Finding the Best Shortcut in a Geometric Network [*]

Mohammad Farshi[†] Panos Giannopoulos[‡] Joachim Gudmundsson[§]

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 times we consider several approximation 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.

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 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 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 [6] and broadcasting in communication networks [1]. The problem of constructing spanners has received considerable attention from a theoretical perspective, see the surveys [3, 7].

Apx. factor	Time complexity	Space	Sec.
1	$\mathcal{O}(n^3m + n^4 \log n)$	$\mathcal{O}(n)$	2
1	$\mathcal{O}(n^4)$	$\mathcal{O}(n^2)$	2
3	$\mathcal{O}(nm + n^2 \log n)$	$\mathcal{O}(n)$	3
$2+\varepsilon$	$\mathcal{O}(nm + n^2(\log n + 1/\varepsilon^{3d}))$	$\mathcal{O}(n^2)$	4

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

Most known algorithms either construct a spanner given a point set or prunes a given graph, but in many applications the geometric 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. 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 Euclidean graph G construct a graph G' by adding an edge to G such that the stretch factor of G' is minimized.

We present one exact algorithm and several approximation algorithms. 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 G respectively.

2 Finding an optimal solution

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

[*]M.F. supported by Ministry of Science, Research and Technology of I. R. Iran.

[†]Department of Mathematics and Computing Science, TU Eindhoven, 5600 MB, Eindhoven, the Netherlands. m.farshi@tue.nl

[‡]Department of Computer Science, Utrecht University, Utrecht, the Netherlands. panos@cs.uu.nl

[§]NICTA, Sydney, Australia. National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council. Joachim.Gudmundsson@nicta.com.au

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.

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 and trees [4, 5].

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 $\delta_G(x, u) + |uv| + \delta_G(v, y)$ and $\delta_G(x, v) + |vu| + \delta_G(u, y)$, which can be done in constant time since $\delta_G(x, u)$ and $\delta_G(v, y)$ 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.*

3 Adding the 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 the 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 the graph obtained from G by adding the bottleneck edge, and let $t_{\mathcal{B}}$ be the stretch factor of $G_{\mathcal{B}}$.

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

The main result of this section is:

Theorem 3 *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 $t_{\mathcal{B}} < 3t_{\mathcal{P}}$.*

We end this section by giving a lower bound for the bottleneck approach.

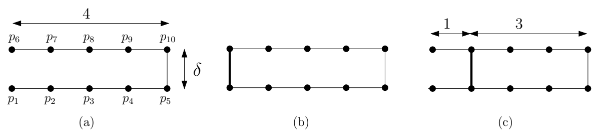

Figure 1: (a) The input graph G. (b) The bottleneck solution compared to (c) the optimal solution.

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

Proof. Consider the graph G, as in Fig. 1(a). More specifically, G is a graph with ten vertices $p_i = ((i-1) \bmod 5, \lfloor i/5 \rfloor \cdot \delta)$, $1 \leq i \leq 10$, and nine edges (p_5, p_{10}) and (p_j, p_{j+1}), for $1 \leq j \leq 4$ and $6 \leq j \leq 9$. If we assume that δ is a very small positive real value then (p_1, p_6) is the bottleneck in G and $t_{\mathcal{B}} = \frac{4+\delta}{\delta}$, see Fig. 1(b).

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

Hence, we have an upper bound of 3 and a lower bound of $(2 - \varepsilon)$ when adding the bottleneck edge to the input graph.

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 4.1, 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 4.2 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 (WSPD) defined by Callahan and Kosaraju (see [2]). They 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 (s is called the separation constant of the WSPD).

4.1 Linear number of candidate edges

In this section we show how to obtain a $(2 + \varepsilon)$-approximation in cubic time.

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 4–9, 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 dist. matrix of $G$.
2.  $\{(A_i, B_i)\}_{i=1}^{k} \leftarrow$WSPD of $V$ with $s = \frac{256}{\varepsilon^2}$.
3.  $t' \leftarrow \infty$.
4.  for $i \leftarrow 1$ to $k$
5.          Select a point $a_i \in A_i$ and a point $b_i \in B_i$.
6.          $G_i \leftarrow G = (V, E \cup (a_i, b_i))$.
7.          $t_i \leftarrow$StretchFactor$(G_i, M)$.
8.          if $t_i < t'$
9.             then $t' \leftarrow t_i$ and $e \leftarrow (a_i, b_i)$
10. return $G' = (V, E \cup \{e\})$.
```

Lemma 4 *Algorithm* ExpandGraph *requires $\mathcal{O}(n^3/\varepsilon^{2d})$ time and $\mathcal{O}(n^2)$ space.*

It remains to analyze the quality of the solution obtained from algorithm ExpandGraph. Let $\Delta(p, q)$ denote the set of point pairs in V such that $u, v \in V$ 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 $(2 + \lambda) \cdot t_{\mathcal{P}}$.*

Note that algorithm ExpandGraph might not test (p, q) stated in Lemma 5. 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 $q \in B_i$. 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 $\varepsilon \leq 1$.

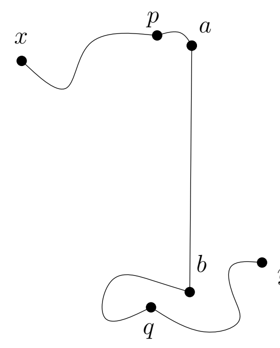

Figure 2: Illustrating the proof of Lemma 6.

Now we are ready to prove the claim. If for all pairs x, y, $(x, y) \notin \Delta(p, q)$ 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 2. It holds that:

$$d_G(a, p) = d_H(a, p) \quad \text{and} \quad d_G(b, q) = d_H(b, q). \quad (1)$$

This follows from the fact that the closest pair x', y' for which it holds that $(x', y') \in \Delta(p, q)$ has inter point distance at least $|x'y'| \geq \frac{\varepsilon}{8}|pq|$, according to Lemma 5. It holds that $|ap|$ and $|bq|$ are bounded by $\frac{2}{s}\ |pq| \leq \frac{\varepsilon^2}{128}\ |pq|$ which is less than $\frac{\varepsilon}{8}|pq|$ since $\varepsilon < 1$. As a consequence $(p, q) \notin \delta_H(a, p)$ and $(p, q) \notin \delta_H(b, q)$. Hence, claim (1) holds, which we will need below.

Next, we consider the length of the path in G' between x and y as illustrated in Fig. 2. Recall that x and y are two arbitrary points of V for which it holds that $(x, y) \in \Delta(p, q)$.

$$\begin{aligned}
d_{G'}(x, y) &\leq d_G(x, p) + d_G(p, a) + |ab| + d_G(b, q) \\
&\quad + d_G(q, y) \\
&\leq d_G(x, p) + d_H(p, a) + |ab| + d_H(b, q) \\
&\quad + d_G(q, y) \qquad \text{(from (7))} \\
&< d_G(x, p) + (1 + 4/s) \cdot |pq| + d_G(q, y) \\
&\quad + \frac{4t_H}{s} \cdot |pq| \qquad \text{(WSPD property)} \\
&\leq d_H(x, y) + \frac{64t_H}{\varepsilon s} \cdot |xy| \qquad \text{(Lemma 5)} \\
&= d_H(x, y) + \frac{\varepsilon}{4} \cdot t_H \cdot |xy|
\end{aligned}$$

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

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

$\Box$

We may now conclude this section with the following theorem.

Theorem 7 *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 $t' \leq (2+\varepsilon) \cdot t_{\mathcal{P}}$.*

4.2 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 the approximate stretch factor. The problem of computing the approximate stretch factor was considered by Narasimhan and Smid in [5]. They showed the following fact:

Fact 1 *([5]) 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.*

Their idea is to compute a well-separated pair decomposition of size $s = 4(1+\varepsilon)/\varepsilon$, 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 7 of the algorithm from $\mathcal{O}(n^2)$ to $\mathcal{O}(n/\varepsilon^d)$. There will be two changes in the ExpandGraph algorithm. First, between steps 2 and 3, the following four lines are inserted:

```
- ${(C_j, D_j)}_{j=1}^{\ell}$ ←WSPD of $V$ with $s' = 4(1+\varepsilon)/\varepsilon$.
- **for** $j$ ←1 **to** $\ell$
    Select a point $c_j \in C_j$ and a point $d_j \in D_j$.
- $\mathcal{S} = \{(c_1, d_1), \ldots, (c_\ell, d_\ell)\}$
```

Then, in step 7 of ExpandGraph we will instead of computing the exact stretch factor of G_i make a call to ApproximateStretchFactor, or ASF for short, with parameters G_i, (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$(G_i, e, M, \mathcal{S})$
**Input:** Euclidean graph $G(V, E)$, edge $e = (a, b) \in E$,
    distance matrix $M$ and a set of point pairs $\mathcal{S}$.
**Output:** A real value $\mathcal{D}_i$.
1.  $\mathcal{D}_i$ ←1
2.  **for** each point pair $(c_j, d_j)$ in $\mathcal{S}$
3.          dist ←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.          $\mathcal{D}_i$ ←max$\{\mathcal{D}_i$, dist$/|c_j d_j|\}$
5.  **return** $\mathcal{D}_i$.
```

We denote the modified algorithm ExpandGraph2.

Theorem 8 *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 $t' \leq (2+\epsilon) \cdot t_{\mathcal{P}}$.*

5 Open problems and Acknowledgements

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 8?
3. A natural extension is to allow more than one edge to be added. Can we generalize our results to this case?

The authors would like to thank René van Oostrum for fruitful discussions during the early stages of this work.

References

[1] 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.

[2] P. B. Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its applications.* Ph.D. thesis, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, 1995.

[3] 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.

[4] S. Langerman, P. Morin, and M. A. Soss. Computing the maximum detour and spanning ratio of planar paths trees, and cycles. In *Proc. STACS*, pages 250–261, 2002.

[5] G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM Journal of Computing*, 30(3):978–989, 2000.

[6] G. Navarro and R. Paredes. Practical construction of metric t-spanners. In *Proc. 5th Workshop on Algorithm Engineering and Experiments*, pages 69–81. SIAM Press, 2003.

[7] 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.