_1 v|_G$ and $|q_2 v|_G$, both of which have been computed in Step 1.

Step 6: The last step is to compute

$$t := \max(t_1, t_2, t'_1, t'_2, \ldots, t'_m, t''_1, t''_2, \ldots, t''_m).$$

Once again, the correctness of the above algorithm is proved in the following lemma.

Lemma 4 *The above algorithm computes an ϵ-approximate stretch factor of the graph G.*

Proof. The proof is by induction. The algorithm trivially computes the stretch factor (precisely) when the graph has only one vertex. Assume that the above algorithm correctly computes an ϵ-approximate stretch factor of all graphs with fewer vertices than G. Let t^* be the exact stretch factor of G. Since each of $t_1, t_2, t'_1, t'_2, \ldots, t'_m, t''_1, t''_2, \ldots, t''_m$ is either $-\infty$ or has the form $|pq|_G/|pq|$ for some points p and q, $p \neq q$, we clearly have $t \leq t^*$. It remains to show that $t^* \leq (1+\epsilon)t$.

Let x and y be two points of S, $x \neq y$, such that $t^* = |xy|_G/|xy|$.

Case 1: x and y are both vertices in G_1, or both vertices in G_2.

Assume w.l.o.g. that x and y are vertices in G_1. Let t_1^* be the exact stretch factor of the tree G_1. Then, by the inductive assumption, we have $t_1^* \leq (1+\epsilon)t_1$. Also, it is easy to see that $t_1^* = t^*$. This implies

$$t^* = t_1^* \leq (1+\epsilon)t_1 \leq (1+\epsilon)t.$$

Case 2: x and y are in different trees.

Assume w.l.o.g. that x is a vertex in G_1 and y is a vertex in G_2. Let i be the 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 nodes u_i and v_i of the split tree such that $A_i = S_{u_i}$ and $B_i = S_{v_i}$.

Since $x \in A_i \cap G_1$, we know that $dist_1(u_i) \neq -\infty$. Let $q_1 \in A_i \cap G_1$ such that $dist_1(u_i) = |q_1 v|_G$. Then $|xv|_G \leq |q_1 v|_G$.

Similarly, since $y \in B_i \cap G_2$, $dist_2(v_i) \neq -\infty$. Let $q_2 \in B_i \cap G_2$ such that $dist_2(v_i) = |q_2 v|_G$. Then $|yv|_G \leq |q_2 v|_G$.

We have

$$t^* = \frac{|xy|_G}{|xy|} = \frac{|xv|_G + |vy|_G}{|xy|} \leq \frac{|q_1 v|_G + |vq_2|_G}{|xy|} = \frac{|q_1 q_2|_G}{|xy|}.$$

By Lemma 1, we have

$$|q_1 q_2| \leq (1 + 4/s)|xy| = (1+\epsilon)|xy|.$$

It follows that

$$t^* \leq (1+\epsilon)\frac{|q_1 q_2|_G}{|q_1 q_2|} = (1+\epsilon)t'_i \leq (1+\epsilon)t.$$

This completes the proof. ∎

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

$$\mathcal{T}(n) = O(n \log n) + \mathcal{T}(n_1) + \mathcal{T}(n_2),$$

where n_1 and n_2 are positive integers such that $n_1 \leq 2n/3$, $n_2 \leq 2n/3$, and $n_1 + n_2 = n + 1$. This recurrence relation solves to $\mathcal{T}(n) = O(n \log^2 n)$.

Theorem 4 *Let S be a set of n points in $\mathbb{R}^d$, let G be a tree having the points of S as its vertices, and let $\epsilon > 0$ be a constant. In $O(n \log^2 n)$ time, we can compute an ϵ-approximate stretch factor of G.*

7 Concluding remarks

This is the first attempt to study the interesting problem of computing stretch factors of given geometric graphs. We provide efficient algorithms to solve the problem on special classes of graphs such as paths, cycles and trees. The problem of finding efficient algorithms for broader classes of graphs remains open. More specifically, the problem of efficiently determining the stretch factor of planar graphs is an interesting open problem. The main tool that we employ, namely, the method of using well-separated pair decompositions seems particularly noteworthy.

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] Sanjeev Arora, Michelangelo Grigni, David Karger, Philip Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, pages 33–41, 1998.

[3] 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.

[4] P. B. Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its applications.* Ph.D. thesis, Dept. Comput. Sci., Johns Hopkins University, Baltimore, Maryland, 1995.

[5] 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.

[6] 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.

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

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

[9] 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.

[10] David 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.

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

[12] H. Imai and Ta. Asano. Dynamic orthogonal segment intersection search. *J. Algorithms*, 8:1–18, 1987.

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

[14] 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.

[15] 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.