Finding the Best Shortcut in a Geometric Network

Mohammad Farshi[1,⋆], Panos Giannopoulos[2,⋆⋆], and Joachim Gudmundsson[3]

[1] Dept. of Computer Science, TU Eindhoven, The Netherlands. m.farshi@tue.nl
[2] Institut für Informatik, Freie Universität Berlin, Germany. panos@inf.fu-berlin.de
[3] NICTA⋆⋆⋆Sydney, Australia. joachim.gudmundsson@nicta.com.au

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

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 $e = (u, v)$ has a weight $wt(e)$. A Euclidean network is a geometric network where the weight of the edge $e = (u, v)$ is equal to the Euclidean distance $d(u, v)$ between its two endpoints u and v. Let $t > 1$ be a real number. We say that G is a *t-spanner* for V, if for each pair of points $u, v \in V$, there exists a path in G of weight at most t times the Euclidean distance between u and v. The minimum t such that G is a t-spanner for V is called the stretch factor, or dilation, of G.

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 and have been a subject of considerable research. Recently spanners found interesting practical applications in areas such as metric space searching [26, 27] and broadcasting in communication networks [2, 14, 24].

Several well-known theoretical results also use the construction of t-spanners as a building block, for example, Rao and Smith [28] made a breakthrough by showing an optimal $\mathcal{O}(n \log n)$-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 spanners has received considerable attention from a theoretical perspective, see [1, 3–5, 8–10, 16, 19, 20, 22, 23, 29, 31], and the surveys [12, 30].

All the above algorithms construct a network from scratch but in many applications the geometric network is already given, and the problem at hand is to extend the network with an additional

⋆ Supported by Ministry of Science, Research and Technology of I. R. Iran.
⋆⋆ Research partially conducted at IICS, Utrecht University, The Netherlands.
⋆⋆⋆ National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

Input graph	Apx. factor	Time complexity	Space	Section
Metric	1	$\mathcal{O}(n^3 m + n^4 \log n)$	$\mathcal{O}(n)$	2
Metric	1	$\mathcal{O}(n^4)$	$\mathcal{O}(n^2)$	2
Euclidean	$1+\varepsilon$	$\mathcal{O}(n^3/\varepsilon^d))$	$\mathcal{O}(n^2)$	2.1
Metric	3	$\mathcal{O}(nm + n^2 \log n)$	$\mathcal{O}(n)$	3
Metric	$2+\varepsilon$	$\mathcal{O}(nm + n^2(\log n + 1/\varepsilon^{3d})$	$\mathcal{O}(n^2)$	4
t-spanner	$1+\varepsilon$	$\mathcal{O}((t^7/\varepsilon^4)^d \cdot n^2)$	$\mathcal{O}((t^3/\varepsilon^2)^d \, n \log(tn))$	5

Table 1. Complexity bounds for the algorithms presented in the paper.

edge, or edges, while minimizing the stretch factor of the resulting graph. Surprisingly this problem has not been studied previously, 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.

We will denote by $|uv|$ the Euclidean distance between u and v, and $\delta_G(u,v)$ denotes the shortest path between u and v in G with length $d_G(u,v)$. Finally, $G_\mathcal{P}$ will denote the optimal solution, while $t_\mathcal{P}$ and t denotes the stretch factor of $G_\mathcal{P}$ and the input graph G respectively.

2 Three simple algorithms

We consider the problem of computing an optimal solution $G_\mathcal{P}$. That is, we are given a t-spanner $G = (V,E)$, and the aim is to compute a $t_\mathcal{P}$-spanner $G_\mathcal{P} = (V, E \cup \{e\})$.

A naïve approach to decide which edge to add is to test every possible candidate edge. The number of such edges is obviously $\left(\frac{n(n-1)}{2} - m\right) = \mathcal{O}(n^2)$. Testing a candidate edge e entails computing the stretch factor of the graph $G' = (V, E \cup \{e\})$, therefore we briefly consider the problem of computing the stretch factor of a given Euclidean graph. This problem has recently received considerable attention, see for example [11, 13, 21, 25].

A trivial upper bound is obtained by computing the All-Pairs-Shortest-Path for the given graph G. Running Dijkstra's algorithm – implemented using Fibonacci heaps – gives the stretch factor of G in time $\mathcal{O}(mn + n^2 \log n)$ using linear space. This algorithm 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, 21, 25].

Applying the above bounds for computing the exact stretch factor of a Euclidean graph gives us that $G_\mathcal{P}$ can be computed in time $\mathcal{O}(n^3(m + n \log n))$ using linear space.

An 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 if there is a shorter path using the edge (u,v). That is, we only have to check the length of the paths $\delta_G(x,u) + |uv| + \delta_G(v,y)$, $\delta_G(x,v) + |vu| + \delta_G(u,y)$ and $\delta_G(x,y)$, which can be done in constant time since shortest path distances between every pair of vertices in G already have been computed (provided that we store this information). Hence by first computing all-pair-shortest paths of G we obtain:

Lemma 1. *Given a Euclidean graph G, an optimal solution $G_{\mathcal{P}}$ can be computed in time $\mathcal{O}(n^4)$ using $\mathcal{O}(n^2)$ space.*

Proof. Computing the all-pair-shortest path requires cubic time and the result is stored in an $n \times n$ matrix. The $\mathcal{O}(n^2)$ edges are tested for insertion, for each candidate edge one performs $\mathcal{O}(n^2)$ shortest path queries, which each can be answered in constant time as described above. □

The above lemma can be generalized to hold for any graph for which the weight of the edges obey the triangle inequality.

2.1 A $(1+\varepsilon)$-approximation

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 $\mathcal{O}(n^2)$ shortest path queries. One way to speed up the computation is to compute an approximate stretch factor. t' is said to be a β-approximate stretch factor of G if $t_G \leq t' \leq \beta \cdot t_G$, where t_G is the stretch factor of G. The problem of computing the approximate stretch factor of a geometric graph was considered by Narasimhan and Smid in [25]. They showed the following fact:

Fact 1 *(Narasimhan and Smid [25]) Given a Euclidean graph G and a real value $\varepsilon > 0$, a $(1+\varepsilon)^2$-approximative stretch factor of G can be computed by performing $\mathcal{O}(n/\varepsilon^d)$ many $(1+\gamma)$-approximate distance queries, where γ is a positive constant smaller than ε.*

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 $\mathcal{O}(n)$ shortest path queries as stated in Fact 1. As a results the time to compute the stretch factor decreases from $\mathcal{O}(n^2)$ to $\mathcal{O}(n/\varepsilon^d)$, thus the total running time decreases from $\mathcal{O}(n^4)$ to $\mathcal{O}(n^3/\varepsilon^d)$.

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

Proof. The time bound follows from the above discussion, it remains to prove that G' has stretch factor $((1+\varepsilon) \cdot t_{\mathcal{P}})$. For each candidate graph G'_i let t'_i be its approximate stretch factor and let t_i be its exact dilation.

Set $\varepsilon' = \sqrt{1+\varepsilon} - 1$. From Fact 1 it follows that for each candidate graph G'_i, $t'_i \leq (1+\varepsilon')^2 \cdot t_i$. Assume that $t_{\mathcal{P}} = t_j$ for some index j. As a result it holds that:

$$t_{\mathcal{P}} = t_j \leq t'_j \leq (1+\varepsilon')^2 \cdot t_j = (1+\varepsilon')^2 \cdot t_{\mathcal{P}} = (1+\varepsilon) \cdot t_{\mathcal{P}}.$$

□

3 Adding a bottleneck edge

In this section we study the approach of adding an edge between a pair of vertices in G that decides the stretch factor of G.

Consider an optimal solution $G_{\mathcal{P}}$ and denote by x and y the two endpoints of an edge added to G to obtain $G_{\mathcal{P}}$. Assume that a pair of vertices deciding the stretch factor of G is (u, v), i.e., the length of the path between u and v in G is exactly $t \cdot |uv|$. We call this edge a bottleneck edge of G. Let $G_{\mathcal{B}}$ be a graph obtained from G by adding one bottleneck edge, and let $t_{\mathcal{B}}$ be the stretch factor of $G_{\mathcal{B}}$. Note that $G_{\mathcal{B}}$ can be computed in the same time as the stretch factor of G, i.e., in $\mathcal{O}(mn + n^2 \log n)$ time for Euclidean graphs.

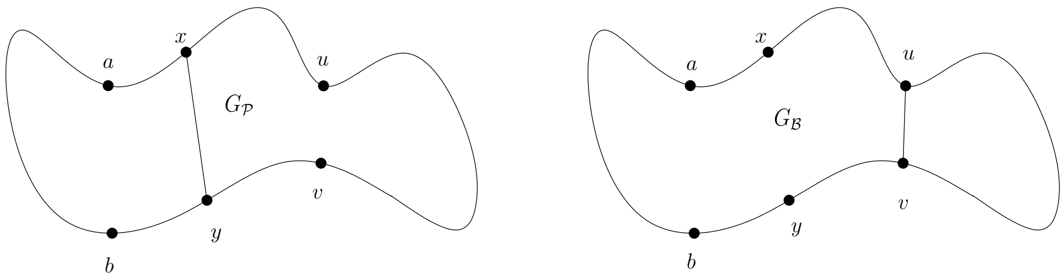

Fig. 1. (x, y) is the optimal edge added to G and (u, v) is a bottleneck edge.

Lemma 2. *Given a Euclidean graph G in $\mathbb{R}^d$ it holds that $t_{\mathcal{B}} < 3t_{\mathcal{P}}$.*

Proof. Recall that t denotes the stretch factor of G. First note that if $t_{\mathcal{P}} > t/3$ then the lemma holds and we are done. Thus we may assume that $t_{\mathcal{P}} \leq t/3$. The proof of the lemma is done by considering a pair of vertices, denoted (a, b), that decides the stretch factor of $G_{\mathcal{B}}$. Note that the path $\delta_{G_{\mathcal{P}}}(a, b)$ must include the added edge (x, y), otherwise $d_{G_{\mathcal{P}}}(a, b) = d_G(a, b) \geq d_{G_{\mathcal{B}}}(a, b)$ and we are done. Also, we will assume without loss of generality that a shortest path in $G_{\mathcal{P}}$ from a to b goes from a to x and then to b via y. The same is assumed about a shortest path in $G_{\mathcal{P}}$ between u and v, i.e., from u to x and then to v via y.

Our first step is to bound the distance between the bottleneck vertices u and v. This is done by bounding the length of the path in G between x and y as follows, see Fig. 1.

$$\begin{aligned} d_G(u, v) &\leq d_{G_{\mathcal{P}}}(u, v) - |xy| + d_G(x, y) \\ &\leq t_{\mathcal{P}} \cdot |uv| - |xy| + t \cdot |xy| \\ &\leq \frac{t}{3} \cdot |uv| - |xy| + t \cdot |xy| \\ &< \frac{t}{3} \cdot |uv| + t \cdot |xy|. \end{aligned}$$

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

$$|uv| < 3/2 \cdot |xy|. \tag{1}$$

Also,

$$\begin{aligned} t \cdot |uv| &= d_G(u,v) \\ &\leq d_G(u,a) + d_G(a,b) + d_G(b,v) \\ &\leq d_G(u,a) + t \cdot |ab| + d_G(b,v) \end{aligned}$$

which implies that

$$t \cdot (|uv| - |ab|) \leq d_G(u,a) + d_G(b,v), \tag{2}$$

and

$$\begin{aligned} d_G(a,u) + 2|xy| + d_G(v,b) &\leq d_G(a,x) + d_G(x,u) + 2|xy| + d_G(v,y) + d_G(y,b) \\ &= d_{G_{\mathcal{P}}}(a,b) + d_{G_{\mathcal{P}}}(u,v) \\ &\leq t_{\mathcal{P}}(|ab| + |uv|), \end{aligned} \tag{3}$$

which gives that

$$d_G(a,u) + d_G(v,b) \leq t_{\mathcal{P}}(|ab| + |uv|) - 2|xy|. \tag{4}$$

By putting together (2) and (4) we have

$$\begin{aligned} t(|uv| - |ab|) &\leq d_G(a,u) + d_G(v,b) \\ &\leq t_{\mathcal{P}}(|ab| + |uv|) - 2|xy| \\ &< t_{\mathcal{P}}(|ab| + |uv|), \end{aligned}$$

which implies that

$$|ab|(t_{\mathcal{P}} + t) > |uv|(t - t_{\mathcal{P}})$$

and

$$|ab| > \frac{t - t_{\mathcal{P}}}{t_{\mathcal{P}} + t} \cdot |uv| > \frac{t - \frac{t}{3}}{\frac{t}{3} + t} \cdot |uv| = \frac{1}{2} \cdot |uv|. \tag{5}$$

Now we are ready to put together the results:

$$\begin{aligned} t_{\mathcal{B}} \cdot |ab| &= d_{G_{\mathcal{B}}}(a,b) \\ &\leq d_G(a,u) + |uv| + d_G(v,b) \\ &< d_G(a,u) + \frac{3}{2}|xy| + d_G(v,b) && \text{(from (1))} \\ &< d_G(a,u) + 2|xy| + d_G(v,b) \\ &\leq t_{\mathcal{P}}\,(|ab| + |uv|) && \text{(from (3))} \\ &< 3t_{\mathcal{P}} \cdot |ab| && \text{(from (5))} \end{aligned}$$

This completes the proof of the lemma since $t_{\mathcal{B}} < 3t_{\mathcal{P}}$. □

The above lemma can be generalized to hold for any graph for which the edge weights obey the triangle inequality. We conclude by stating the main result of this section followed by a lower bound for the bottleneck approach.

Theorem 2. *Given a Euclidean graph $G = (V, E)$ one can in $\mathcal{O}(mn + n^2 \log n)$ time, using $\mathcal{O}(n)$ space, compute a $t_{\mathcal{B}}$-spanner $G' = (V, E \cup \{e\})$ where $t_{\mathcal{B}} < 3t_{\mathcal{P}}$.*

Observation 1 *There exists a graph G such that $(2 - \varepsilon) \cdot t_{\mathcal{P}} \leq t_{\mathcal{B}}$, for any $0 < \varepsilon < 1$.*

Proof. Consider the graph G, as in Fig. 2a. More specifically, G is a graph with ten vertices $p_i = ((i-1) \mod 5, \lfloor i/5 \rfloor \cdot \delta)$, $1 \leq i \leq 10$, 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 δ: (p_1, p_6) is the bottleneck in G and $t_{\mathcal{B}} = \frac{4+\delta}{\delta}$, see Fig. 2b.

In the case when edge (p_2, p_7) is added to G, as shown in Fig. 2c, the resulting graph has stretch factor $(2 + \delta)/\delta$. Combining the upper and lower bounds gives $\frac{t_{\mathcal{B}}}{t_{\mathcal{P}}} = \frac{4+\delta}{2+\delta} = (2 - \varepsilon)$, where the last equality follows if we set $\delta = \frac{2\varepsilon}{1-\varepsilon}$. □

Recently, Grüne [15] improved the lower bound in Observation 1 to $(3 - \varepsilon)$, so the upper bound stated in Lemma 2 is tight.

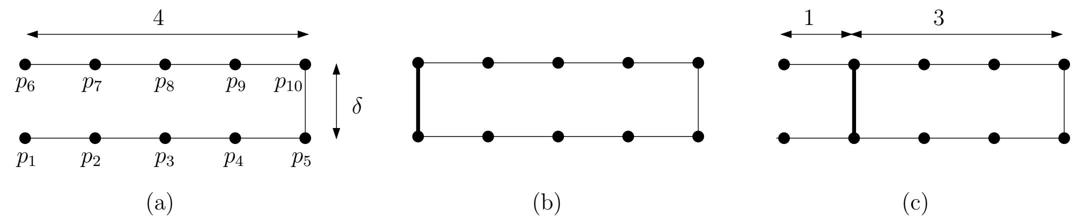

Fig. 2. (a) The input graph G. (b) The bottleneck solution compared to (c) the optimal solution.

4 A $(2 + \varepsilon)$-approximation

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

Moreover, Section 4.3 shows 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 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 1.

Lemma 3 ([6]). *Let A and B be two finite sets of points that are well-separated w.r.t. s, let x and p be points of A, and let y and q be points of B. Then (i) $|xy| \leq (1+2/s) \cdot |xq|$, (ii) $|xy| \leq (1+4/s) \cdot |pq|$, and (iii) $|px| \leq (2/s) \cdot |pq|$.*

Definition 2 ([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, $\{A_1, B_1\}, \dots, \{A_m, B_m\}$, such that*

1. *$A_i \cap B_i = \emptyset$, for all $i = 1, \dots, m$,*
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 $m = \mathcal{O}(s^d n)$ can be computed in $\mathcal{O}(s^d n + n \log n)$ 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. It will below be shown 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 straight-forward. 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. Finally, steps 4–8, each candidate edge is tested by computing the stretch factor of the candidate graph.

Algorithm ExpandGraph(G, ε)
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. *Algorithm* ExpandGraph *requires* $\mathcal{O}(n^3/\varepsilon^{2d})$ *time and* $\mathcal{O}(n^2)$ *space.*

Proof. The complexity of all steps of the algorithm, except step 6, is straight-forward to calculate. Recall that step 1 requires $\mathcal{O}(mn+n^2\log n)$ time and quadratic space, and step 2 requires $\mathcal{O}(n/\varepsilon^{2d}+n\log n)$ 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 $\mathcal{O}(n/\varepsilon^{2d})$.

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

It remains to analyze the quality of the solution obtained from algorithm ExpandGraph. We need to compare the graph resulting from adding an optimal edge to G and the graph G' resulting from 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_i, b_i) tested by the algorithm should be a good candidate. However the separation constant of our well separated pair decomposition only depends on ε 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. Therefore we will, in Lemma 5, prove the existence of a "short" edge e' that is a good approximation of the optimal edge and then, in Lemma 6, we show that ExpandGraph computes a good approximation of e'. Note also that in Section 5 it will be shown that if the separation constant also depends on the stretch factor of G then a $(1 + \varepsilon)$-approximation can be proven.

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, the set of point pairs for which the shortest path between them in $G \cup \{(p, q)\}$ passes through (p, q).

Lemma 5. *For any given constant* $0 < \lambda \leq 1$, *there exists a point pair* $p, q \in V$ *such that for every pair* $(u, v) \in \Delta(p, q)$ *it holds that* $|uv| \geq \frac{\lambda}{2}|pq|$, *and the stretch factor of* $G \cup \{(p, q)\}$ *is bounded by* $(2 + \lambda) \cdot t_{\mathcal{P}}$.

Proof. The proof is done in two steps. First a point pair $p_j, q_j \in V$ is selected that fulfills the first requirement of the lemma. Then, we prove the second requirement, i.e., the stretch factor of $G \cup \{(p_j, q_j)\}$ is bounded by $(2 + \lambda) \cdot t_{\mathcal{P}}$.

Consider an optimal solution $G_1 = G \cup \{(p_1, q_1)\}$, with stretch factor $t_1 = t_{\mathcal{P}}$. If for each point pair $(u, v) \in \Delta(p_1, q_1)$ it holds that $|uv| \geq \lambda/2 \cdot |p_1q_1|$ then we have found the point pair $(p_j = p_1, q_j = q_1)$ that we are searching for. Otherwise, let $e_2 = (p_2, q_2)$ denote the closest pair in $\Delta(p_1, q_1)$, and continue the search for (p_j, q_j). See Fig. 3a for an illustration. Note that for each $(u, v) \in \Delta(p_1, q_1)$ it holds that $|p_2q_2| \leq |uv|$ and, especially, $|p_2q_2| \leq |p_1q_1|$ since $(p_1, q_1) \in \Delta(p_1, q_1)$. Furthermore, $|p_2q_2| \leq (\lambda/2) \cdot |p_1q_1|$ since there is a point pair $(u, v) \in \Delta(p_1, q_1)$ such that $|uv| < (\lambda/2) \cdot |p_1q_1|$.

We define e_3 in a similar way, that is, if for each point pair $(u, v) \in \Delta(p_2, q_2)$ it holds that $|uv| \geq \lambda/2 \cdot |p_2q_2|$ then we have found the point pair $(p_j = p_2, q_j = q_2)$ that we are searching for. Otherwise, let $e_3 = (p_3, q_3)$ denote the closest pair in $\Delta(p_2, q_2)$.

We continue to define the edges $e_4, \ldots, e_j$ and the corresponding graphs $G_4, \ldots, G_j$ until we find a point pair (p_j, q_j) for which there is no edge e_{j+1} such that $(p_{j+1}, q_{j+1}) \in \Delta(p_j, q_j)$ and $|p_{j+1}q_{j+1}| < \lambda/2 \cdot |p_jq_j|$. Based on the construction we have two basic properties: (a) $\{(p_i, q_i)\}$ make a sequence of decreasing length, i.e., if $i < j$ then $|p_i q_i| \geq |p_j q_j|$, and (b) for each i, $|p_{i+1} q_{i+1}| \leq (\lambda/2) \cdot |p_i q_i|$.

We claim that G_j has stretch factor at most $(2 + \lambda) \cdot t_{\mathcal{P}}$. Before we continue we need to prove:

$$d_{G_i}(p_{i+1}, q_{i+1}) \leq t_1 \cdot |p_{i+1} q_{i+1}|. \tag{6}$$

The inequality is obviously true for $i = 1$. For $i > 1$ it holds that $|p_{i+1} q_{i+1}| < |p_2 q_2|$ 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 $d_G(p_{i+1}, q_{i+1}) = d_{G_1}(p_{i+1}, q_{i+1}) \leq t_1 \cdot |p_{i+1}, q_{i+1}|$. Since G is a subgraph of G_i, the length of the shortest path in G_i 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 $t_1 \cdot |p_{i+1} q_{i+1}|$. Thus, inequality (6) holds.

We continue with the second part of the proof. It will be shown that for every pair $u, v \in V$ it holds that $d_{G_j}(u, v) \leq (2 + \lambda) \cdot d_{G_1}(u, v)$. If $(u, v) \notin \Delta(p_1, q_1)$ then we are done since $d_{G_j}(u, v) \leq d_G(u, v) = d_{G_1}(u, v)$. Otherwise, if $(u, v) \in \Delta(p_1, q_1)$, the following holds (see also Fig. 3a for an illustration):

$$\begin{aligned}
d_{G_j}(u, v) &\leq d_{G_1}(u, v) - |p_1 q_1| + (d_{G_1}(p_2, q_2) - |p_1 q_1|) + \ldots + (d_{G_{j-1}}(p_j, q_j) - |p_{j-1} q_{j-1}|) + |p_j q_j| \\
&< t_1 \cdot |uv| - |p_1 q_1| + ((t_1 - 1) \cdot |p_2 q_2|) + \ldots + ((t_1 - 1) \cdot |p_j q_j|) + |p_j q_j| && \text{(from (6))} \\
&< t_1 \cdot |uv| + (t_1 - 1)(|p_2 q_2| + \ldots + |p_j q_j|) && (\text{since } |p_j q_j| < |p_1 q_1|) \\
&< t_1 \cdot |uv| + t_1 \cdot \sum_{i=2}^{j} \left(\frac{\lambda}{2}\right)^{i-2} |p_2 q_2| && (\text{since } |p_{i+1} q_{i+1}| \leq (\lambda/2) \cdot |p_i q_i|) \\
&\leq t_1 \cdot |uv| + t_1 \cdot |uv| \cdot \sum_{i=1}^{j-2} \left(\frac{\lambda}{2}\right)^{i} && (\text{since } |p_2 q_2| \leq |uv|) \\
&< (2 + \lambda) \cdot t_1 \cdot |uv|
\end{aligned}$$

Thus, $t_j < (2 + \lambda) \cdot t_1$ which concludes the lemma, since $t_1 = t_{\mathcal{P}}$. □

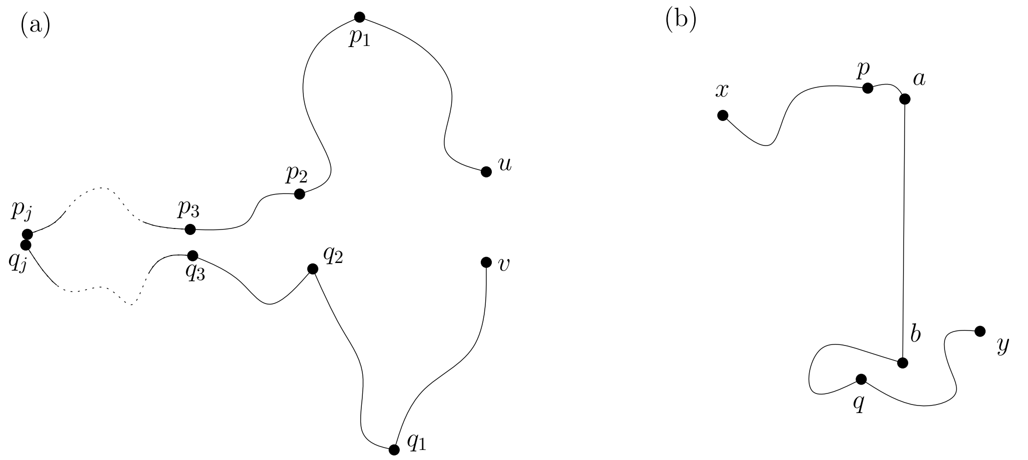

Fig. 3. (a) Illustrating the proof of Lemma 5. (b) Illustrating the proof of Lemma 6.

In the previous lemma we showed the existence of a "short" candidate edge (p, q) 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 6. *For any given constant $0 < \varepsilon \leq 1$ it holds that the graph G' returned by algorithm* ExpandGraph *has stretch factor at most $(2 + \varepsilon) \cdot t_{\mathcal{P}}$.*

Proof. According to Lemma 5 there exists an edge (p, q) such that for every pair $(u, v) \in \Delta(p, q)$ it holds that $|uv| \geq \frac{\lambda}{2}|pq|$, and the stretch factor t_H of $H = G \cup \{(p, q)\}$ is bounded by $(2 + \lambda) \cdot t_{\mathcal{P}}$. Let (A_i, B_i) be the well-separated pair computed in step 2 of the algorithm such that $p \in A_i$ and $q \in B_i$. According to Definition 2 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 $\lambda = \varepsilon/4$ would then prove the lemma since $(2 + \varepsilon/4)(1 + \varepsilon/4) < (2 + \varepsilon)$, for $\varepsilon \leq 1$.

Now we are ready to prove the claim. To compute the stretch factor of G' one performs a shortest path 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 3b. It holds that:

$$d_G(a, p) = d_H(a, p) \quad \text{and} \quad d_G(b, q) = d_H(b, q). \tag{7}$$

Lemma 5 states that if $(x', y') \in \Delta(p, q)$ then $|x'y'| \geq \frac{\varepsilon}{8}|pq|$. But the distances $|ap|$ and $|bq|$ are less than $\frac{2}{s}\,|pq| = \frac{\varepsilon^2}{128}\,|pq|$ which is less than $\frac{\varepsilon}{8}\,|pq|$ since $\varepsilon \leq 1$. As a consequence $(a, p) \notin \Delta(p, q)$ and $(b, q) \notin \Delta(p, q)$, thus $(p, q) \notin \delta_H(a, p)$ and $(p, q) \notin \delta_H(b, q)$. Hence, claim (7) holds, which we will need below.

Next, we consider the length of the path in G' between x and y as illustrated in Fig. 3b. Recall that x and y are two arbitrary points of V for which it holds that $(x, y) \in \Delta(p, q)$.

$$\begin{aligned}
d_{G'}(x, y) &\leq d_G(x, p) + d_G(p, a) + |ab| + d_G(b, q) + d_G(q, y) \\
&\leq d_G(x, p) + d_H(p, a) + |ab| + d_H(b, q) + d_G(q, y) \\
&\leq d_G(x, p) + |ab| + d_G(q, y) + t_H \cdot (|pa| + |bq|) \\
&\leq d_G(x, p) + (1 + 4/s) \cdot |pq| + d_G(q, y) + \frac{4t_H}{s} \cdot |pq| \\
&\leq d_H(x, y) + \frac{8t_H}{s} \cdot |pq| \\
&\leq d_H(x, y) + \frac{64t_H}{\varepsilon s} \cdot |xy| \\
&= d_H(x, y) + \frac{\varepsilon}{4} \cdot t_H \cdot |xy|
\end{aligned}$$

The second inequality follows from (7), the fourth inequality follows from Lemma 3 and the sixth inequality follows from the fact that $(x, y) \in \Delta(p, q)$.

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

$$\frac{d_{G'}(x, y)}{|xy|} \leq \frac{d_H(x, y)}{|xy|} + \frac{\frac{\varepsilon}{4} t_H |xy|}{|xy|} \leq \left(1 + \frac{\varepsilon}{4}\right) \cdot t_H.$$

Finally, according to Lemma 5 and the fact that $\lambda = \varepsilon/4$ it holds that $t_H \leq (2 + \varepsilon/4) \cdot t_{\mathcal{P}}$. This completes the lemma since $(2 + \varepsilon/4)(1 + \varepsilon/4) < (2 + \varepsilon)$. □

We may now conclude this section with the following theorem.

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

4.3 Speed-up the algorithm

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 $\mathcal{O}(n^2)$ shortest path queries. One way to speed up the computation is to compute an approximate stretch factor. As in Section 2.1 we will use Fact 1 by Narasimhan and Smid [25].

Their idea is to compute a well-separated pair decomposition of size $\mathcal{O}(s^d n)$ with respect to $s = 4(1 + \varepsilon)/\varepsilon$, 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 stretch factor needs to be computed.

We will use their idea to speed up step 6 of the algorithm from $\mathcal{O}(n^2)$ to $\mathcal{O}(n/\varepsilon^d)$. On the other hand we will not use the fact that only approximate distance queries are needed, instead the exact shortest distance will be computed, thus $\gamma = 0$. 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.

Algorithm ExpandGraph2(G, ε)
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 $s = 256/\varepsilon^2$.
3. $\{(C_j, D_j)\}_{j=1}^{\ell} \leftarrow$WSPD of the set V with respect to $s' = 4(1 + \varepsilon)/\varepsilon$.
4. **for** $j \leftarrow 1$ **to** ℓ
5. Select an arbitrary point c_j of C_j and an arbitrary point d_j of D_j.
6. $\mathcal{S} = \{(c_1, d_1), \ldots, (c_\ell, d_\ell)\}$
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. $t_i \leftarrow \mathrm{ASF}((a_i, b_i), M, \mathcal{S})$.
11. **if** $t_i < t'$
12. **then** $t' \leftarrow t_i$ and $e \leftarrow (a_i, b_i)$
13. **return** $G' = (V, E \cup \{e\})$.

Instead of computing the exact stretch factor of G_i we make a call to ApproximateStretch-Factor, or ASF for short, with parameters (a_i, b_i), M, and $\mathcal{S}$. Note that the number of point pairs in $\mathcal{S}$ is bounded by $\mathcal{O}(n/\varepsilon^d)$.

Algorithm ASF$(e, M, \mathcal{S})$
Input: Edge $e = (a, b) \in E$, distance matrix M and a set of point pairs $\mathcal{S}$.
Output: A real value $\mathcal{D}$.
1. $\mathcal{D} \leftarrow 1$
2. **for** each point pair (c_j, d_j) in $\mathcal{S}$
3. dist $\leftarrow \min\{M[c_j, d_j],\ \ M[c_j, a] + |ab| + M[b, d_j],\ \ M[c_j, b] + |ba| + M[a, d_j]\}$
4. $\mathcal{D} \leftarrow \max\{\mathcal{D},\ \text{dist}/|c_j d_j|\}$
5. **return** $\mathcal{D}$.

Theorem 4. *Given a Euclidean graph $G = (V, E)$ and a real constant $\epsilon > 0$ one can in $\mathcal{O}(nm + n^2(\log n + 1/\epsilon^{3d}))$ time, using $\mathcal{O}(n^2)$ space, compute a t'-spanner $G' = (V, E \cup \{e\})$ such that $t' \leq (2 + \epsilon) \cdot t_{\mathcal{P}}$.*

Proof. The complexity of all steps of the algorithm, except step 10, is as in Lemma 4. Steps 1–7 requires $\mathcal{O}(mn + n^2 \log n + n/\varepsilon^{2d})$ time. It remains to consider step 10 of the algorithm. Note that the number of times step 10 is executed is $\mathcal{O}(n/\varepsilon^{2d})$. Procedure ASF performs $\mathcal{O}(n/\varepsilon^d)$ shortest-path queries, instead of $\mathcal{O}(n^2)$, thus the total time needed by step 10 is $\mathcal{O}(\frac{n}{\varepsilon^{2d}} \cdot \frac{n}{\varepsilon^d})$. Summing up the running times gives the stated time complexity.

In Lemma 6 it was proven that the solution returned by algorithm ExpandGraph had a stretch factor that was at most a factor $(2 + \varepsilon)$ 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 $(1 + \varepsilon)^2$-approximate stretch factor it is not hard to verify that the same arguments as in Lemma 6 can be applied to prove that the algorithm ExpandGraph2 returns a graph with stretch factor at most $(1 + \varepsilon)^2 \cdot (2 + \varepsilon) \cdot t_{\mathcal{P}}$. Setting $\varepsilon = \min\{\epsilon/10, 1\}$ concludes the proof of the theorem. □

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 the complexity of the algorithms and improve the approximation factor.

Fact 2 *([17]) Let V be a set of n points in $\mathbb{R}^d$, let $t > 1$ and $0 < \varepsilon \leq 1$ be real numbers, and let $G = (V, E)$ be a t-spanner for V. In $\mathcal{O}(m + \frac{nt^{5d}}{\varepsilon^{2d}}(\log n + (t/\varepsilon)^d))$ time, we can preprocess G into a data structure of size $\mathcal{O}(\frac{t^{3d}}{\varepsilon^{2d}} n \log(tn))$ such that for any two distinct points p and q in V, a $(1 + \varepsilon)$-approximation to the shortest-path distance between p and q in G can be computed in time $\mathcal{O}\big((t^5/\varepsilon^2)^d\big)$.*

The query structure in Fact 2 is denoted M' and is constructed by algorithm QueryStructure. We have to use a modified version of ASF, denoted ASF′, that takes the query structure M' as input instead of the matrix M. The shortest path distance queries using M in ASF is replaced in 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.

Algorithm ExpandGraph3(G, t, ε)
Input: Euclidean t-spanner $G = (V, E)$ and two real constants $t > 1$ and $\varepsilon > 0$.
Output: Euclidean graph $G' = (V, E \cup \{e\})$.
1. $M' \leftarrow$QueryStructure(G, t, ε) using Fact 2.
2. $\mathcal{S} = \{(A_i, B_i)\}_{i=1}^{k} \leftarrow$WSPD of V with respect to the separation constant $s = 8(t+1)/\varepsilon$.
3. $t_{\mathcal{C}} \leftarrow \infty$
4. **for** $i \leftarrow 1$ **to** k
5. Select arbitrary points $a_i \in A_i$ and $b_i \in B_i$.
6. $t_i \leftarrow$ASF$'((a_i, b_i), M', \mathcal{S})$.
7. **if** $t_i < t_{\mathcal{C}}$
8. **then** $t_{\mathcal{C}} \leftarrow t_i$ and $e_C \leftarrow (a_i, b_i)$
9. **return** $G' = (V, E \cup \{e_C\})$.

Lemma 7. ExpandGraph3 *runs in* $\mathcal{O}((t^7/\varepsilon^4)^d \cdot n^2)$ *time and uses* $\mathcal{O}((t^3/\varepsilon^2)^d \, n \log(tn))$ *space.*

Proof. The time complexity of steps 1–3 is dominated by step 1, thus $\mathcal{O}(m + n(t^5/\varepsilon^2)^d(\log n + (t/\varepsilon)^d))$ time. Step 6 is executed $\mathcal{O}((t/\varepsilon)^d n)$ times, and each iteration requires $\mathcal{O}((t/\varepsilon)^d n \cdot (t^{5d}/\varepsilon^{2d}))$ 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 $\mathcal{O}((t^3/\varepsilon^2)^d \, n \log(tn))$ space, instead of the quadratic space needed earlier. □

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 ε and t which is the main different compared to the previous algorithms. This allows us to improve the approximation factor.

Lemma 8. *Given a Euclidean graph $G = (V, E)$ with constant stretch factor t and a positive real constant ε. Let $\{(A_i, B_i)\}_{i=1}^{k}$ be a well-separated pair decomposition of V with respect to $s = \frac{8(t+1)}{\varepsilon}$. For every pair (A_i, B_i) and all elements $a_1, a_2 \in A_i$ and $b_1, b_2 \in B_i$, let $G_1 = (V, E \cup \{(a_1, b_1)\})$ and $G_2 = (V, E \cup \{(a_2, b_2)\})$. 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 to v via b_2. We have,

$$\begin{aligned}
d_{G_1}(u, v) &\leq d_G(u, a_2) + d_G(a_2, a_1) + |a_1 b_1| + d_G(b_1, b_2) + d_G(b_2, v) \\
&\leq d_G(u, a_2) + t|a_1 a_2| + |a_1 b_1| + t|b_1 b_2| + d_G(b_2, v) \\
&\leq d_G(u, a_2) + \frac{4t}{s}|a_2 b_2| + (1 + 4/s) \cdot |a_2 b_2| + d_G(b_2, v) \\
&< d_G(u, a_2) + |a_2 b_2| + d_G(b_2, v) + \frac{8t}{s}|a_2 b_2| \\
&= d_{G_2}(u, v) + \frac{t\varepsilon}{t+1}|a_2 b_2| \\
&< (1+\varepsilon) \cdot d_{G_2}(u, v)
\end{aligned}$$

In the second inequality we used Lemma 3, in the fifth inequality we used the fact that $s = 8(t+1)/\varepsilon$ and in the final step we used that $d_{G_2}(u, v) \geq |a_2 b_2|$ since $(u, v) \in \Delta(a_2, b_2)$. The lemma follows. □

Lemma 9. *Algorithm* ExpandGraph3 *returns a graph with stretch factor at most* $(1+\varepsilon)^3 \cdot t_{\mathcal{P}}$.

Proof. Assume that $t_{\mathcal{P}}$ is the stretch factor of an optimal solution $G \cup \{(p,q)\}$, and let G' with dilation $t_{\mathcal{C}}$ be the output of the above algorithm.

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

Let (A_j, B_j) be the pair in the well-separated pair decomposition such that $p \in A_j$ and $q \in B_j$, or $p \in B_j$ and $q \in A_j$. From Lemma 8 it follows that $t_j^* \leq (1+\varepsilon) \cdot t_{\mathcal{P}}$. As a result it follows that $t_{\mathcal{C}} \leq t_j \leq (1+\varepsilon)^2 \cdot t_j^* \leq (1+\varepsilon)^3 \cdot t_{\mathcal{P}}$. Therefore $t_{\mathcal{P}} \leq t_{\mathcal{C}} \leq (1+\varepsilon)^3 \cdot t_{\mathcal{P}}$ which completes the lemma. □

The following theorem follows by setting $\varepsilon = \min\{\epsilon/15, 1\}$ and combining Lemmas 7 and 9.

Theorem 5. *Given a Euclidean t-spanner* $G = (V, E)$ *and two real constants* $t > 1$ *and* $\epsilon > 0$ *it holds that a graph* G' *can be computed in time* $\mathcal{O}((t^7/\epsilon^4)^d \cdot n^2)$ *with stretch factor* $(1+\epsilon) \cdot t_{\mathcal{P}}$ *using* $\mathcal{O}((t^3/\epsilon^2)^d \, n \log(tn))$ *space.*

6 Concluding remarks

We considered the problem of adding a shortcut 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 $\mathcal{O}(nm + n^2(\log n + 1/\varepsilon^{3d}))$ using $\mathcal{O}(n^2)$ space. Several problems remain open.

1. Is there an exact algorithm with running time $o(n^4)$ using linear space?
2. Can we achieve a $(1+\varepsilon)$-approximation within the same time bound as in Theorem 4?
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 2.1 and Sergio Cabello for simplifying the algorithm in Section 5.

References

1. I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.
2. K. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, and O. Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):408–421, 2003.
3. S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th ACM Symposium on Theory of Computing*, pages 489–498, 1995.
4. S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 703–712, 1994.
5. P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry: Theory and Applications*, 28:11–18, 2004.
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. *Journal of the ACM*, 42:67–90, 1995.

7. A. Czumaj and A. Lingas. Fast approximation schemes for Euclidean multi-connectivity problems. In *Proc. 27th International Colloquium on Automata, Languages and Programming*, volume 1853 of *Lecture Notes in Computer Science*, pages 856–868. Springer-Verlag, 2000.
8. G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proc. 9th Annual ACM Symposium on Computational Geometry*, pages 53–62, 1993.
9. G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry and Applications*, 7:297–315, 1997.
10. G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 215–222, 1995.
11. A. Ebbers-Baumann, R. Klein, E. Langetepe, and A. Lingas. A fast algorithm for approximating the detour of a polygonal chain. In *Proc. 9th European Symposium on Algorithms*, volume 2161 of *Lecture Notes in Computer Science*, pages 321–332, Springer-Verlag, 2001.
12. D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers, Amsterdam, 2000.
13. D. Eppstein and K. Wortman. Minimum dilation stars. In *Proc. 21th ACM Symposium on Computational Geometry*, 2005.
14. A. M. Farley, A. Proskurowski, D. Zappala, and K. J. Windisch. Spanners and message distribution in networks. *Discrete Applied Mathematics*, 137(2):159–171, 2004.
15. A. Grüne. Tightness of upper bound on t_B/t_P. Manuscript, March 2005.
16. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. *SIAM Journal of Computing*, 31(5):1479–1500, 2002.
17. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles revisited. In *Proc. 13th International Symposium on Algorithms and Computation*, volume 2518 of *Lecture Notes in Computer Science*, pages 357–368, Springer-Verlag, 2002.
18. J. Gudmundsson, G. Narasimhan, and M. Smid. Fast pruning of geometric spanners. In *Proc. 22nd International Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Comp. Science*, pages 508–520, Springer-Verlag, 2005.
19. J. M. Keil. Approximating the complete Euclidean graph. In *Proc. 1st Scandinavian Workshop on Algorithmic Theory*, volume 318 of *Lecture Notes in Computer Science*, pages 208–213, Springer-Verlag, 1988.
20. J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete and Computational Geometry*, 7:13–28, 1992.
21. S. Langerman, P. Morin, and M. A. Soss. Computing the maximum detour and spanning ratio of planar paths trees, and cycles. In *Proc. 19th International Symposium on Theoretical Aspects of Computer Science*, volume 2285 of *Lecture Notes in Computer Science*, pages 250–261, Springer-Verlag, 2002.
22. 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.
23. C. Levcopoulos, G. Narasimhan, and M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32:144–156, 2002.
24. X.-Y. Li. Applications of computational geomety in wireless ad hoc networks. In X.-Z. Cheng, X. Huang, and D.-Z. Du, editors, *Ad Hoc Wireless Networking*. Kluwer, 2003.
25. G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM Journal of Computing*, 30(3):978–989, 2000.
26. G. Navarro and R. Paredes. Practical construction of metric t-spanners. In *Proc. 5th SIAM Workshop on Algorithm Engineering and Experiments*, pages 69–81, 2003.
27. G. Navarro, R. Paredes, and E. Chvez. t-spanners as a data structure for metric space searching. In *Proc. 9th International Symposium on String Processing and Information Retrieval*, volume 2476 of *Lecture Notes in Comp. Science*, pages 298–309. Springer-Verlag, 2002.
28. S. Rao and W. D. Smith. Approximating geometrical graphs via spanners and banyans. In *Proc. 30th ACM Symposium on the Theory of Computing*, pages 540–550, 1998.
29. J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry & Applications*, 1:99–107, 1991.
30. 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 Publishers, Amsterdam, 2000.
31. P. M. Vaidya. A sparse graph almost as good as the complete graph on points in K dimensions. *Discrete Computational Geometry*, 6:369–381, 1991.