IMPROVING THE STRETCH FACTOR OF A GEOMETRIC NETWORK BY EDGE AUGMENTATION

MOHAMMAD FARSHI*, PANOS GIANNOPoulos†, AND JOACHIM GUDMUNDSSON‡

Abstract. Given a Euclidean graph $G$ in $\mathbb{R}^d$ with $n$ vertices and $m$ edges, we consider the problem of adding an edge to $G$ such that the stretch factor of the resulting graph is minimized. Currently, the fastest algorithm for computing the stretch factor of a graph with positive edge weights runs in $O(nm + n^2 \log n)$ time, resulting in a trivial $O(n^3 m + n^4 \log n)$ time algorithm for computing the optimal edge. First, we show that a simple modification yields the optimal solution in $O(n^2)$ space. To reduce the running time we consider several approximation algorithms.

Key words. Computational Geometry, Approximation algorithms, Geometric networks.

AMS subject classifications. 65D18, 68U05, 68Q25

1. Introduction. Consider a set $V$ of $n$ points in $\mathbb{R}^d$. A network on $V$ can be modeled as an undirected graph $G$ with vertex set $V$ of size $n$ and an edge set $E$ of size $m$ where every edge $(u, v)$ has a positive weight $w(u, v)$. A Euclidean network is a geometric network where the weight of the edge $(u, v)$ is equal to the Euclidean distance $|uv|$ between its two endpoints $u$ and $v$.

For two vertices $u, v$ in a weighted graph $G$ we use $\delta_G(u, v)$ to denote a shortest path between $u$ and $v$ in $G$ and the length of the path is denoted by $d_G(u, v)$. Consider a weighted graph $G = (V, E)$ and a graph $G' = (V, E')$ on the same vertex set but with edge set $E' \subseteq E$. We say that $G'$ is a $t$-spanner of $G$ if for each pair of vertices $u, v \in V$ we have that $d_G(u, v) \leq t \cdot d_G(u, v)$. The minimum $t$ such that $G$ is a $t$-spanner for $V$ is called the stretch factor, or dilation, of $G$.

We say that a Euclidean network $G = (V, E)$ is a $t$-spanner if $G = (V, E)$ is a $t$-spanner of the complete network on $V$. In other words, for any two points $p, q \in V$ the graph distance in $G$ is at most $t$ times the Euclidean distance between the two points.

Complete graphs represent ideal communication networks, but they are expensive to build; sparse spanners represent low-cost alternatives. The weight of the spanner network is a measure of its sparseness; other sparseness measures include the number of edges, the maximum degree, and the number of Steiner points. Spanners for complete Euclidean graphs as well as for arbitrary weighted graphs find applications in robotics, network topology design, distributed systems, design of parallel machines, and many other areas. Recently spanners found interesting practical applications in areas such as metric space searching [29, 30] and broadcasting in communication networks [2, 14, 25].

Several well-known theoretical results also use the construction of $t$-spanners as a building block, for example, Rao and Smith [32] made a breakthrough by showing an optimal $O(n \log n)$-time approximation scheme for the well-known Euclidean traveling salesperson problem, using $t$-spanners (or banyans). Similarly, Czumaj and Lingas [7] showed approximation schemes for minimum-cost multi-connectivity problems in geometric graphs. The problem of constructing geometric spanners has received considerable attention from a theoretical perspective, see [1, 3, 4, 5, 8, 9, 10, 17, 20, 21, 23, 24, 33, 36], the surveys [12, 16, 34] and the book by Narasimhan and Smid [28]. Note that considerable research has also been done in the construction of spanners for general graphs, see for example, the book by Peleg [31] or the recent work by Elkin and Peleg [11] and Thorup and Zwick [35].

All the existing algorithms construct a network from scratch but in many applications the network is already given, and the problem at hand is to extend the network with an additional edge, or edges, while minimizing the stretch factor of the resulting graph. The problem was first stated by Narasimhan [26] and,
surprisingly, it has not been studied earlier, to the best of the authors’ knowledge. In this paper we study
the following problem:

**Problem.** Given a graph \( G \), construct a graph \( G' \) by adding an edge to \( G \) such that the stretch factor of \( G' \)
is minimized.

The results presented in this paper are summarized in Table 1. Note that some of the presented bounds
hold for any graph with positive edge weights (weighted graphs) while some only hold for Euclidean graphs.
Finally, throughout this paper we will use \( G_P \) to denote the optimal solution, while \( t_P \) and \( t \) denote the
stretch factor of \( G_P \) and the input graph \( G \) respectively.

<table>
<thead>
<tr>
<th>Input graph</th>
<th>Approximation factor</th>
<th>Time complexity</th>
<th>Space</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weighted graph</td>
<td>1</td>
<td>( O(n^3m + n^4 \log n) )</td>
<td>( O(m) )</td>
<td>2.1</td>
</tr>
<tr>
<td>Weighted graph</td>
<td>1</td>
<td>( O(n^4) )</td>
<td>( O(n^2) )</td>
<td>2.1</td>
</tr>
<tr>
<td>Euclidean graph</td>
<td>( 1 + \varepsilon )</td>
<td>( O(n^3/\varepsilon^d) )</td>
<td>( O(n^2) )</td>
<td>2.2</td>
</tr>
<tr>
<td>Weighted graph</td>
<td>3</td>
<td>( O(nm + n^2 \log n) )</td>
<td>( O(m) )</td>
<td>3</td>
</tr>
<tr>
<td>Euclidean graph</td>
<td>( 2 + \varepsilon )</td>
<td>( O(nm + n^2 (\log n + 1/\varepsilon^d)) )</td>
<td>( O(n^2) )</td>
<td>4</td>
</tr>
<tr>
<td>Euclidean t-spanner</td>
<td>( 1 + \varepsilon )</td>
<td>( O((t^7/\varepsilon^d)^d \cdot n^2) )</td>
<td>( O(m + (t^3/\varepsilon^d)^d \cdot n \log (tn)) )</td>
<td>5</td>
</tr>
</tbody>
</table>

**Table 1.1**

Complexity bounds for the algorithms presented in the paper.

2. Three simple algorithms. A naïve approach to decide which edge to add is to test every possible
candidate edge. The number of such edges is obviously \( \frac{n(n-1)}{2} - m = O(n^2) \). Testing a candidate edge \( e \)
entails computing the stretch factor of the graph \( G' = (V, E \cup \{e\}) \), denoted the candidate graph. Therefore
we briefly consider the problem of computing the stretch factor of a given graph with positive edge weights.
This problem has recently received considerable attention, see for example [13, 22, 27].

2.1. Exact algorithms. We consider the problem of computing an optimal solution \( G_P \). That is, we
are given a t-spanner \( G = (V, E) \), and the aim is to compute a \( t_P \)-spanner \( G_P = (V, E \cup \{e\}) \).

A trivial upper bound is obtained by computing the length of the shortest paths between every pair of
vertices in \( G' \). This can be done by running Dijkstra’s algorithm – implemented using Fibonacci heaps –
n times, resulting in an \( O(nm + n^2 \log n) \) time algorithm using linear space. This approach is quite slow
and we would like to be able to compute the stretch factor more efficiently, but no faster algorithm is
known for any graphs except planar graphs, paths, cycles, stars and trees [13, 22, 27]. Applying the stated
bound to the problem of computing the exact stretch factor of \( G' \) gives that \( G_P \) can be computed in time
\( O(n^3(m + n \log n)) \) using linear space.

A small improvement can be obtained by observing that when an edge \((u, v)\) is about to be tested, we do
not have to check all possible shortest paths between two vertices \( x, y \in V \) again, it suffices to check whether
there is a shorter path using the edge \((u, v)\). That is, we only have to compute \( d_G(x, u) + w(u, v) + d_G(v, y),
\( d_G(x, v) + w(v, u) + d_G(u, y) \) and \( d_G(x, y) \), which can be done in constant time since the length of a shortest
path between every pair of vertices in \( G \) has already been computed (provided that we store this information).
Hence, by first computing all-pair-shortest paths of \( G \) we obtain:

**Lemma 2.1.** Given a graph \( G \) with positive edge weights, an optimal solution \( G_P \) can be computed in
time \( O(n^4) \) using \( O(n^2) \) space.

**Proof.** Computing the all-pair-shortest path requires cubic time and all the distances are stored in an
\( n \times n \) matrix. The \( O(n^2) \) edges are tested for insertion, for each candidate edge compute the length of the
shortest path between every pair of points in \( G \), each of which can be done in constant time as described
above. \( \Box \)
2.2. A \((1 + \varepsilon)\)-approximation for Euclidean graphs. In the previous section we showed that an optimal solution can be obtained by testing a quadratic number of candidate edges. Testing each candidate edge entails \(O(n^2)\) distance queries, where a distance query asks for the length of a shortest path in the graph between two query points. One way to speed up the computation is to compute an approximate stretch factor. \(t'\) is said to be a \(\beta\)-approximate stretch factor of \(G\) if \(tc \leq t' \leq \beta \cdot tc\), where \(tc\) is the stretch factor of \(G\). The problem of computing an approximate stretch factor of a geometric graph was considered by Narasimhan and Smid in [27]. They showed the following fact:

**Fact 1.** (Narasimhan and Smid [27]) Given a Euclidean graph \(G\) and a real value \(\tau > 0\), a \((1 + \tau)^2\)-approximate stretch factor of \(G\) can be computed by performing \(O(n/\tau^d)\) many \((1 + \gamma)\)-approximate distance queries, where \(\gamma\) is a positive constant smaller than \(\tau\).

The algorithm is almost as stated in the previous section with the exception that when the stretch factor of the candidate graph is computed we approximate it by only performing \(O(n/\tau^d)\) shortest path queries as stated in Fact 1. As a result the time to compute the stretch factor decreases from \(O(n^2)\) to \(O(n/\tau^d)\), thus the total running time decreases from \(O(n^d)\) to \(O(n^3/\tau^d)\).

**Theorem 2.2.** Given a Euclidean graph \(G = (V, E)\) and a real constant \(\varepsilon > 0\), one can in \(O(n^3/\varepsilon^d)\) time, using \(O(n^2)\) space, compute a \(t'\)-spanner \(G' = (V, E \cup \{e\})\) such that \(t' \leq (1 + \varepsilon) \cdot t_p\).

**Proof.** The time bound follows from the above discussion setting \(\tau = \sqrt{1 + \varepsilon} - 1\), where \(\tau\) is as stated in Fact 1. It remains to prove that \(G'\) has stretch factor \(((1 + \varepsilon) \cdot t_p)\).

For each candidate graph \(G'_i\), let \(t'_i\) be its approximate stretch factor as computed by the algorithm and let \(t_i\) be its exact stretch factor. From Fact 1 it follows that for each candidate graph \(G'_i\), \(t'_i \leq (1 + \tau)^2 \cdot t_i\). Assume that \(t_P = t_j\) and that \(t' = t'_k\) for some indices \(j\) and \(k\). As a result we have:

\[
t' = t'_k \leq t'_j \leq (1 + \tau)^2 \cdot t_j = (1 + \tau)^2 \cdot t_P = (1 + \varepsilon) \cdot t_P \quad \text{and} \quad t_P \leq t_k \leq t'_k = t'.
\]

Thus, \(t_P \leq t' \leq (1 + \varepsilon) \cdot t_P\).

3. Adding a bottleneck edge. Consider a graph \(G = (V, E)\) with positive edge weights and stretch factor \(t\). In this section we analyze the following simple algorithm: Add an edge between a pair of vertices in \(G\) with stretch factor \(t\), this edge is called a bottleneck edge of \(G\).

Let \(G_B\) be a graph obtained from \(G\) by adding a bottleneck edge, and let \(t_B\) be the stretch factor of \(G_B\). Note that \(G_B\) can be computed in the same time as the stretch factor of \(G\) can be decided, i.e., in \(O(mn + n^2 \log n)\) time for graphs with positive edge weights.

![Fig. 3.1. (x, y) is the optimal edge added to G and (u, v) is a bottleneck edge.](image)

**Lemma 3.1.** Given a graph \(G\) with positive edge weights it holds that \(t_B < 3t_P\).

**Proof.** Recall that \(t\) denotes the stretch factor of \(G\) and that \(G_P\) denotes the optimal graph. Let \((x, y)\) be the edge added to \(G\) to obtain \(G_P\), and let \((u, v)\) be the edge added to \(G\) to obtain \(G_B\), i.e., \((u, v)\) is a bottleneck edge of \(G\), as illustrated in Fig. 3.1.

First note that if \(t_P > t/3\) then the lemma holds and we are done. Thus we may assume that \(t_P \leq t/3\). The proof of the lemma is done by considering a pair of vertices, denoted \((a, b)\), that are endpoints of a bottleneck edge of \(G_B\). Fix a path \(d_{G_P}(a, b)\). If this path does not include the edge \((x, y)\) then \(d_{G_P}(a, b) =\)
\(d_G(a, b) \geq d_{G^\delta}(a, b)\) and we are done. Therefore, we may assume that the path \(\delta_{G^\delta}(a, b)\) includes \((x, y)\). Also, we will assume without loss of generality that a shortest path in \(G_P\) from \(a\) to \(b\) goes from \(a\) to \(x\) and then to \(b\) via \(y\), otherwise the labels \(a\) and \(b\) may be switched. Note that \(\delta_{G^\delta}(u, v)\) must pass through \((x, y)\) otherwise we have \(t_P \geq d_{G^\delta}(u, v)/|uv| = d_G(u, v)/|uv| = t\) which means that \(t = t_P\), which contradicts the assumption that \(t_P \leq t/3\). Furthermore, we assume that a shortest path in \(G_P\) from \(u\) to \(v\) goes from \(u\) to \(x\) and then to \(v\) via \(y\), otherwise the labels \(u\) and \(v\) may be switched.

As a first step we bound the distance between the endpoints of the bottleneck edge \(u\) and \(v\). This is done by bounding the length of the path in \(G\) between \(x\) and \(y\) as follows, see Fig. 3.1.

\[
d_G(u, v) \leq d_{G^\delta}(u, v) - |xy| + d_G(x, y) \\
\leq t_P \cdot |uv| - |xy| + t \cdot |xy| \\
\leq \frac{t}{3} \cdot |uv| - |xy| + t \cdot |xy| \\
< \frac{t}{3} \cdot |uv| + t \cdot |xy|.
\]

Since \(d_G(u, v) = t \cdot |uv|\) it follows that

\[
|uv| < \frac{3}{2} \cdot |xy|.
\] (3.1)

Also,

\[
t \cdot |uv| = d_G(u, v) \\
\leq d_G(u, a) + d_G(a, b) + d_G(b, v) \\
\leq d_G(u, a) + t \cdot |ab| + d_G(b, v)
\]

which implies that

\[
t \cdot (|uv| - |ab|) \leq d_G(u, a) + d_G(b, v), \tag{3.2}
\]

and

\[
d_G(a, u) + 2|xy| + d_G(v, b) \leq d_G(a, x) + d_G(x, u) + 2|xy| + d_G(v, y) + d_G(y, b) \\
= d_{G^\delta}(a, b) + d_{G^\delta}(u, v) \\
\leq t_P(|ab| + |uv|), \tag{3.3}
\]

which gives that

\[
d_G(a, u) + d_G(v, b) \leq t_P(|ab| + |uv|) - 2|xy|. \tag{3.4}
\]

By putting together (3.2) and (3.4) we have

\[
t(|uv| - |ab|) \leq d_G(a, u) + d_G(v, b) \\
\leq t_P(|ab| + |uv|) - 2|xy| \\
< t_P(|ab| + |uv|),
\]

which implies that

\[
|ab|(t_P + t) > |uv|(t - t_P)
\]

and

\[
|ab| > \frac{t - t_P}{t_P + t} \cdot |uv| > \frac{t - \frac{1}{4} - \frac{1}{3}}{\frac{3}{4} + \frac{1}{3}} \cdot |uv| = \frac{1}{2} \cdot |uv|. \tag{3.5}
\]
Now we are ready to put together the results:

\[ t_B \cdot |ab| = d_{G_B}(a,b) \]
\[ \leq d_G(a,u) + |uv| + d_G(v,b) \]
\[ < d_G(a,u) + \frac{3}{2} |xy| + d_G(v,b) \] (from (3.1))
\[ < d_G(a,u) + 2 |xy| + d_G(v,b) \]
\[ \leq t_P (|ab| + |vw|) \] (from (3.3))
\[ < 3t_P \cdot |ab| \] (from (3.5))

This completes the proof of the lemma since \( t_B < 3t_P \). \( \square \)

We conclude by stating the main result of this section followed by a lower bound for the bottleneck approach.

**Theorem 3.2.** Given a graph \( G = (V,E) \) with positive edge weights, a \( t_B \)-spanner \( G' = (V,E \cup \{e\}) \) with \( t_B < 3t_P \) can be computed in \( O(mn + n^2 \log n) \) time using \( O(m) \) space.

**Observation 1.** There exists a Euclidean graph \( G \) such that \((2 - \varepsilon) \cdot t_P \leq t_B\), for any \( 0 < \varepsilon < 1 \).

**Proof.** Consider the graph \( G \), as in Fig. 3.2a. More precisely, \( G \) is a graph with ten vertices \( p_i = ((i-1) \mod 5, ((i-1)/5) \cdot \delta) \), \( 1 \leq i \leq 10 \), and nine edges \((p_5, p_{10})\) and \((p_j, p_{j+1})\), for \( 1 \leq j \leq 4 \) and \( 6 \leq j \leq 9 \). For any value \( \delta \leq 1 \): \((p_1, p_6)\) is a bottleneck edge in \( G \) and \( t_B = \frac{4 \sqrt{3}}{5} \), see Fig. 3.2b.

In the case where edge \((p_2, p_7)\) is added to \( G \), as shown in Fig. 3.2c, the resulting graph has stretch factor \((2 + \delta)/\delta\). Combining the upper and lower bounds gives \( \frac{15\delta}{t_P} \geq \frac{4 \sqrt{3}}{2 \sqrt{5}} = (2 - \varepsilon) \), where the last equality follows if we set \( \delta = \min\{1, \frac{2 \sqrt{5}}{4 \sqrt{3}}\} \).

Grüne [15] improved the lower bound in Observation 1 to \((3 - \varepsilon)\), so the upper bound stated in Lemma 3.1 is tight.

**Fig. 3.2.** (a) The input graph \( G \), (b) the graph \( G_B \), and (c) the graph \( G_P \).

### 4. A \((2 + \varepsilon)\)-approximation for Euclidean graphs.

In the remaining of the paper we will develop approximation algorithms for Euclidean graphs. In this section we present a fast approximation algorithm which guarantees an approximation factor of \((2 + \varepsilon)\). The algorithm is similar to the algorithms presented in Section 2 in the sense that it tests candidate edges. Testing a candidate edge entails computing the stretch factor of the input graph augmented with the candidate edge. The main difference is that we will show, in Section 4.2, that only a linear number of candidate edges need to be tested to obtain a solution that gives a \((2 + \varepsilon)\)-approximation, instead of a quadratic number of edges.

Moreover, in Section 4.3 we show that the same approximation bound can be achieved by performing only a linear number of shortest path queries for each candidate edge. The candidate edges are selected by using the well-separated pair decomposition, which we briefly define below.

#### 4.1. Well-separated pair decomposition.

Our algorithm uses the well-separated pair decomposition defined by Callahan and Kosaraju [6]. We briefly review this decomposition before we state the algorithms.

**Definition 4.1** ([6]). 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 with respect to \( s \), if there are two disjoint \( d \)-dimensional balls \( C_A \)
and $C_B$, having the same radius, such that (i) $C_A$ contains the bounding box $R(A)$ of $A$, (ii) $C_B$ contains the bounding box $R(B)$ of $B$, and (iii) the minimum distance between $C_A$ and $C_B$ is at least $s$ times the radius of $C_A$.

The parameter $s$ will be referred to as the separation constant. The next lemma follows easily from Definition 4.1.

**Lemma 4.2 ([6]).** Let $A$ and $B$ be two finite sets of points that are well-separated w.r.t. $s$, let $x$ and $y$ be points of $A$, and let $y$ and $q$ be points of $B$. Then (i) $|xy| \leq (1 + 4/s) \cdot |pq|$, and (ii) $|px| \leq (2/s) \cdot |pq|$.

**Definition 4.3 ([6]).** Let $S$ be a set of $n$ points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A well-separated pair decomposition (WSPD) for $S$ with respect to $s$ is a sequence of pairs of non-empty subsets of $S$, $(A_1, B_1), \ldots, (A_m, B_m)$, such that

1. $A_i \cap B_i = \emptyset$, for all $i = 1, \ldots, m$,
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 (i) $p \in A_i$ and $q \in B_i$, or (ii) $q \in A_i$ and $p \in B_i$,
3. $A_i$ and $B_i$ are well-separated w.r.t. $s$, for $1 \leq i \leq m$.

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

Callahan and Kosaraju showed that a WSPD of size $m = O(s^d n)$ can be computed in $O(s^d n + n \log n)$ time.

**4.2. Linear number of candidate edges.** In this section we show how to obtain a $(2+\varepsilon)$-approximation in cubic time. As mentioned above, the algorithm is similar to the algorithm presented in Section 2 in the sense that it tests candidate edges. Here we will show that only a linear number of candidate edges is needed to be tested to obtain a solution that gives a $(2 + \varepsilon)$-approximation.

The approach is straightforward. First the algorithm computes the length of the shortest path in $G$ between every pair of points in $V$. The distances are saved in a matrix $M$. Next, the well-separated pair decomposition is computed. Note that, in Step 5, the candidate edges will be chosen using the well-separated pair decomposition. In Step 6, the function StretchFactor returns the stretch factor of the graph on $V$ with edge set $E \cup \{(a_i, b_i)\}$, i.e., in steps 5–8, a candidate edge is tested by computing the stretch factor of $G$ with the candidate edge $(a_i, b_i)$ added to $G$.

**Algorithm ExpandGraph($G, \varepsilon$)**

**Input:** Euclidean graph $G = (V, E)$ and a real constant $\varepsilon > 0$.

**Output:** Euclidean graph $G' = (V, E' \cup \{e\})$.

1. $M \leftarrow$ All-Pairs-Shortest-Path distance matrix of $G$.
2. $\{(A_i, B_i)\}_{i=1}^k \leftarrow$ WSPD of the set $V$ with respect to separation constant $s = \frac{256}{\varepsilon^2}$.
3. $t' \leftarrow \infty$.
4. for $i \leftarrow 1$ to $k$
5. Select arbitrary points $a_i \in A_i$ and $b_i \in B_i$.
6. $t_i \leftarrow$ StretchFactor($a_i, b_i, M$).
7. if $t_i < t'$
8. then $t' \leftarrow t_i$ and $e \leftarrow (a_i, b_i)$
9. return $G' = (V, E' \cup \{e\})$.

Next, we bound the running time of the approximation algorithm and then prove the approximation bound.

**Lemma 4.4.** Algorithm ExpandGraph requires $O(n^3/\varepsilon^{2d})$ time and $O(n^2)$ space.

**Proof.** The complexity of all steps of the algorithm, except step 6, is straightforward to calculate. Recall that step 1 requires $O(nm + n^2 \log n)$ time and quadratic space, and step 2 requires $O(n/\varepsilon^{2d} + n \log n)$ time according to Section 4.1. Thus, it remains to consider step 6 of the algorithm. Note that the number of times step 6 is executed is $O(n/\varepsilon^{2d})$.

Let $G_1 = (V, E \cup \{(a_i, b_i)\})$. Since we computed the all-pair shortest distances of $G$, and stored the results in a matrix $M$ it holds that shortest path distance queries in $G_1$ can be computed in constant time.
That is, for a query \((p,q)\) return \(\min\{M[p,q], M[p,a_i] + |a_ib_i| + M[b_i,q], M[p,b_i] + |b_i a_i| + M[a_i,q]\}\). For each candidate edge, a quadratic number of queries are performed, thus summing up we get \(O(n^2 \cdot \varepsilon^2)\), as stated in the lemma. \(\square\)

It remains to analyze the quality of the solution obtained from algorithm \textsc{ExpandGraph}. We need to compare the graph resulting from adding an optimal edge to \(G\) and the graph \(G'\) resulting from \textsc{ExpandGraph}. Let \(e = (a,b)\) be an optimal edge and let \((A_i, B_i)\) be the well-separated pair such that \(a \in A_i\) and \(b \in B_i\). At first sight, it seems that the edge \((a,b)\) tested by the algorithm should be a good candidate. However, the separation constant of our well-separated pair decomposition only depends on \(\varepsilon\) which implies that the shortest path between \(a\) and \(a_i\), and between \(b\) and \(b_i\) could be very long compared to the distance between \(a\) and \(b\). In Lemma 4.5, we show the existence of a “short” edge \(e'\) that is a good approximation of the optimal edge and then, in Lemma 4.6, we show that \textsc{ExpandGraph} computes a good approximation of \(e'\).

Let \(\Delta(p,q)\) denote the set of point pairs in \(V\) such that the point pair \((u,v)\) belongs to \(\Delta(p,q)\) if and only if \((p,q) \in \delta_{G\cup \{(p,q)\}}(u,v)\). That is, \(\Delta(p,q)\) is the set of point pairs for which a shortest path between them in \(G \cup \{(p,q)\}\) passes through \((p,q)\).

![Fig. 4.1. (a) Illustrating the proof of Lemma 4.5. (b) Illustrating the proof of Lemma 4.6.](image)

**Lemma 4.5.** For any given constant \(0 < \lambda \leq 1\), there exists a point pair \(p,q \in V\) such that:

1. \(|uv| \geq \frac{\lambda}{3} |pq|\) for every pair \((u,v)\) \(\in \Delta(p,q)\), and
2. the stretch factor of \(G \cup \{(p,q)\}\) is bounded by \((2 + \lambda) \cdot t_P\).

**Proof.** The proof is done in two steps. First a point pair \(p_j, q_j \in V\) is selected that fulfills (I). Then, we prove that this pair will also fulfill (II), i.e., the stretch factor of \(G \cup \{(p_j, q_j)\}\) is bounded by \((2 + \lambda) \cdot t_P\).

Consider an optimal solution \(G_1 = G \cup \{(p_1, q_1)\}\). If \((p_1, q_1)\) fulfills (I) then we are done, i.e., we have found the point pair \((p = p_1, q = q_1)\) we are searching for. Otherwise, let \(e_2 = (p_2, q_2)\) denote the closest pair in \(\Delta(p_1, q_1)\). Since there exists a pair \((u,v)\) \(\in \Delta(p_1, q_1)\) such that \(|uv| < \frac{\lambda}{3} |p_1q_1|\) and since \((p_2, q_2)\) is the closest pair in \(\Delta(p_1, q_1)\) we have \(|p_2q_2| < \frac{\lambda}{3} |p_1q_1|\), as illustrated in Fig. 4.1a.

If \((p_2, q_2)\) fulfills (I) then \((p = p_2, q = q_2)\) and we are done. Otherwise, let \(e_3 = (p_3, q_3)\) denote the closest pair in \(\Delta(p_2, q_2)\). We continue this procedure until we find a point pair \((p_j, q_j)\) that satisfies (I). Since, for each \(i > 0\) \(|p_{i+1}q_{i+1}| < \frac{\lambda}{3} |p_iq_i|\), the process must terminate.

Now for each \(1 \leq i \leq j\), let \(G_i = G \cup \{(p_i, q_i)\}\) where \((p_i, q_i)\) are the point pairs constructed above. We claim that \(G_j\) has stretch factor at most \((2 + \lambda) \cdot t_P\). Before we continue we need to prove:

\[
d_{G_i}(p_{i+1}, q_{i+1}) \leq t_P \cdot |p_{i+1}q_{i+1}|. \tag{4.1}
\]

The inequality is obviously true for \(i = 1\). For \(i > 1\) it holds that \(|p_{i+1}q_{i+1}| < |p_2q_2|\) which implies that \((p_{i+1}, q_{i+1}) \notin \Delta(p_1, q_1)\) since \((p_2, q_2)\) is the closest pair in \(\Delta(p_1, q_1)\). This, in turn, implies that
Thus, $x_d$ path in $G$, such that Definition 4.3 such a well-separated pair must exist. Next, consider the candidate edge $(u, v)$. Lemma 4.5 states that if $(u, v)$, for which the resulting graph has small stretch factor. Note that algorithm ExpandGraph might not test $(p, q)$. However, in the following lemma it will be shown that algorithm ExpandGraph will test an edge $(a, b)$ that is almost as good as $(p, q)$.

**Lemma 4.6.** For any given constant $0 < \varepsilon \leq 1$ it holds that the graph $G'$ returned by algorithm ExpandGraph has stretch factor at most $(2 + \varepsilon) \cdot t_P$.

**Proof.** According to Lemma 4.5, there exists an edge $(p, q)$ such that for every pair $(u, v) \in \Delta(p, q)$ it holds that $|uv| \geq \frac{1}{4}|pq|$, and the stretch factor $t_H$ of $H = G \cup \{p, q\}$ is bounded by $(2 + \lambda) \cdot t_P$. Let $(A_1, B_1)$ be the well-separated pair computed in step 2 of the algorithm such that $p \in A_1$ and $q \in B_1$. According to Definition 4.3 such a well-separated pair must exist. Next, consider the candidate edge $(a_i, b_i)$ tested by the algorithm, such that $a_i, p \in A_i$ and $b_i, q \in B_i$. For simplicity of writing we will use $a$ and $b$ to denote $a_i$ and $b_i$, respectively.

Our claim is that the stretch factor $t'$ of $G' = G \cup \{(a, b)\}$ is bounded by $(1 + \varepsilon/4) \cdot t_H$. Thus setting $\lambda = t_H/4$ would then prove the lemma since $(2 + \varepsilon/4) \cdot (1 + \varepsilon/4) < (2 + \varepsilon)$, for $\varepsilon \leq 1$.

Now we are ready to prove the claim. To compute the stretch factor of $G'$ the algorithm performs a shortest path distance query between each pair of points in $V$. If it holds that $(x, y) \notin \Delta(p, q)$ for every pair of points $x, y \in V$ then the claim is obviously true, thus we only have to consider the pairs $x, y$ for which it holds that $(x, y) \in \Delta(p, q)$, see Fig. 4.1b. Now the claim is:

$$d_G(p_{i+1}, q_{i+1}) = d_G(p_{i+1}, q_{i+1}) \leq t_P \cdot |p_{i+1}, q_{i+1}|.$$  

Since $G$ is a subgraph of $G$, the length of the shortest path in $G$, between $p_{i+1}$ and $q_{i+1}$ must be bounded by the length of the shortest path in $G$ between $p_{i+1}$ and $q_{i+1}$, which is bounded by $t_P \cdot |p_{i+1}, q_{i+1}|$. Thus, inequality (4.1) holds.

We continue with the second part of the proof. If $(u, v) \notin \Delta(p, q)$ then we are done since $d_G(u, v) \leq d_G(u, v) = d_G(u, v)$. Otherwise, if $(u, v) \in \Delta(p, q)$, the following holds (see Fig. 4.1a for an illustration):

$$d_G(u, v) \leq d_G(u, v) - |p_{i+1}, q_{i+1}| + \cdots + (|p_{i+1}, q_{i+1}| - |p_{i+1}, q_{i+1}|) + |p_{i+1}, q_{i+1}|$$

Thus, $t_j < (2 + \lambda) \cdot t_P$ which concludes the lemma. $\square$
have:

\[ d_{G'}(x, y) \leq d_G(x, p) + d_G(p, a) + |ab| + d_G(b, q) + d_G(q, y) \]
\[ \leq d_G(x, p) + d_H(p, a) + |ab| + d_H(b, q) + d_G(q, y) \text{ (cf. (7))} \]
\[ \leq d_G(x, p) + |ab| + d_G(q, y) + t_H \cdot (|pa| + |bq|) \]
\[ \leq d_G(x, p) + (1 + 4/s) \cdot |pq| + d_G(q, y) + \frac{4t_H}{s} \cdot |pq| \text{ (Lemma 4.2)} \]
\[ \leq d_H(x, y) + \frac{8t_H}{s} \cdot |pq| \]
\[ \leq d_H(x, y) + \frac{64t_H}{\varepsilon s} \cdot |xy| \text{ (Lemma 4.5)} \]
\[ = d_H(x, y) + \frac{\varepsilon}{4} \cdot t_H \cdot |xy| \]

The stretch factor of the path in \( G' \) between \( x \) and \( y \) is:

\[ \frac{d_G(x, y)}{|xy|} \leq \frac{d_H(x, y)}{|xy|} + \frac{\varepsilon}{4}t_H \leq \left(1 + \frac{\varepsilon}{4}\right) t_H. \]

Finally, according to Lemma 4.5 and the fact that \( \lambda = \varepsilon/4 \) it holds that \( t_H \leq (2 + \varepsilon/4) \cdot t_P \). This completes the lemma since \((2 + \varepsilon/4)(1 + \varepsilon/4) < (2 + \varepsilon)\). \( \Box \)

We may now conclude this section with the following theorem.

**Theorem 4.7.** Given a Euclidean graph \( G = (V, E) \) in \( \mathbb{R}^d \) one can in time \( O(n^3/\varepsilon^{2d}) \), using \( O(n^2) \) space, compute a \( t' \)-spanner \( G' = (V, E \cup \{e\}) \), where \( t' \leq (2 + \varepsilon) \cdot t_P \).

### 4.3. Speeding up algorithm ExpandGraph

In the previous section we showed that a \((2 + \varepsilon)\)-approximate solution can be obtained by testing a linear number of candidate edges. Testing each candidate edge entails \( O(n^2) \) shortest path queries. One way to speed up the computation is to compute an approximate stretch factor. As in Section 2.2 we will use Fact 1 by Narasimhan and Smid [27].

Their idea is to compute a well-separated pair decomposition of size \( O(s^d n) \) with respect to \( s = 4(1 + \tau)/\tau \), and then for each well-separated pair \((A_i, B_i)\) select an arbitrary pair \( a_i \in A_i \) and \( b_i \in B_i \). They prove that these are the only pairs for which the \((1 + \tau)^2\)-approximate stretch factor needs to be computed.

We will use their idea to speed up step 6 of EXPANDGRAPH from \( O(n^2) \) to \( O(n/\varepsilon^d) \), i.e., we check a linear number of pairs in order to compute an approximate stretch factor using Fact 1. However, we will not use the fact that only approximate distance queries are needed, instead the exact shortest distance will be computed, thus \( \gamma = 0 \) where \( \gamma \) is as stated in Fact 1. There will be two main changes in the EXPANDGRAPH algorithm; two well-separated pair decompositions will be computed and the computation of the stretch factor will be different. Instead of computing the exact stretch factor of \( G \) with the candidate edge \((a_i, b_i)\) added to \( G \), we compute the approximate stretch factor. This is done by a call to APPROXIMATESTRETCHFACTOR, or ASF for short, with parameters \((a_i, b_i), M \) and \( S \). The ASF algorithm is stated in more detail below. Note that the number of point pairs in \( S \) is bounded by \( O(n/\varepsilon^d) \).

**Algorithm** ExpandGraph2\((G, \varepsilon)\)

**Input:** Euclidean graph \( G = (V, E) \) and a real constant \( \varepsilon > 0 \).

**Output:** Euclidean graph \( G' = (V, E \cup \{e\}) \).

1. \( M \leftarrow \text{All-Pairs-Shortest-Path distance matrix of } G \).
2. \( \{(A_i, B_i)\}_{i=1}^\ell \leftarrow \text{WSPD of the set } V \text{ with respect to } s = 256/\varepsilon^2 \).
3. \( \{(C_j, D_j)\}_{j=1}^\ell \leftarrow \text{WSPD of the set } V \text{ with respect to } s' = 4(1 + \varepsilon)/\varepsilon \).
4. for \( j \leftarrow 1 \) to \( \ell \)
5. \quad Select an arbitrary point \( c_j \) of \( C_j \) and an arbitrary point \( d_j \) of \( D_j \).
6. \( S = \{(c_1, d_1), \ldots, (c_\ell, d_\ell)\} \)
7. \( t' \leftarrow \infty \).
8. for \( i \leftarrow 1 \) to \( k \)
9. Select an arbitrary point \( a_i \) of \( A_i \) and an arbitrary point \( b_i \) of \( B_i \).
10. \( t_i \leftarrow \operatorname{ASF}((a_i, b_i), M, S) \).
11. if \( t_i < t' \)
12. then \( t' \leftarrow t_i \) and \( e \leftarrow (a_i, b_i) \)
13. return \( G' = (V, E \cup \{e\}) \).

For completeness we also state the \( \operatorname{ASF} \) algorithm.

**Algorithm** \( \operatorname{ASF}((a, b), M, S) \)

**Input:** Vertex pair \((a, b) \in V^2\), distance matrix \( M \) and a set of point pairs \( S \).

**Output:** A real value \( D \).
1. \( D \leftarrow 1 \)
2. for each point pair \((c_j, d_j)\) in \( S \)
3. \( \text{dist} \leftarrow \min \{M[c_j, d_j], M[c_j, a] + \|a\| + M[b, d_j], M[c_j, b] + \|b\| + M[a, d_j]\} \)
4. \( D \leftarrow \max \{D, \text{dist}/|c_j d_j|\} \)
5. return \( D \).

**Theorem 4.8.** Given a Euclidean graph \( G = (V, E) \) and a real constant \( \epsilon > 0 \) one can in \( O(nm + n^2(\log n + 1/\epsilon^2d)) \) time, using \( O(n^2) \) space, compute a \( t \)-spanner \( G' = (V, E \cup \{e\}) \) such that \( t' \leq (2 + \epsilon) \cdot t_p \).

**Proof.** The complexity of all steps of the algorithm, except step 10, is as in Lemma 4.4. Steps 1–7 require \( O(nm + n^2 \log n + n/\epsilon^2d) \) time. It remains to consider step 10 of the algorithm. Note that the number of times step 10 is executed is \( O(n/\epsilon^2d) \). Procedure \( \operatorname{ASF} \) performs \( O(n/\epsilon^2d) \) shortest-path queries, instead of \( O(n^2) \), thus the total time needed by step 10 is \( O(n^2/\epsilon^2d \cdot n) \). Summing up the running times gives the stated time complexity.

In Lemma 4.6 it was proven that the solution returned by algorithm \( \operatorname{EXPANDGRAPH} \) had a stretch factor that was at most a factor \((2 + \epsilon)\) worse than the stretch factor of an optimal solution. Since the modified algorithm does not compute the exact stretch factor of a candidate graph, but instead computes a \((1 + \epsilon)^2\)-approximate stretch factor it is not hard to verify that the same arguments as in Lemma 4.6 can be applied to prove that the algorithm \( \operatorname{EXPANDGRAPH} \) returns a graph with stretch factor at most \((1 + \epsilon)^2 \cdot (2 + \epsilon) \cdot t_p \).

Setting \( \epsilon = \min \{\epsilon/10, 1\} \), concludes the proof of the theorem. \( \square \)

**5. A special case: \( G \) has constant stretch-factor.** In the special case when the stretch factor of a graph \( G \) is known to be constant there are well-known tools that we can use to decrease both the time complexity and the space complexity of the algorithms and improve the approximation factor.

**Fact 2.** ([18]) Let \( V \) be a set of \( n \) points in \( \mathbb{R}^d \), let \( t > 1 \) and \( 0 < \epsilon \leq 1 \) be real numbers, and let \( G = (V, E) \) be a \( t \)-spanner for \( V \). In \( O(m + n^{2d}/t^2 \cdot (\log n + (t/\epsilon)^d)) \) time, we can preprocess \( G \) into a data structure of size \( O(n^{2d}/t^2 \cdot n \log n) \) such that for any two distinct points \( p \) and \( q \) in \( V \), a \((1 + \epsilon)^2\)-approximation to the shortest-path distance between \( p \) and \( q \) in \( G \) can be computed in time \( O((t^5/\epsilon^2)^d) \).

The query structure in Fact 2 is denoted \( M' \) and is constructed by algorithm \( \operatorname{QUERYSTRUCTURE} \). We have to use a modified version of \( \operatorname{ASF} \), denoted \( \operatorname{ASF}' \), that takes the query structure \( M' \) as input instead of the matrix \( M \). The shortest path distance queries using \( M \) in \( \operatorname{ASF} \) are replaced in \( \operatorname{ASF}' \) by performing approximate shortest path distance queries using \( M' \).

Next we state the main algorithm. Recall that the parameter \( t \) is a constant and an upper bound on the stretch factor of the input graph \( G \). Also note that this algorithm only needs one well-separated pair decomposition.

**Algorithm** \( \operatorname{EXPANDGRAPH} \) \((G, t, \epsilon)\)

**Input:** Euclidean \( t \)-spanner \( G = (V, E) \) and two real constants \( t > 1 \) and \( \epsilon > 0 \).

**Output:** Euclidean graph \( G' = (V, E \cup \{e\}) \).
1. \( M' \leftarrow \operatorname{QUERYSTRUCTURE}((G, t, \epsilon) \text{ using Fact 2.}) \)
2. \( \{(A_i, B_j)\}_{i=1}^{k} \leftarrow \operatorname{WSPD} \text{ of } V \text{ with respect to the separation constant } s = 8(t + 1)/\epsilon \).
3. for \( j \leftarrow 1 \text{ to } k \)
4. Select an arbitrary point \( a_j \) of \( A_j \) and an arbitrary point \( b_j \) of \( B_j \).
In the second inequality we used Lemma 4.2, in the fifth inequality we used the fact that in the final step we used that $\varepsilon$ to the previous algorithms. This allows us to improve the approximation factor.

### Lemma 5.1. ExpandGraph3 runs in $O((t^7/\varepsilon^4)\cdot n^2)$ time and uses $O((t^3/\varepsilon^2)\cdot n\cdot \log(t\cdot n))$ space.

**Proof.** The time complexity of steps 1–3 is dominated by step 1, thus $O(m+n(t^7/\varepsilon^4)\cdot (\log n + (t/\varepsilon)^d))$ time. Step 8 is executed $O((t/\varepsilon)^d\cdot (t^5d/\varepsilon^{2d}))$ time according to Facts 1 and 2. Summing up the time bounds gives the time bound stated in the algorithm.

The space bound follows since the approximate distance oracle stated in Fact 2 only uses $O((t^3/\varepsilon^2)\cdot n\cdot \log(t\cdot n))$ space, instead of the quadratic space needed earlier. \(\square\)

Now, we show that this algorithm computes a $(1+\varepsilon)$-approximation of the optimal solution. Note that in ExpandGraph3 the separation constant depends both on $\varepsilon$ and $t$ which is the main difference compared to the previous algorithms. This allows us to improve the approximation factor.

### Lemma 5.2. Let $G = (V, E)$ be a Euclidean graph with constant stretch factor $t$ and a positive real constant $\varepsilon$, and let $\{(A_i, B_i)\}_{i=1}^t$ be a well-separated pair decomposition of $V$ with respect to $s = \frac{8(t+1)}{\varepsilon}$. For every pair $(A_i, B_j)$ and any elements $a_1, a_2 \in A_i$ and $b_1, b_2 \in B_j$, let $G_1 = (V, E \cup \{(a_1, b_1), (a_2, b_2)\})$ and $G_2 = (V, E \cup \{(a_2, b_2)\})$, and let $t_1$ and $t_2$ denote the stretch factor of $G_1$ and $G_2$, respectively. It holds that $t_1 \leq (1+\varepsilon)t_2$.

**Proof.** It suffices to prove that for every pair of points $(u, v) \in \Delta(a_2, b_2)$ there exists a path in $G_1$ of length at most $(1+\varepsilon)\cdot d_{G_2}(u, v)$. Without loss of generality we may assume that the shortest path between $u$ and $v$ in $G_2$, goes from $u$ to $a_2$ and then to $b_2$. We have:

$$d_{G_1}(u, v) \leq d_{G}(u, a_2) + d_{G}(a_2, a_1) + |a_1b_1| + d_{G}(b_1, b_2) + d_{G}(b_2, v)$$

$$\leq d_{G}(u, a_2) + t|a_1a_2| + |a_1b_1| + t|b_1b_2| + d_{G}(b_2, v)$$

$$\leq d_{G}(u, a_2) + \frac{4t}{s}|a_2b_2| + (1 + \frac{1}{4s})|a_2b_2| + d_{G}(b_2, v)$$

$$< d_{G}(u, a_2) + \frac{|a_2b_2| + d_{G}(b_2, v) + \frac{8t}{s}|a_2b_2|}{t+1}$$

$$= d_{G_2}(u, v) + \frac{t\varepsilon}{t+1}|a_2b_2|$$

In the second inequality we used Lemma 4.2, in the fifth inequality we used the fact that $s = \frac{8(t+1)}{\varepsilon}$ and in the final step we used that $d_{G_2}(u, v) \geq |a_2b_2|$ since $(u, v) \in \Delta(a_2, b_2)$. The lemma follows. \(\square\)

### Lemma 5.3. Algorithm ExpandGraph3 returns a graph with stretch factor at most $(1+\varepsilon)^3 \cdot t_P$.

**Proof.** Assume that $t_P$ is the stretch factor of an optimal solution $G \cup \{(p, q)\}$, and let $G'$ with stretch factor $t_C$ be the output of the above algorithm.

We will use the same notations as in the algorithm. For each $i$ let $t_i^* = t_i$ be the stretch factor of $G_i = G \cup \{(a_i, b_i)\}$. According to Fact 1 we have $t_i^* \leq t_i \leq (1+\varepsilon)^2 \cdot t_i^*$, for each $i$.

Let $(A_j, B_q)$ be the pair in the well-separated pair decomposition such that $p \in A_j$ and $q \in B_q$, or $p \in B_j$ and $q \in A_j$. From Lemma 5.2 it follows that $t_j^* \leq (1+\varepsilon)^2 \cdot t_j$. As a result it follows that $t_C \leq t_j \leq (1+\varepsilon)^2 \cdot t_j^* \leq (1+\varepsilon)^3 \cdot t_P$. Therefore $t_P \leq t_C \leq (1+\varepsilon)^3 \cdot t_P$ which completes the lemma. \(\square\)

The following theorem follows by setting $\varepsilon = \min\{\varphi/15, 1\}$ and combining Lemmas 5.1 and 5.3, we have the following:

### Theorem 5.4. Let $V$ be a set of $n$ points in $\mathbb{R}^d$, let $d > 1$ and $\varphi > 0$ be real numbers, and let $G = (V, E)$ be a $t$-spanner of $V$. One can in $O((t^7/\varphi)^d\cdot n^2)$ time, using $O((t^3/\varphi)^d\cdot n\cdot \log(tn))$ space, compute a $t'$-spanner $G' = (V, E \cup \{e\})$ such that $t' \leq (1+\varepsilon)^3 \cdot t_P$. 

11
6. Concluding remarks. We considered the problem of adding an edge to a Euclidean graph such that the stretch factor of the resulting graph is minimized, and gave several algorithms. Our main result is a $(2 + \varepsilon)$-approximation algorithm with running time $O(nm + n^2(\log n + 1/\varepsilon^4))$ using $O(n^2)$ space. Several problems remain open.

1. Is there an exact algorithm with running time $o(n^4)$?
2. Can we achieve a $(1 + \varepsilon)$-approximation within the same time bound as in Theorem 4.8?
3. A natural extension is to allow more than one edge to be added. Can we generalize our results to this case?

7. Acknowledgements. The authors would like to thank René van Oostrum for fruitful discussions during the early stages of this work, Mohammad Ali Abam for discussions about Section 2.2 and Sergio Cabello for simplifying the algorithm in Section 5.

Finally, we thank the anonymous referees for many insightful comments and suggestions on how to improve the paper.

REFERENCES


[21] S. Langerman, P. Morin, and M. A. Soss. Computing the maximum detour and spanning ratio of planar paths, and

[23] C. Levcopoulos and A. Lingas. There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. Algorithmica, 8:251–256, 1992.


