Fast Algorithms for Diameter-Optimally Augmenting Paths*

U. Große[1], J. Gudmundsson[2], C. Knauer[1], M. Smid[3], and F. Stehn[1]

[1] Institut für Angewandte Informatik, Universität Bayreuth, Bayreuth, Germany
[2] School of Information Technology, University of Sydney, Sydney, Australia
[3] School of Computer Science, Carleton University, Ottawa, Canada

Abstract. We consider the problem of augmenting a graph with n vertices embedded in a metric space, by inserting one additional edge in order to minimize the diameter of the resulting graph. We present an exact algorithm for the cases when the input graph is a path that runs in $O(n \log^3 n)$ time. We also present an algorithm that computes a $(1+\varepsilon)$-approximation in $O(n+1/\varepsilon^3)$ time for paths in $\mathbb{R}^d$, where d is a constant.

1 Introduction

Let $G = (V, E)$ be a graph in which each edge has a positive weight. The weight (or length) of a path is the sum of the weights of the edges on this path. For any two vertices x and y in V, we denote by $\delta_G(x, y)$ their shortest-path distance, i.e., the minimum weight of any path in G between x and y. The diameter of G is defined as $\max\{\delta_G(x, y) : x, y \in V\}$.

Assume that we are also given weights for the non-edges of the graph G. In the *Diameter-Optimal k-Augmentation Problem*, DOAP(k), we have to compute a set F of k edges in $(V \times V) \setminus E$ for which the diameter of the graph $(V, E \cup F)$ is minimum.

In this paper, we assume that the given graph is a path embedded in a metric space, and the weight of any edge and non-edge is equal to the distance between its vertices. We consider the case when $k = 1$; thus, we want to compute one non-edge which, when added to the graph, results in an augmented graph of minimum diameter. Surprisingly, no non-trivial results were known even for this restricted case.

Throughout the rest of the paper, we assume that $(V, |\cdot|)$ is a metric space, consisting of a set V of n elements (called points). The distance between any two points x and y is denoted by $|xy|$. We assume that an oracle is available that returns the distance between any pair of points in $O(1)$ time. Our contribution is as follows:

1. If G is a path, we solve problem DOAP(1) in $O(n \log^3 n)$ time.
2. If G is a path and the metric space is $\mathbb{R}^d$, where d is a constant, we compute a $(1+\varepsilon)$-approximation for DOAP(1) in $O(n + 1/\varepsilon^3)$ time.

* The research on this topic has been initiated during the *Korean Workshop on Computational Geometry 2014* (KW2014)

1.1 Related Work

The Diameter-Optimal k-Augmentation Problem for edge-weighted graphs, and many of its variants, have been shown to be NP-hard [16], or even $W[2]$-hard [9, 10]. Because of this, several special classes of graphs have been considered. Chung and Gary [5] and Alon et al. [1] considered paths and cycles with unit edge weights and gave upper and lower bounds on the diameter that can be achieved. Ishii [11] gave a constant factor approximation algorithm (approximating both k and the diameter) for the case when the input graph is outerplanar. Erdős et al. [7] investigated upper and lower bounds for the case when the augmented graph must be triangle-free.

The general problem: The Diameter-Optimal Augmentation Problem can be seen as a bicriteria optimization problem: In addition to the weight, each edge and non-edge has a cost associated with it. Then the two optimization criteria are (1) the total cost of the edges added to the graph and (2) the diameter of the augmented graph. We say that an algorithm is an (α, β)-approximation algorithm for the DOAP problem, with $\alpha, \beta \geq 1$, if it computes a set F of non-edges of total cost at most $\alpha \cdot B$ such that the diameter of $G' = (V, E \cup F)$ is at most $\beta \cdot D^B_{\text{opt}}$, where D^B_{opt} is the diameter of an optimal solution that augments the graph with edges of total cost at most B.

For the restricted version when all costs and all weights are identical [2, 4, 6, 12, 13], Bilò et al. [2] showed that, unless P=NP, there does not exist a $(c \log n, \delta < 1 + 1/D^B_{\text{opt}})$-approximation algorithm for DOAP if $D^B_{\text{opt}} \geq 2$. For the case in which $D^B_{\text{opt}} \geq 6$, they proved that, again unless P=NP, there does not exist a $(c \log n, \delta < \frac{5}{3} - \frac{7-(D^B_{\text{opt}}+1) \bmod 3}{3D^B_{\text{opt}}})$-approximation algorithm.

Li et al. [13] showed a $(1, 4+2/D^B_{\text{opt}})$-approximation algorithm. The analysis of the algorithm was later improved by Bilò et al. [2], who showed that it gives a $(1, 2+2/D^B_{\text{opt}})$-approximation. In the same paper they also gave an $(O(\log n), 1)$-approximation algorithm.

For general costs and weights, Dodis and Khanna [6] gave an $O(n \log D^B_{\text{opt}}, 1)$-approximation algorithm. Their result is based on a multi-commodity flow formulation of the problem. Frati et al. [9] recently considered the DOAP problem with arbitrary integer costs and weights. Their main result is a $(1, 4)$-approximation algorithm with running time $O((3^B B^3 + n + \log(Bn))Bn^2)$.

Geometric graphs: In the geometric setting, when the input is a geometric graph embedded in the Euclidean plane, there are very few results on graph augmentation in general. Rutter and Wolff [15] proved that the k-connectivity and k-edge-connectivity augmentation problems are NP-hard on plane geometric graphs, for $k = 2, 3, 4$, and 5; the problem is infeasible for $k \geq 6$ because every planar graph has a vertex of degree at most 5. Currently, there are no known approximation algorithms for this problem. Farshi et al. [8] gave approximation algorithms for the problem of adding one edge to a geometric graph while minimizing the dilation. There were several follow-up papers [14, 17], but there is still no non-trivial result known for the case when $k > 1$.

2 Augmenting a Path with One Edge

We are given a path $P = (p_1, \ldots, p_n)$ on n vertices in a metric space and assume that it is stored in an array $P[1, \ldots, n]$. To simplify notation, we associate a vertex with its index, that is $p_k = P[k]$ is also referred to as k for $1 \leq k \leq n$. This allows us to extend the total order of the indices to the vertex set of P. We denote the start vertex of P by s and the end vertex of P by e.

For $1 \leq k < l \leq n$, we denote the subpath $(p_k, \ldots, p_l)$ of P by $P[k, l]$, the cycle we get by adding the edge $\overline{p_k p_l}$ to $P[k, l]$ by $C[k, l]$, and the (unicyclic) graph we get by adding the edge $\overline{p_k p_l}$ as a *shortcut* to P by $\overline{P}[k, l]$; the length of $X \in \{P, P[k, l], C[k, l]\}$ is denoted by $|X|$. We will consider the functions $\overline{p}_{k,l} := \delta_{\overline{P}[k,l]}$ and $c_{k,l} := \delta_{C[k,l]}$, where δ_G is the length of the shortest path between two vertices in G. For $1 \leq k < l \leq n$, we let

$$M(k, l) := \max_{1 \leq x < y \leq n} \overline{p}_{k,l}(x, y)$$

denote the *diameter* of the graph $\overline{P}[k, l]$.

Our goal is to compute a shortcut $\overline{p_k p_l}$ for P that minimizes the diameter of the resulting unicyclic graph, i.e., we want to compute

$$m(P) := \min_{1 \leq k < l \leq n} M(k, l).$$

We will prove the following result:

Theorem 1. *Given a path P on n vertices in a metric space, we can compute $m(P)$, and a shortcut realizing that diameter, in $O(n \log^3 n)$ time.*

The algorithm consists of two parts. We first describe a sequential algorithm for the *decision problem*. Given P and a threshold parameter $\lambda > 0$, decide if $m(P) \leq \lambda$ (see Lemma 1 a) below). In a second step, we argue that the sequential algorithm can be implemented in a parallel fashion (see Lemma 1 b) below), thus enabling us to use the parametric search paradigm of Megiddo.

Lemma 1. *Given a path P on n vertices in a metric space and a real parameter $\lambda > 0$, we can decide in*

a) *$O(n \log n)$ time, or in*
b) *$O(\log n)$ parallel time using n processors*

whether $m(P) \leq \lambda$; the algorithms also produce a feasible shortcut if it exists.

To prove this lemma, observe that

$$m(P) \leq \lambda \text{ iff } \bigvee_{1 \leq k < l \leq n} M(k, l) \leq \lambda.$$

The algorithm checks, for each $1 \leq k < n$, whether there is some $k < l \leq n$ such that $M(k, l) \leq \lambda$. If one such index k is found, we know that $m(P) \leq \lambda$;

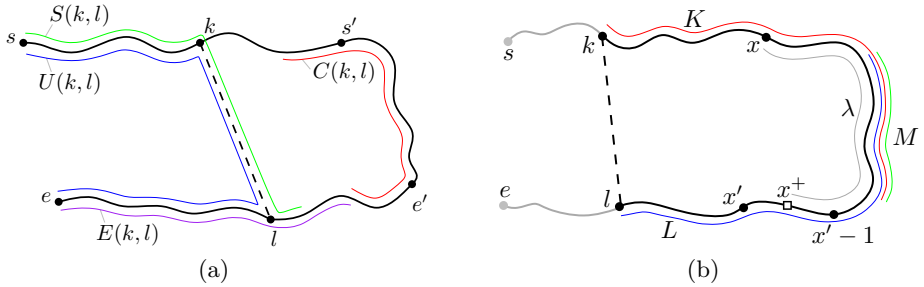

Fig. 1: **(a)** Illustration of the four distances that define the diameter of a shortcut $\overline{p_k p_l}$: $U(k,l)$ is the length of the shortest path connecting s and e; $O(k,l)$ is the length of the longest shortest path between any two points in $C[k,l]$; $S(k,l)$ ($E(k,l)$) is the length of the longest shortest path from s (e) to any vertex in $C(k,l)$. **(b)** Illustration of the computation of $O(k,l)$.

otherwise $m(P) > \lambda$. Clearly this approach also produces a feasible shortcut if it exists.

We decompose the function $M(k,l)$ into four monotone parts. This will facilitate our search for a feasible shortcut and enable us to do (essentially) binary search: For $1 \le k < l \le n$, we let

$$S(k,l) := \max_{k \le x \le l} \overline{p}_{k,l}(s,x), \qquad E(k,l) := \max_{k \le x \le l} \overline{p}_{k,l}(x,e),$$
$$U(k,l) := \overline{p}_{k,l}(s,e), \qquad O(k,l) := \max_{k \le x < y \le l} c_{k,l}(x,y).$$

Then we have $M(k,l) = \max\{S(k,l), E(k,l), U(k,l), O(k,l)\}$. The triangle inequality implies that

$$S(k,l) \le S(k,l+1), \qquad E(k,l) \ge E(k,l+1),$$
$$U(k,l) \ge U(k,l+1), \qquad O(k,l) \le O(k,l+1).$$

The function U is easy to evaluate once we have the array $D[1,\ldots,n]$ of the prefix-sums of the edge lengths: $D[i] := \sum_{1 \le j < i} |p_j p_{j+1}|$. These sums can be computed in $O(n)$ time sequentially or in $O(\log n)$ time using n processors. If in addition to D, the vertices $s' = \max\{v \mid \delta_P(s,v) \le \lambda\}$ and $e' = \min\{v \mid \delta_P(v,e) \le \lambda\}$ are computed for a fixed λ in $O(\log n)$ time (via binary search on D), the following decision problems can be answered in constant time:

$$S(k,l) \le \lambda, \quad E(k,l) \le \lambda, \quad U(k,l) \le \lambda.$$

We denote the maximum of these three functions by

$$N(k,l) = \max(S(k,l), E(k,l), U(k,l)).$$

Now clearly

$$M(k,l) = \max(N(k,l), O(k,l))$$

and, consequently

$$M(k,l) \leq \lambda \text{ iff } N(k,l) \leq \lambda \text{ and } O(k,l) \leq \lambda.$$

For fixed $1 \leq k < n$, the algorithm will first check whether there is some $k < l \leq n$ with $N(k,l) \leq \lambda$. If no such l exists, we can conclude that $M(k,l) > \lambda$ for all $k < l \leq n$. The monotonicity of S, E, and U implies that, for fixed $1 \leq k < n$, the set

$$N_k := \{k < l \leq n \mid N(k,l) \leq \lambda\}$$

is an *interval*. This interval can be computed (using binary search in P and in D as described above) in $O(\log n)$ time. If $N_k = \emptyset$ we can conclude that for the $1 \leq k < n$ under consideration and for all $k < l \leq n$, we have that $M(k,l) > \lambda$.

If N_k is non-empty, the monotonicity of O implies that it is sufficient to check for $l_k = \min N_k$ (i.e. the starting point of the interval) whether $O(k,l_k) \leq \lambda$:

$$\exists k < l \leq n : O(k,l) \leq \lambda \text{ iff } O(k,l_k) \leq \lambda.$$

Note that in this case we know that $N(k,l_k) \leq \lambda$.

Deciding the diameter of small cycles: We now describe how to decide for a given shortcut $1 \leq k < l \leq n$ if $O(k,l) \leq \lambda$, given that *we already know that* $N(k,l) \leq \lambda$. To this end, consider the following sets of vertices from $C[k,l]$: $K := \{k \leq x \leq l \mid \delta_P(k,x) \leq \lambda\}$, $L := \{k \leq x \leq l \mid \delta_P(x,l) \leq \lambda\}$, $M := K \cap L$, $K' := K \setminus L$, $L' := L \setminus K$.

These sets are intervals and can be computed in $O(\log n)$ time by binary search. Since $N(k,l) \leq \lambda$, we can conclude the following:

- the set of vertices of $C[k,l]$ is $K \cup L$
- $c_{k,l}(x,y) \leq \lambda$ for all $x, y \in K$
- $c_{k,l}(x,y) \leq \lambda$ for all $x, y \in L$
- $c_{k,l}(x,y) \leq \lambda$ for all $x \in M$, $y \in C[k,l]$

Consequently, if $c_{k,l}(x,y) > \lambda$ for $x, y \in C[k,l]$, we can conclude that $x \in K'$ and $y \in L'$. In order to establish that $O(k,l) \leq \lambda$, it therefore suffices to verify that

$$\bigwedge_{x \in K', y \in L'} c_{k,l}(x,y) \leq \lambda.$$

Note that on P any vertex x of K' is at least λ away from the vertex l, i.e., $\delta_P(x,l) > \lambda$. Let x^+ be point on (a vertex or an edge of) P that is closer (along P) by a distance of λ to l than to x, i.e., x^+ is the unique point on P such that

$$\delta_P(x^+,l) < \delta_P(x,l) \text{ and } \delta_P(x,x^+) = \lambda.$$

The next (in the direction of l) *vertex* of P will be denoted by x', i.e., $x < x' \leq l$ is the unique vertex of P such that

$$\delta_P(x,x'-1) \leq \lambda \text{ and } \delta_P(x,x') > \lambda.$$

Since x is a vertex of K', x' is a vertex of L'. For the following discussion we denote the distance achieved in $C[k,l]$ by using the shortcut by $c^+_{k,l}$ and the distance achieved by travelling along P only by $c^-_{k,l}$, i.e.,

$$c^-_{k,l}(x,y) := \delta_P(x,y) \text{ and } c^+_{k,l}(x,y) := \delta_P(x,k) + |\overline{p_k p_l}| + \delta_P(l,y).$$

Clearly

$$c_{k,l}(x,y) = \min(c^+_{k,l}(x,y), c^-_{k,l}(x,y)), \text{ and } |C[k,l]| = c^+_{k,l}(x,y) + c^-_{k,l}(x,y).$$

For every vertex $y < x'$ on L' we have that $c_{k,l}(x,y) \leq c^-_{k,l}(x,y) \leq \lambda$, so if there is some vertex $x' \neq y \in L'$ such that $c_{k,l}(x,y) > \lambda$, we know that $x' < y \leq l$; in that case we have that $c^+_{k,l}(x,y) \leq c^+_{k,l}(x,x')$. Since we assume that $c_{k,l}(x,y) > \lambda$, we also know that $c^+_{k,l}(x,y) > \lambda$ and we can conclude that $c^+_{k,l}(x,x') > \lambda$, and consequently that $c_{k,l}(x,x') > \lambda$, i.e., for all $x \in K'$ we have that

$$\bigwedge_{y \in L'} c_{k,l}(x,y) \leq \lambda \text{ iff } c_{k,l}(x,x') \leq \lambda.$$

The distance between (the point) x^+ and (the vertex) x' on P is called the *defect* of x and is denoted by $\Delta(x)$, i.e., $\Delta(x) = \delta_P(x^+, x')$.

Lemma 2.

$$c_{k,l}(x,x') \leq \lambda \text{ } \textit{iff} \text{ } |C[k,l]| \leq \Delta(x) + 2\lambda$$

Proof. Observe that

$$\begin{aligned} |C[k,l]| &= \delta_P(x,k) + |\overline{p_k p_l}| + \delta_P(l,x') + \delta_P(x',x^+) + \delta_P(x^+,x) \\ &= \delta_P(x,k) + |\overline{p_k p_l}| + \delta_P(l,x') + \Delta(x) + \lambda \\ &= c^+_{k,l}(x,x') + \Delta(x) + \lambda. \end{aligned}$$

Since $c^-_{k,l}(x,x') > \lambda$, we have that $c_{k,l}(x,x') \leq \lambda$ iff $c^+_{k,l}(x,x') \leq \lambda$; the claim follows. □

To summarize the above discussion we have the following chain of equivalences (here $\Delta_{k,l} := |C[k,l]| - 2\lambda$):

$$O(k,l) \leq \lambda \Leftrightarrow \bigwedge_{x \in K'} c_{k,l}(x,x') \leq \lambda \Leftrightarrow \bigwedge_{x \in K'} \Delta_{k,l} \leq \Delta(x) \Leftrightarrow \min_{x \in K'} \Delta(x) \geq \Delta_{k,l}.$$

Since K' is an interval, the last condition can be tested easily after some preprocessing: To this end we compute a $1d$-range tree on D and associate with each vertex in the tree the minimum Δ-value of the corresponding canonical subset. For every *vertex* x of P that is at least λ away from the end vertex of P we can compute $\Delta(x)$ in $O(\log n)$ time by binary search in D. With these values the range tree can be built in $O(n)$ time. A query for an interval K' then gives us

Algorithm 1: Algorithm for deciding if $m(P) \leq \lambda$

```
DecisionAlgorithm(P, λ) ;                          // Decide if m(P) ≤ λ
1 begin
    global D ← ComputePrefixSums(P);
    global s' ← max{v | δ_P(s, v) ≤ λ};
    global e' ← min{v | δ_P(v, e) ≤ λ};
    global T ← ComputeRangeTree(P, λ);
    for 1 ≤ k < n do
        N_k ← ComputeFeasibleIntervalForN(k, λ);
        if N_k ≠ ∅ and CheckOForShortcut(k, min(N_k), λ) then
            return True
    return False
  end
  CheckOForShortcut(k, l, λ) ;                     // Decide if O(k, l) ≤ λ
2 begin
    K' ← {k ≤ x ≤ l | δ_P(k, x) ≤ λ ∧ δ_P(x, l) > λ};  // Compute the interval
    by binary search
    μ ← min_{x∈K'} Δ(x) ;                          // Query the range tree T
    return (μ ≥ |C[k, l]| − 2λ)
  end
```

$\mu := \min_{x \in K'} \Delta(x)$ in $O(\log n)$ time and we can check the above condition in $O(1)$ time.

We describe the algorithm in pseudocode; see Algorithm 1.

The correctness of the algorithm follows from the previous discussion. ComputePrefixSums runs in $O(n)$ time, ComputeRangeTree runs in $O(n \log n)$ time, ComputeFeasibleIntervalForN runs in $O(\log n)$ time, a call to CheckOForShortcut requires $O(\log n)$ time. The total runtime is therefore $O(n \log n)$. It is easy to see that with n processors, the steps ComputePrefixSums and ComputeRangeTree can be realized in $O(\log n)$ parallel time and that with this number of processors, all calls to CheckOForShortcut can be handled in parallel. Therefore, the entire algorithm can be parallelized and has a parallel runtime of $O(\log n)$, as stated in Lemma 1 b). This concludes the proof of Lemma 1.

When we plug this result into the parametric search technique of Megiddo, we get the algorithm for the optimization problem as claimed in Theorem 1.

From the above discussion, we note that, since there are only four possible distances to compute to determine the diameter of a path augmented with one shortcut edge, the following corollary follows immediately.

Corollary 1. *Given a path P on n vertices in a metric space and a shortcut (u, v), the diameter of $P \cup (u, v)$ can be computed in $O(n)$ time.*

3 An Approximation Algorithm in Euclidean Space

In Section 2, we presented an $O(n \log^3 n)$-time algorithm for the problem when the input graph is a path in a metric space. Here we show a simple $(1+\varepsilon)$-approximation algorithm with running time $O(n + 1/\varepsilon^3)$ for the case when the input graph is a path in $\mathbb{R}^d$, where d is a constant. The algorithm will use two ideas: clustering and the well-separated pair decomposition (WSPD) as introduced by Callahan and Kosaraju [3].

Definition 1 ([3]). *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 A, (i) C_B contains B, and (ii) 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 ([3]). *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) $|xy| \leq (1 + 4/s) \cdot |pq|$, and (ii) $|px| \leq (2/s) \cdot |pq|$.*

Definition 2 ([3]). *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 $m = \mathcal{O}(s^d n)$ can be computed in $\mathcal{O}(s^d n + n \log n)$ time.

Algorithm We are given a polygonal path P on n vertices in $\mathbb{R}^d$. We assume without loss of generality that the total length of P is 1. Partition P into $m = 1/\varepsilon_1$ subpaths $P_1, \ldots, P_m$, each of length ε_1, for some constant $0 < \varepsilon_1 < 1$ to be defined later. Note that a subpath may have one (or both) endpoint in the interior of an edge. For each subpath P_i, $1 \leq i \leq m$, select an arbitrary vertex r_i along P_i as a representative vertex, if it exists. The set of representative vertices is denoted R_P; note that the size of this set is at most $m = 1/\varepsilon_1$. Let $P(R)$ be the path consisting of the vertices of R_P, in the order in which they appear along the path P. We give each edge (u, v) of $P(R)$ a weight equal to $\delta_P(u, v)$. the interior of an edge of P, then $\delta_P(u, v)$ is defined in the natural way.)

Imagine that we "straighten" the path $P(R)$, so that it is contained on a line. In this way, the vertices of this path form a point set in $\mathbb{R}^1$; we compute a well-separated pair decomposition $\mathcal{W}$ for the one-dimensional set R_P, with separation constant $1/\varepsilon_2$, with $0 < \varepsilon_2 < 1/4$ to be defined later. Then, we go through all pairs $\{A, B\}$ in $\mathcal{W}$ and compute the diameter of $P(R) \cup \{(rep(A), rep(B)\}$, where $rep(A)$ and $rep(B)$ are representative points of A and B, respectively, which are arbitrarily chosen from their sets. Note that the number of pairs in $\mathcal{W}$ is $O(1/\varepsilon_1\varepsilon_2)$. Finally the algorithm outputs the best shortcut.

Analysis We first discuss the running time and then turn our attention to the approximation factor of the algorithm.

The clustering takes $O(n)$ time, and constructing the WSPD of R_P takes $O(\frac{1}{\varepsilon_1\varepsilon_2} + \frac{1}{\varepsilon_1}\log\frac{1}{\varepsilon_1})$ time. For each of the $O(1/\varepsilon_1\varepsilon_2)$ well-separated pairs in $\mathcal{W}$, computing the diameter takes, by Corollary 1, time linear in the size of the uni-cyclic graph, that is, $O(\frac{1}{\varepsilon_1^2\varepsilon_2})$ time in total.

Lemma 4. *The running time of the algorithm is* $O(n + \frac{1}{\varepsilon_1^2\varepsilon_2})$.

Before we consider the approximation bound, we need to define some notation. Consider any vertex p in P. Let $r(p)$ denote the representative vertex of the subpath of P containing p. For any two vertices p and q in P, let $\{A, B\}$ be the well-separated pair such that $r(p) \in A$ and $r(q) \in B$. The representative points of A and B will be denoted $w(p)$ and $w(q)$, respectively.

Lemma 5. *For any shortcut* $e = (p, q)$ *and for any two vertices* $x, y \in P$, *we have*

$$(1 - 4\varepsilon_2) \cdot \delta_G(x, y) - 6\varepsilon_1 \leq \delta_H(w(x), w(y)) \leq (\frac{1}{1 - 4\varepsilon_2}) \cdot \delta_G(x, y) + 6\varepsilon_1,$$

where $G = P \cup (p, q)$ *and* $H = P(R) \cup (w(p), w(q))$.

Proof. We only prove the second inequality, because the proof of the first inequality is almost identical.

Consider two arbitrary vertices x, y in P, and consider a shortest path in G between x and y. We have two cases:
Case 1: If $\delta_G(x, y) = \delta_P(x, y)$, then $\delta_H(r(x), r(y)) \leq \delta_P(x, y) + 2\varepsilon_1$.
Case 2: If $\delta_G(x, y) < \delta_P(x, y)$, then the shortest path in G between x and y must traverse (p, q). Assume that the path is $x \rightsquigarrow p \to q \rightsquigarrow t$, thus $\delta_G(x, y) = \delta_P(x, p) + |pq| + \delta_P(q, y)$. Consider the following three observations:

(1) $|pq| \geq |r(p)r(q)| - 2\varepsilon_1$ and $|w(p)w(q)| \leq (1 + 4\varepsilon_2) \cdot |r(p)r(q)|$. Consequently, $|w(p)w(q)| \leq (1 + 4\varepsilon_2) \cdot (|pq| + 2\varepsilon_1)$.

(2) We have

$$\begin{aligned}\delta_P(x,p) &\geq \delta_P(w(x),w(p)) - \delta_P(w(x),x) - \delta_P(w(p),p)\\ &\geq \delta_P(w(x),w(p)) - (\varepsilon_1 + \delta_P(w(x),r(x))) - (\varepsilon_1 + \delta_P(w(y),r(y)))\\ &\geq \delta_P(w(x),w(p)) - (\varepsilon_1 + 2\varepsilon_2\delta_P(w(x),w(p))) - (\varepsilon_1 + 2\varepsilon_2\delta_P(w(x),w(p)))\\ &= (1-4\varepsilon_2)\cdot\delta_P(w(x),w(p)) - 2\varepsilon_1\\ &\geq (1-4\varepsilon_2)\cdot\delta_H(w(x),w(p)) - 2\varepsilon_1\end{aligned}$$

That is, $\delta_H(w(x),w(p)) \leq \frac{1}{1-4\varepsilon_2}\cdot\delta_P(x,p) + 2\varepsilon_1$.

(3) We have, $\delta_H(w(y),w(q)) \leq \frac{1}{1-4\varepsilon_2}\cdot\delta_P(y,q) + 2\varepsilon_1$, following the same arguments as in (2).

Putting together the three observations we get:

$$\begin{aligned}\delta_H(w(x),w(y)) &\leq \delta_H(w(x),w(p)) + |w(p)w(q)| + \delta_H(w(q),w(y))\\ &\leq (\frac{1}{1-4\varepsilon_2})\cdot\delta_P(x,p) + 2\varepsilon_1) + ((1+4\varepsilon_2)\cdot(|pq| + 2\varepsilon_1))\\ &\quad +(\frac{1}{1-4\varepsilon_2})\cdot\delta_P(y,q) + 2\varepsilon_1)\\ &< (\frac{1}{1-4\varepsilon_2})\cdot\delta_G(x,y) + 6\varepsilon_1,\end{aligned}$$

where the last inequality follows from the fact that $0 < \varepsilon_2 < 1/4$. This concludes the proof of the lemma. □

By setting $\varepsilon_1 = \varepsilon/60$ and $\varepsilon_2 = \varepsilon/32$ and using the fact that the diameter of H is at least $1/2$, we obtain the following theorem that summarizes this section.

Theorem 2. *Given a path P with n vertices in $\mathbb{R}^d$ and a real number $\varepsilon > 0$, we can compute a shortcut to P in $O(n + 1/\varepsilon^3)$ time such that the resulting uni-cyclic graph has diameter at most $(1+\varepsilon)\cdot d_{\text{opt}}$, where d_{opt} is the diameter of an optimal solution.*

References

1. N. Alon, A. Gyárfás, and M. Ruszinkó. Decreasing the diameter of bounded degree graphs. *Journal of Graph Theory*, 35:161–172, 1999.
2. D. Bilò, L. Gualà, and G. Proietti. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theoretical Computer Science*, 417:12–22, 2012.
3. 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.
4. V. Chepoi and Y. Vaxès. Augmenting trees to meet biconnectivity and diameter constraints. *Algorithmica*, 33(2):243–262, 2002.

5. F. R. K. Chung and M. R. Garey. Diameter bounds for altered graphs. *Journal of Graph Theory*, 8(4):511–534, 1984.
6. Y. Dodis and S. Khanna. Designing networks with bounded pairwise distance. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 750–759, 1999.
7. P. Erdős, A. Gyárfás, and M. Ruszinkó. How to decrease the diameter of triangle-free graphs. *Combinatorica*, 18(4):493–501, 1998.
8. M. Farshi, P. Giannopoulos, and J. Gudmundsson. Improving the stretch factor of a geometric network by edge augmentation. *SIAM Journal on Computing*, 38(1):226–240, 2005.
9. F. Frati, S. Gaspers, J. Gudmundsson, and L. Mathieson. Augmenting graphs to minimize the diameter. *Algorithmica*, pages 1–16, 2014.
10. Y. Gao, D. R. Hare, and J. Nastos. The parametric complexity of graph diameter augmentation. *Discrete Applied Mathematics*, 161(10–11):1626–1631, 2013.
11. T. Ishii. Augmenting outerplanar graphs to meet diameter requirements. *Journal of Graph Theory*, 74:392–416, 2013.
12. S. Kapoor and M. Sarwat. Bounded-diameter minimum-cost graph problems. *Theory of Computing Systems*, 41(4):779–794, 2007.
13. C.-L. Li, S. T. McCormick, and D. Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. *Operations Research Letters*, 11(5):303–308, 1992.
14. J. Luo and C. Wulff-Nilsen. Computing best and worst shortcuts of graphs embedded in metric spaces. In *19th International Symposium on Algorithms and Computation*, Lecture Notes in Computer Science. Springer, 2008.
15. I. Rutter and A.Wolff. Augmenting the connectivity of planar and geometric graphs. *Journal of Graph Algorithms and Applications*, 16(2):599–628, 2012.
16. A. A. Schoone, H. L. Bodlaender, and J. van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11:409–427, 1997.
17. C. Wulff-Nilsen. Computing the dilation of edge-augmented graphs in metric spaces. *Computational Geometry - Theory and Applications*, 43(2):68–72, 2010.