Approximate Distance Queries for Weighted Polyhedral Surfaces

Hristo N. Djidjev[1] and Christian Sommer[2]

[1] Los Alamos National Laboratory, Los Alamos, NM 87545
[2] Massachusetts Institute of Technology, Cambridge, MA 02139

Abstract. Let P be a planar polyhedral surface consisting of n triangular faces, each assigned with a positive weight. The weight of a path p on P is defined as the weighted sum of the Euclidean lengths of the portions of p in each face multiplied by the corresponding face weights. We show that, for every $\varepsilon \in (0,1)$, there exists a data structure, termed *distance oracle*, computable in time $O(n\varepsilon^{-2}\log^3(n/\varepsilon)\log^2(1/\varepsilon))$ and of size $O(n\varepsilon^{-3/2}\log^2(n/\varepsilon)\log(1/\varepsilon))$, such that $(1+\varepsilon)$–approximate distance queries in P can be answered in time $O(\varepsilon^{-1}\log(1/\varepsilon)+\log\log n)$. As in previous work (Aleksandrov, Maheshwari, and Sack (*J. ACM 2005*) and others), the big–O notation hides constants depending logarithmically on the ratio of the largest and smallest face weights and reciprocally on the sine of the smallest angle of P. The tradeoff between space and query time of our distance oracle is a significant improvement in terms of n over the previous best tradeoff obtained by a distance oracle of Aleksandrov, Djidjev, Guo, Maheshwari, Nussbaum, and Sack (*Discrete Comput. Geom. 2010*), which requires space roughly quadratic in n for a comparable query time.

1 Introduction

We design an efficient algorithm and a data structure to answer approximate distance queries between points on a weighted planar polyhedral surface. The problems of computing shortest paths arise in numerous application areas and have consistently been among the most active research topics in theoretical computer science.

In many applications, multiple shortest paths have to be computed for the same domain between different pairs of points without knowing the sequence of pairs in advance. In this version of the problem, an appropriate data structure can be computed in a *preprocessing* phase of the algorithm, and a *query* algorithm may utilize the data structure in order to efficiently answer shortest-path or distance queries.

Algorithms for the shortest-path query problem have been developed for planar graphs [6,4,11] and, in general, for graphs with small separators [5]. Algorithms with better space/query-time tradeoffs and faster query times are possible if *approximate* distances and shortest paths are acceptable. A *stretch–α distance oracle* is a data structure that allows computing a distance estimate that does not exceed α times the weight of a shortest path [14]. For planar graphs, Thorup [13] and Klein [8] construct, for any $\epsilon > 0$, a $(1+\epsilon)$–stretch oracle of size $O(n\log(n)/\epsilon)$ in $O(n\log^3(n)/\epsilon^2)$ time that answers distance queries in $O(1/\epsilon)$ time. Their work has been extended to graphs with bounded genus [7] and to minor-free graphs [1].

Modeling a real-world problem as a shortest-path problem often requires the use of non-Euclidean distances. Such types of distances have been intensively studied in recent years and they are the focus of this work. We consider the *weighted region distance* introduced by Mitchell and Papadimitriou [10], where the geometric region is divided into triangles or tetrahedra each assigned an individual weight. The weight or length of a path p is then defined as the weighted sum of the portions of p in each triangle (tetrahedron) multiplied by the corresponding weight. Such distance measures are appropriate, for instance, when computing a route through a terrain with certain terrain properties such as terrain type (e.g. water, sand, or rock), slope, and obstacles, encoded as weights. Another potential application is seismology, where seismic waves follow shortest paths, and the speed of a wave in each layer depends on the type and density of the rock.

Compared to the Euclidean case, computing a shortest path in a weighted region is a more involved problem. While in the interior of each triangle each connected portion of a shortest path is a straight-line segment; when crossing a boundary between two triangles the path locally satisfies *Snell's Law* $w^-\sin(\varphi^-) = w^+\sin(\varphi^+)$, where w^- and w^+ are the weights of the triangles before and after the boundary and φ^- and φ^+ are the corresponding acute angles between the portions of the paths in the corresponding triangles and the normal to the boundary, respectively. It is believed that an exact algorithm for finding a shortest path between two arbitrary points does not exist for arbitrary values of n.

Accordingly, several algorithms for finding approximate shortest paths have been developed. Mitchell and Papadimitriou [10] first studied the problem and developed an algorithm for computing a $(1+\varepsilon)$–approximate shortest path between a pair of nodes in $O(n^8\log(n/\varepsilon))$ time and using $O(n^4)$ space. Their result was subsequently improved in several papers, culminating in the algorithm by Aleksandrov, Maheshwari, and Sack [3] with $O(\frac{n}{\sqrt{\varepsilon}}\log(n/\varepsilon)\log(1/\varepsilon))$ time and $O(n)$ space. The big-O notation hides further dependencies on the maximum and minimum weight ratio and on the angles of the triangulation.

In this paper we study the problem of constructing a distance oracle for answering shortest-path queries between pairs of points on a weighted polyhedral surface. This query version of the shortest-path problem was studied previously by Aleksandrov, Djidjev, Guo, Maheshwari, Nussbaum, and Sack [2]. They prove the following.

Theorem 1 (Aleksandrov et al. [2, Theorem 7]). *Let P be a weighted polyhedral surface of genus g consisting of n triangular faces. Let $\varepsilon \in (0,1)$ and $q \in (\varepsilon^{-1/2}\log^2(1/\varepsilon), \varepsilon^{-1/2}(g+1)^{2/3}n^{1/3})$. There exists a data structure for* $(1+$

ε)–approximate point-to-point distance queries with space $O(\frac{(g+1)n^2}{\varepsilon^{3/2}q}\log^4(1/\varepsilon))$, preprocessing time $O(\frac{(g+1)n^2}{\varepsilon^{3/2}q}\log(n/\varepsilon)\log^4(1/\varepsilon))$, and query time $O(q)$.

The approach of [2] is the following. First, the domain is discretized (see also [3] and Section 2.2) by defining a set of suitably spaced points (called *Steiner points*) on the bisectors of each triangle. Second, a graph is defined with nodes being the Steiner points and edges being added between any pair of nodes whose corresponding Steiner points belong to bisectors of the same or an adjacent triangle. The weight on the edges is equal to the locally computed distance between the corresponding Steiner points. The resulting graph G_ε has $O((n/\sqrt{\varepsilon})\log(1/\varepsilon))$ nodes and $O((n/\varepsilon)\log^2(1/\varepsilon))$ edges.

It is shown that any shortest path in the region between two Steiner points can be approximated by a path in G_ε between the same points with weight at most $1+\varepsilon$ times larger.

In this paper we combine the discretization methodology of Aleksandrov et al. [2,3] with novel algorithms for preprocessing G_ε and querying the resulting data structure. We prove the following result.

Theorem 2. *Let P be a planar polyhedral surface consisting of n triangular faces with a positive weight assigned to each of them. Let $\epsilon \in (0,1)$. There exists a data structure to answer $(1+\epsilon)$–approximate point-to-point distance queries in P with $O(n\epsilon^{-3/2}\log^2(n/\epsilon)\log^2(1/\epsilon))$ space, $O(n\epsilon^{-2}\log^3(n/\epsilon)\log^2(1/\epsilon))$ preprocessing time, and $O(\epsilon^{-1}\log(1/\epsilon)+\log\log n)$ query time.*

The constants in the big-O bounds depend on the geometry of P in the same way as in [3] and [2]. The performance gain (compared to Aleksandrov et al. [2]) has three main reasons: *i)* we only *approximate* distances in G_ε (without any consequences to the quality of the final approximation), *ii)* we use ε–covers on shortest-path separators (as defined in [13], using the scaled version to improve the query time), and *iii)* we work on an implicit representation of a planar version of G_ε to keep the dependency on n almost linear and the dependency on $1/\varepsilon$ quadratic in the preprocessing time, less than quadratic in the space requirements, and linear in the query time.

2 Preliminaries

2.1 Definitions

For the sake of brevity, we assume some familiarity with previous work by Aleksandrov et al. [3,2]. We only include (extracted from [2]) the most important definitions. Let P be a planar polyhedral surface in 3–dimensional Euclidean space consisting of n triangular faces $f_1,\ldots,f_n$. Each face f_i has an associated positive weight w_i, representing the cost of traveling a unit Euclidean distance inside f_i. The cost of traveling along an edge is the minimum of the weights of the triangles incident to that edge. Edges are assumed to be part of the triangle they inherit their weight from. The cost of a path π in P is defined as

$||\pi|| = \sum_{i=1}^{n} w_i\, |\pi_i|$, where $|\pi_i|$ denotes the Euclidean length of the portion π_i of π in f_i. Path lengths in graphs are denoted by $\mathsf{len}(\cdot)$, distances are denoted by $\mathsf{d}(\cdot, \cdot)$.

2.2 Domain Discretization

The approach of [3] is, given a node s of P and a parameter ε, to first discretize the polyhedral surface P to obtain a graph G_ε. The shortest-path lengths from s in G_ε are $(1+\varepsilon)$–approximations for the corresponding distances in P [3, Theorem 3.2]. The discretization of P is constructed as follows.

For each triangle, we compute the bisectors for all its angles. Let v denote a node of P, let α denote one of its angles, and let ℓ denote the corresponding bisector. On ℓ, we add Steiner points $p_0, \ldots p_k$ as follows. p_0 is at distance $r(v) := \varepsilon \frac{w_{\min}(v)}{7 w_{\max}(v)} \delta(v)$, where $\delta(v)$ is the minimum Euclidean distance from v to the set of edges incident to triangles around v but *not* incident to v, and $w_{\min}(v)$ and $w_{\max}(v)$ are the minimum and the maximum weight of triangles incident to v, respectively. The remaining Steiner points p_i are chosen as a geometric progression such that $|p_{i-1}p_i| = \sin(\alpha/2)\sqrt{\varepsilon/2}\,|vp_{i-1}|$ for $i = 1, \ldots, k$. Using these Steiner points, we can compute $(1+\varepsilon/2)$–approximate distances between any two nodes on the boundary of a triangle.

The number of Steiner points on bisector ℓ of angle α at node v is at most $C(\ell)\varepsilon^{-1/2}\log_2(2/\varepsilon)$, where $C(\ell) < \frac{1.61}{\sin\alpha}\log_2(2\,|\ell|\,/r(v))$ [3, Lemma 2.3]. Following [3,12,2], we assume that $C(\ell)$ is bounded by a constant.

We construct a graph G_ε on $\Theta(n\varepsilon^{-1/2}\log(1/\varepsilon))$ nodes, wherein each node corresponds to either an original node v or to a Steiner point. The edges of G_ε are chosen such that nodes corresponding to Steiner points on neighboring bisectors are connected. Two bisectors are called *neighbors* if the corresponding triangles share at least one edge. Each bisector has nine neighboring bisectors[3] (three within its own triangle and two in each of the adjacent triangles); consequently, the number of edges in G_ε is $\Theta(n\varepsilon^{-1}\log^2(1/\varepsilon))$. The edges have weights defined as the distance in P restricted to the two triangles the corresponding bisectors lie in.

For each shortest path p in P, let $\pi = \pi(p)$ denote the approximating path in G_ε constructed by the algorithm from [3]. We also use the inverse mapping defined by $\pi^{-1}(\pi) = p$. By [3], p and π have the same source and target. We also use the following property that follows from the construction in [3].

Lemma 1. *Let p_1 and p_2 be two non-intersecting shortest paths in P. Then no pair of segments of the paths $\pi(p_1)$ and $\pi(p_2)$ intersect.*

[3] Here we use the version of Aleksandrov et al. [2, Section 3.1], wherein each bisector is defined to be a neighbor of all the nine bisectors in adjacent triangles, as opposed to the six bisectors sharing an edge as in [3]. Note that a bisector is a neighbor of itself.

2.3 Approximate Distance Oracles for Planar Graphs

Our preprocessing and query algorithms build on techniques that have been used previously to construct approximate distance oracles for planar graphs [9,13].

The first technique is to *approximately represent shortest paths that intersect a shortest path* [9, Lemma 4]. Let Q be a shortest path of length $O(\delta)$ in a graph G. For any $\epsilon > 0$ there exists a set of nodes $C_\epsilon(Q) \subseteq V(Q)$ (or simply $C(Q)$ if ϵ is clear from the context) termed *cover* of size $O(1/\epsilon)$ such that for those pairs of nodes (u, v) at distance $\delta \leq \mathsf{d}(u, v) \leq 2\delta$ for which all the shortest paths between u and v intersect Q, there is a node q termed *portal* in the cover $q \in C_\epsilon(Q)$ such that

$$\mathsf{d}(u, v) \leq \mathsf{d}(u, q) + \mathsf{d}(q, v) \leq (1 + \epsilon)\mathsf{d}(u, v). \tag{1}$$

The distance oracle involves storing with each node v the portals that cover v with respect to several shortest paths (and the distances associated with these portals).

The second technique is to *recursively separate a planar graph by shortest paths*. Given a triangulated planar graph on N nodes and a rooted spanning tree, Thorup [13] demonstrates how to find a triangle such that the paths from the root of the tree to the corners of the triangle separate the graph into at least two disconnected subgraphs of size at most $2N/3$. The recursive application of this separator theorem yields components that are separated from each other by a constant number of shortest paths.

Note that we cannot directly use the algorithms from [13] due to two main differences between G_ϵ and the graphs considered in [13]: *i)* our graph is *not* planar and *ii)* our weights are not integral but real. Furthermore, a direct adaptation of the algorithms in [13] would result in a rather high dependency on ε, which is considered undesirable in this line of work [3]. We adapt Thorup's algorithms for our scenario with main improvements with respect to the parameter ε, both in the preprocessing and query times.

3 Discretization, Pseudo-Planarization, and Separator Decomposition

3.1 Discretizing the weights

We replace the original weights of G_ε by integral weights in $\{0, 1, \ldots, N\}$, in order to apply the techniques in [13] on G_ε. We determine the value of N and the mapping from the weights of G_ε to $\{0, 1, \ldots, N\}$.

For any $\delta > 0$ consider the mapping i_q defined by the formula $i_q(w) = \lceil \frac{w}{\varepsilon q} \rceil$, where $q = \frac{\varepsilon\delta}{n \log^2(1/\varepsilon)}$. Define $N = N(\delta, q, \varepsilon) = i_q(\delta) = \lceil \frac{\delta}{\varepsilon q} \rceil = O\left(\frac{n \log^2(1/\varepsilon)}{\varepsilon^2}\right)$. Let $\mathsf{w}(e)$ denote the weight of any edge e of G_ε. We have the following.

Lemma 2. *Let G'_ε be the graph with the same nodes and edges as G_ε and weight on each edge e set to $i_q(\mathsf{w}(e))$. Then, for each shortest path p in G'_ε with weight* *at most δ between a pair s, t of nodes, there is a path between s and t in G'_ε whose weight in G_ε is within an additive factor of $O(\varepsilon\delta)$ of the weight of p. The largest edge weight of G'_ε does not exceed N.*

3.2 Pseudo-planarization

One of the main ingredients in Thorup's distance oracle is a *shortest-path separator* that consists of three shortest paths separating a planar graph into at least two subgraphs of weight at most a third of the size of the original graph. In our case, G_ε is not planar. A set of three shortest paths generally does *not* separate the graph into two edge-disjoint components as each edge of G_ε is intersected (in a geometric sense) by roughly $1/\sqrt{\varepsilon}$ edges. In our construction, we concurrently compute separators and pseudo-planarize the graph. Whenever edges geometrically intersect with a shortest-path separator, we split the edge and we add a node to represent this intersection on the separator path.

Note that we cannot use the separator construction algorithm from [13] directly, since our graph is neither planar nor triangulated. The surface P, however, is triangulated, and we use that triangulation for the purpose of the separator construction. We also cannot "planarize" G_ε by adding a node for every intersection of two line segments, since the dependency on ε would increase. While the polynomial dependency on n is of primary importance, in this line of work [3, Table 1], the low dependency on ε^{-1} is also considered relevant.

Construction of $\hat{G}_\varepsilon$ We initialize $\hat{G}_\varepsilon$ as G_ε. For each triangle bisector ℓ, we sort the nodes on ℓ by their distance from the corresponding node of P and for each node on ℓ we store its position. We need these positions to count the number of nodes on the left (right) of a separator path.

Let r be the node in G_ε corresponding to an arbitrary node of a triangle in P. We compute a single-source shortest path tree in G_ε rooted at r. Let T denote that tree and for any node $u \in V(G_\varepsilon)$, let $T(u)$ denote the path from r to u in T. We start with an arbitrary triangle A in P (as opposed to [13], where triangles correspond to faces of the plane graph). Let x, y, z denote the nodes that correspond to the nodes of A, respectively. We remove all edges incident to nodes or intersected by edges of $S(A) = T(x) \cup T(y) \cup T(z)$, as well as all edges intersected by any edge of the triangle A. We say that $S(A)$ defines a *balanced* separator if no component of the resulting graph contains more than two thirds of the nodes of the original graph. If $S(A)$ defines a balanced separator, we recurse on each subgraph. Otherwise, we "flip" one node (wlog we flip z) to obtain a neighboring triangle A' such that $T(x) \cup T(y) \cup T(z')$ is a separator with better balance. Once we have found a good separator (corresponding to a triangle in P; let x'', y'', z'' denote the nodes corresponding to its endpoints), we compute, for each edge $e \in T(x'') \cup T(y'') \cup T(z'')$, all the geometric intersections with edges $e' \in E(G_\varepsilon)$. For each intersection a node is added to $V(\hat{G}_\varepsilon)$ and for each such edge e' we add its two parts e'_1, e'_2, each weighted by its length, as edges to $E(\hat{G}_\varepsilon)$. After this step, the union of paths $\hat{T}(x'') \cup \hat{T}(y'') \cup \hat{T}(z'')$ actually separates $\hat{G}_\varepsilon$ into subgraphs. We contract the separator into a new root and continue this procedure recursively in each component as in [13].

Note that $\hat{G}_\varepsilon$ is constructed with respect to a set of shortest-path separators. Even though the graph $\hat{G}_\varepsilon$ is not planar, these paths recursively separate $\hat{G}_\varepsilon$ into subgraphs in a balanced way (note that $\hat{G}_\varepsilon$ is planar in the immediate vicinity of these paths).

The number of edges of $\hat{G}_\varepsilon$ can be estimated as follows. After the corresponding ε-covers and distances are computed, S is contracted to a new node r^S, which is *suppressed* [13]. Suppression means that, while r^S and all its adjacent edges are considered for constructing recursive separators, they are not considered for computing distances/shortest paths. When a separator S of the original graph is constructed, the separator property we use afterwards is that any path between a pair of nodes from different components of $G_\varepsilon \setminus S$ must contain a node from S. Then all new edges are also suppressed. Hence, when recursively constructing separators for the components of $G_\varepsilon \setminus S$, we do not have to add new nodes and edges for intersections between separator edges and suppressed (new) edges. As a result, each edge of G_ε can be divided into two new edges at most once and hence $\left|E(\hat{G}_\varepsilon)\right| \leq 2\,|E(G_\varepsilon)|$.

Lemma 3. *The graph $\hat{G}_\varepsilon$ and the shortest-path separators can be computed in time $O(|V|\log^2|V| + |E|\log|V|)$, where $|V| = O((n/\sqrt{\epsilon})\log(1/\epsilon))$ and $|E| = O(n\epsilon^{-1}\log^2(1/\epsilon))$. Given two points s and t in $\hat{G}_\varepsilon$, one can find in $O(1)$ time a separator from the shortest-path separator decomposition separating s and t.*

4 Preprocessing Algorithm

We describe the preprocessing algorithm for our approximate distance oracle. We first give a brief overview, next we provide pseudocode, and finally we analyze the algorithm. The algorithm consists of three phases.

1. Discretization: the algorithm discretizes the surface and the weights
2. Pseudo-planarization: the algorithm finds shortest-path separators Q and adds nodes in order to make Q separating (Section 3.2)
3. Data-structure construction: the algorithm computes links to portals on the separator paths Q for increasing scales

Its pseudocode is listed as Preprocess in the following.

```
Preprocess $(P, \varepsilon, \delta)$
    let $G_\varepsilon$ be the graph obtained by the surface discretization in [2,3] and
    the weight discretization described in Section 3.1 with maximum weight $N$
    compute $\hat{G}_\varepsilon$ as described in the previous section
    compute Nearest Common Ancestor data structure for separator tree
    for each separator path $Q$
        partition $Q$ into maximal pieces of length $\leq \delta$
        enumerate the subpaths $Q_j^\delta$
```

```
for each subpath $Q_j^\delta$ compute an $\varepsilon$–cover (equally spaced points) $C(Q_j^\delta)$
enumerate the nodes in all the $C(Q_j^\delta)$
for $r \in \{0, \dots 4\}$
    let $\mathcal{C}^j$ denote the covers for all subpaths $Q_j^\delta$ with $j \mod 5 \equiv r$
    for $i \in \{1, 2, \dots O(1/\varepsilon)\}$
        let $N_i$ be the set of all nodes $i$ in all the covers in $\mathcal{C}^j$
        compute multiple-source shortest-path tree with all nodes
        in $N_i$ as sources of weighted depth $2\delta$
```

Lemma 4 ([9, Lemma 4]). *For any path $u - v$ of length $[\delta, 2\delta]$ in $\hat{G}_\epsilon$ that intersects a shortest-path separator Q_j^δ there is an alternative path $u - q - v$ for a portal $q \in C(Q_j^\delta)$ such that $\mathsf{len}(u - q - v) \leq (1+\epsilon)\mathsf{len}(u - v)$.*

Lemma 5. *For a triangulated surface P with n triangles, given $\varepsilon > 0$ and $\delta > 0$, algorithm* Preprocess (P, ε, δ) *computes an $(1+\varepsilon)$–approximate distance oracle for distances of length $\ell \in [\delta, 2\delta]$ in P in time $O(n\epsilon^{-2} \log^2(n/\varepsilon) \log^2(1/\epsilon))$. The space requirement is $O(n\varepsilon^{-3/2} \log(n/\varepsilon) \log(1/\varepsilon))$.*

Proof. Each Steiner point stores ϵ–covers of size $O(\epsilon^{-1})$ per level. There are $O(\log(n/\epsilon) \log(1/\epsilon))$ levels. The space per scale is thus $O(n\varepsilon^{-3/2} \log(n/\varepsilon) \log(1/\varepsilon))$.

Computing the covers requires $O(\epsilon^{-1})$ shortest-path-tree constructions per level. The time per scale is bounded by $O(n\epsilon^{-2} \log^2(n/\varepsilon) \log^2(1/\epsilon))$. ☐

5 Answering Approximate Shortest-Path Queries

5.1 Overview of the method

In order to answer approximate distance queries for an arbitrary pair of query points s and t from P, we use an algorithm for answering distance queries in $\hat{G}_\varepsilon$. Let, for any point p in P that is not in G_ε, the neighborhood $\mathcal{N}(p)$ of p be defined as the set of nodes of G_ε contained in the triangle(s) containing p and all adjacent (i.e. sharing an edge) triangles. If p is a node in G_ε, then we define $\mathcal{N}(p) = \{p\}$. Then, the approximate distance between s and t can be computed as

$$\tilde{\mathsf{d}}(s,t) = \min_{p_s \in \mathcal{N}(s), p_t \in \mathcal{N}(t)} \{\mathsf{d}_P(s, p_s) + \mathsf{d}_{\hat{G}_\epsilon}(p_s, p_t) + \mathsf{d}_P(p_t, t)\}. \tag{2}$$

Since points s and p_s are in the same or in neighboring triangles, $\mathsf{d}_P(s, p_s)$ can be computed using an explicit formula based on Snell's law. The same applies for computing $\mathsf{d}_P(p_t, t)$. For computing $\mathsf{d}_{\hat{G}_\epsilon}(p_s, p_t)$, we use the separator decomposition constructed in Lemma 3. Let Q be a shortest-path separator separating p_s and p_t. Then any path between p_s and p_t in $\hat{G}_\varepsilon$ must contain a node in Q and we therefore have $\mathsf{d}_{\hat{G}_\epsilon}(p_s, p_t) = \min_{q \in Q} \{\mathsf{d}_{\hat{G}_\epsilon}(p_s, q) + \mathsf{d}_{\hat{G}_\epsilon}(q, p_t)\}$.

Unless stated otherwise, we assume in this section that $\mathsf{d}_{\hat{G}_\varepsilon}(s,t) \in [\delta, 2\delta]$, and that Q is a shortest-path separator in $\hat{G}_\varepsilon$ of length $O(\delta)$ separating s and t, as outlined in the preprocessing algorithm. Under those assumptions, $|C(Q)| = O(1/\varepsilon)$ and one can use Lemma 4 to approximate each distance on the right-hand side of the previous equality, thereby having to look at only $1/\varepsilon$ nodes in $C(Q)$ per distance computation instead at all the nodes in Q, resulting in an $1+\varepsilon$ approximation of $\mathsf{d}_{\hat{G}_\epsilon}(p_s, p_t)$. Hence, instead of (2), we can use the equality

$$\hat{\mathsf{d}}(s,t) = \min_{p_s \in \mathcal{N}(s), p_t \in \mathcal{N}(t)} \min_{q \in C(Q)} \mathsf{d}_P(s, p_s) + \mathsf{d}_{\hat{G}_\epsilon}(p_s, q) + \mathsf{d}_{\hat{G}_\epsilon}(q, p_t) + \mathsf{d}_P(p_t, t). \quad (3)$$

The number of pairs (p_s, p_t) is $|\mathcal{N}(s)| \cdot |\mathcal{N}(t)| = \Theta(\epsilon^{-1} \log^2(1/\epsilon))$ and the size of $|C(Q)|$ is at most $1/\varepsilon$. Hence the total time for answering the approximate distance query based on formula (3) is $O(\epsilon^{-2} \log^2(1/\epsilon))$.

In the following we describe a more efficient divide-and-conquer approach for computing the minimum of formula (3) that avoids looking at all pairs p_s, p_t of nodes in the neighborhoods $\mathcal{N}(s)$ and $\mathcal{N}(t)$ and leads to a query time complexity roughly proportional to $\tilde{O}(\epsilon^{-1})$, ignoring logarithmic factors.

5.2 Divide-and-conquer approach

We reduce the problem of computing the minimum from (3) to the problem of finding the distances from all points of $C(Q)$ to s and t. We reduce the problem(s) of finding all distances from (to) $C(Q)$ to a single problem, rather than a sequence of $|C(Q)|$ problems.[4] Once we have those distances, the algorithm requires an additional $O(|C(Q)|) = O(1/\varepsilon)$ time to compute $\hat{\mathsf{d}}(s,t)$. We describe the computation of the distances to t as the ones for s are similar.

The idea is the following. There are $|C(Q)|$ nodes from which we want to compute shortest paths distances and each path could possible go through each of the nodes in $\mathcal{N}(t)$. For large $|C(Q)|$ and $|\mathcal{N}(t)|$ many paths intersect each other. It is well-known that in a planar graph, a pair of shortest intersecting paths to the same target can always be replaced by a pair of shortest non-intersecting paths with the same sources and target as the original, as exploited in [6,5]. We could use that property to significantly reduce the search space when computing the shortest paths from $C(Q)$. Unfortunately, our graph $\hat{G}_\varepsilon$ is not planar. We need to prove a similar property for $\hat{G}_\varepsilon$. First we establish several properties of paths in P and $\hat{G}_\varepsilon$ that follow from the definition of $\hat{G}_\varepsilon$.

Lemma 6. *Let p_1 and p_2 be two intersecting shortest paths in P with the same target t. There exists a path p_2^* in P with the same source and target as p_2 that does not intersect p_1 and such that $\mathsf{len}(p_2^*) = \mathsf{len}(p_2)$.*

Lemma 7. *Let π_1 and π_2 be two intersecting paths in $\hat{G}_\varepsilon$ with the same target node t. There exists a path π_2^* with the same source and target as π_2 that does not intersect π_1 and such that $\mathsf{len}(\pi_2^*) \leq (1+\varepsilon)\mathsf{len}(\pi_2)$.*

[4] This is why we compute portals to paths at different scales. By operating at one scale δ, we can use a *single set of portals $\mathcal{C}$ per path*, wherein we can efficiently search the best Steiner points for a pair of query points.

The next lemma is instrumental in our divide-and-conquer approach.

Lemma 8. *Let Q be a shortest-path separator, let q', q'' and q be nodes of $C(Q)$ such that q is between q' and q'' on Q . Let π' and π'' be two nonintersecting shortest paths in G_ε from q' and q'' to point t of P and let $\mathcal{N}'(t)$ be the subset of $\mathcal{N}(t)$ that is inside or on the boundary of the cycle determined by Q, π', and π''.*

If π' and π'' contain a node from $\mathcal{N}(t)$, then there is a path π in G_ε from q to t containing a node from $N'(t)$ such that $\mathsf{len}(\pi) \leq (1+\varepsilon)\mathsf{len}(\pi_{opt})$, where π_{opt} is the shortest path from q to t in G_ε.

Proof. If π_{opt} intersects neither π' nor π'', then we set $\pi = \pi_{opt}$. Else, suppose that the first path that π_{opt} intersects is π' and let (u', v') be the edge of π' that intersects an edge (u, v) from π_{opt} with v' between u' and t on π' and v between u and t on π. By Lemma 7, there exists a path π from q to t that does not intersect π' and such that $\mathsf{len}(\pi) \leq (1+\varepsilon)\mathsf{len}(\pi_{opt})$. Moreover, by the proof of Lemma 7, π coincides with π' in the portion of π' between v' and t and with π_{opt} in the portion between q and u. Since, by assumption, π' and π'' are not intersecting, then π does not intersect π''. ☐

The next lemma summarizes the previous results of this section and gives a fast algorithm for answering approximate distance queries in P.

Lemma 9. *Let s, t be two points on P and let Q be a separator path separating s from t and let $\mathcal{C}(Q)$ denote the ϵ–cover of Q of size $1/\varepsilon$. One can find in $O(1/\epsilon)$ time a pair of points $(p_s, p_t) \in \mathcal{N}(s) \times \mathcal{N}(t)$ and $q \in C(Q)$ such that*

$$\hat{\mathsf{d}}(s,t) = \mathsf{d}_P(s,p_s)+\mathsf{d}_{\hat{G}_\epsilon}(p_s,q)+\mathsf{d}_{\hat{G}_\epsilon}(q,p_t)+\mathsf{d}_P(p_t,t) \leq (1+O(\varepsilon\log(\varepsilon^{-1})))\delta_{\hat{G}_\varepsilon}(s,t).$$

Proof. To answer the query, we first compute, for each $q \in C(Q)$, the distance $\hat{\mathsf{d}}(q,s)$ from q to s, then the distance $\hat{\mathsf{d}}(q,t)$ from q to t, and finally compute $\min\limits_{q \in C(Q)} \{\hat{\mathsf{d}}(s,q) + \hat{\mathsf{d}}(q,t)\}$ in $O(|C(Q)|) = O(1/\varepsilon)$ time. Hence we only need to describe how to compute $\hat{\mathsf{d}}(s,q)$ in $O(1/\varepsilon)$ time.

Let q' and q'' be the two outermost cover points of Q. Construct the shortest paths from s to q' and q''. These two paths and q form a cycle c through s. Let $\mathcal{N}(s; q', q'')$ denote the subset of $\mathcal{N}(s)$ that is inside c. Let node $q_m \in C(Q)$ divide $C(Q)$ into to roughly equal subsets with respect to the order on Q. By Lemma 8, there exists a shortest path from q_m to s that contains a point from $\mathcal{N}(s; q', q'')$ that is within $1+\varepsilon$ factor of the length of the shortest path between q_m and s. Hence the length of such a shortest path can be found in $O(|\mathcal{N}(s; q', q'')|)$ time. That path divides $\mathcal{N}(s; q', q'')$ into two subsets, $\mathcal{N}(s; q', q_m)$ and $\mathcal{N}(s; q_m, q'')$.

By the same method, we find the length of a shortest path from a middle node between q' and q_m to s and a shortest path from a middle node between q_m and q'' to s. The total time to find *both* these lengths is $O(|\mathcal{N}(s; q', q'')|)$. In the next step we find four more shortest-path lengths (approximate distances) from nodes from $C(Q)$ to s in time $O(|\mathcal{N}(s; q', q'')|)$. The time of this algorithm is $O(\varepsilon^{-1})$ and the approximation ratio is $1 + O(\varepsilon \log(\frac{1}{\varepsilon}))$.

In order to achieve stretch $1+\varepsilon$ in the final algorithm, rather than $1+O(\varepsilon)$, we need to know an explicit bound on the stretch from Lemma 9, instead of an estimation in terms of big-O asymptotics. The proof of the lemma does not yield the constant, but it does give us that the stretch is not exceeding $(1+\varepsilon)^{\log(\frac{1}{\sqrt{\varepsilon}})}$. For any specific value of ε, we can estimate the stretch by that formula and use it for computing the parameters for the preprocessing algorithm. On the other hand, by Lemma 9, we know that there exists a constant k independent of ε such that the stretch is bounded by $1+k\varepsilon\log(\varepsilon^{-1})$.

Lemma 10. *Given two points s and t in P and $\varepsilon \in (0,1)$, one can find in $O(\epsilon^{-1})$ time a path π in $\hat{G}_\varepsilon$ between s and t such that $\mathsf{len}(\pi) \le (1+k\varepsilon\log(\frac{1}{\varepsilon}))\mathsf{d}(s,t)$ for some constant k.*

Proof. By Lemma 3, one can find in $O(\log\log n)$ time a separator path Q of length $O(q)$ separating s and t in $\hat{G}_\varepsilon$. By Lemma 4, there is a set of $C(Q)$ of $O(1/\varepsilon)$ attachment points for Q that can be used to approximate any distance in $[\delta, 2\delta]$ to a point in Q with $1+\varepsilon$ approximation factor.

Let π be a path between s and t in the approximation graph $\hat{G}_\varepsilon$ such that $\mathsf{len}(\pi) \le (1+\varepsilon)\mathsf{d}(s,t)$. As Q separates s and t, there exists a node $q \in Q$ such that $\mathsf{len}(\pi) = \mathsf{d}(s,q) + \mathsf{d}(q,t)$. By Lemma 4, there exists a node $q' \in C(Q)$ such that $\mathsf{d}(s,q') \le (1+\varepsilon)\mathsf{d}(s,q)$ and $\mathsf{d}(t,q') \le (1+\varepsilon)\mathsf{d}(t,q)$. Hence, there is a path π' in $\hat{G}_\varepsilon$ from s to t that contains a node from $C(Q)$ such that $\mathsf{len}(\pi') \le (1+\varepsilon)^2\mathsf{d}(s,t)$. By Lemma 10, one can find in $O(1/\varepsilon)$ time a path π'' in $\hat{G}_\varepsilon$ that contains a node in $C(Q)$ and is within a $1+O(\varepsilon\log(\varepsilon^{-1}))$ factor of the distance between s and t in $\hat{G}_\varepsilon$. Hence,

$$\begin{aligned}\mathsf{len}(\pi'') &\le (1+O(\varepsilon\log(\varepsilon^{-1})))\delta_{\hat{G}_\varepsilon}(s,t) \le (1+O(\varepsilon\log(\varepsilon^{-1})))\mathsf{len}(\pi') \\ &\le (1+O(\varepsilon\log(\varepsilon^{-1})))(1+\varepsilon)^2\mathsf{d}(s,t) = (1+O(\varepsilon\log(\varepsilon^{-1})))\mathsf{d}(s,t). \quad \square\end{aligned}$$

Now we are ready to prove the main result of this paper, Theorem 2. Recall that so far in this section we have assumed that Q is a shortest-path separator in $\hat{G}_\varepsilon$ of length $O(\delta)$ separating s and t, and $\mathsf{d}(s,t) \in [\delta, 2\delta]$. We need to compute such δ in our algorithm and to choose an appropriate value for the approximation factor in Lemma 10 so that the computed path is within a $1+\varepsilon$ approximation factor of the shortest path, as opposed to a $1+O(\varepsilon\log(\frac{1}{\varepsilon}))$ factor.

5.3 Proof of Theorem 2

Proof (of Theorem 2). In order to find an $O(1)$ approximation for $\mathsf{d}(s,t)$ we do a query using Lemma 10 with $\varepsilon = 1/2$ (strictly speaking, we preprocess another distance oracle for $\varepsilon = 1/2$ and query it here) and do a binary search on the values of δ in $\{1, 2, 4, \ldots, 2^{\lceil \log(nN) \rceil}\}$. We find in $O(\log\log(nN)) = O(\log\log n)$ time the minimum value δ_0 for δ for which the path from Lemma 10 has length $l_{1/2} < \infty$. By Lemma 10, $\mathsf{d}(s,t) \le l_{1/2} \le (1+k/2)\mathsf{d}(s,t)$. By the minimum property of δ, $l_{1/2} > 2\delta_0/2 = \delta_0$ and hence $\delta_0 < l_{1/2} < 2\delta_0$. By combining this with the previous inequality we get $\delta_0/(1+k/2) < \mathsf{d}(s,t) < 2\delta_0$.

Next, we do a binary search on the values of δ in $\{2^{i_1}, 2^{i_1+1}, \dots, 2^{i_2}\}$ for $i_1 = \lfloor \log(\delta_0/(1+k/2)) \rfloor$ and $i_2 = \lfloor \log(2\delta_0) \rfloor$ and ε determined by $\varepsilon \le \epsilon/(k \log(\frac{1}{\varepsilon}))$. Such ε can be determined in $O(\log(\epsilon^{-1}))$ time and the search takes $O(1)$ time. By Lemma 10, the returned distance l satisfies

$$l \le (1 + k\varepsilon \log(\frac{1}{\varepsilon}))\mathsf{d}(s,t) \le (1 + k\epsilon/(k \log(\frac{1}{\varepsilon})) \log(\frac{1}{\varepsilon}))\mathsf{d}(s,t) = (1+\epsilon)\mathsf{d}(s,t).$$

In order to determine the running time, we use that $\epsilon = \Theta(\varepsilon \log(1/\varepsilon))$ and hence

$$\varepsilon = \Theta(\epsilon/\log(1/\varepsilon)) = O(\epsilon/\log(1/\epsilon))$$

as $\epsilon = o(\varepsilon^2)$. By replacing that estimation for ε in the complexity bounds in Lemmas 5 and 10, we get the claimed complexity bounds. □

References

1. Abraham, I., Gavoille, C.: Object location using path separators. In: PODC. pp. 188–197 (2006)
2. Aleksandrov, L., Djidjev, H.N., Guo, H., Maheshwari, A., Nussbaum, D., Sack, J.R.: Algorithms for approximate shortest path queries on weighted polyhedral surfaces. Discrete Comput. Geom. 44, 762–801 (2010)
3. Aleksandrov, L., Maheshwari, A., Sack, J.R.: Determining approximate shortest paths on weighted polyhedral surfaces. J. ACM 52(1), 25–53 (2005)
4. Cabello, S.: Many distances in planar graphs. In: SODA. pp. 1213–1220 (2006), a preprint of the journal version is available in the University of Ljubljana preprint series, Vol. 47 (2009), 1089
5. Djidjev, H.: Efficient algorithms for shortest path queries in planar digraphs. In: Graph-Theoretic Concepts in Computer Science, vol. 1197, pp. 151–165 (1997)
6. Fakcharoenphol, J., Rao, S.: Planar graphs, negative weight edges, shortest paths, and near linear time. J. Comput. Syst. Sci. 72(5), 868–889 (2006)
7. Kawarabayashi, K., Klein, P.N., Sommer, C.: Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In: ICALP. pp. 135–146 (2011)
8. Klein, P.N.: Preprocessing an undirected planar network to enable fast approximate distance queries. In: SODA. pp. 820–827 (2002)
9. Klein, P.N., Subramanian, S.: A fully dynamic approximation scheme for shortest paths in planar graphs. Algorithmica 22(3), 235–249 (1998)
10. Mitchell, J.S.B., Papadimitriou, C.H.: The weighted region problem: finding shortest paths through a weighted planar subdivision. J. ACM 38, 18–73 (1991)
11. Mozes, S., Sommer, C.: Exact distance oracles for planar graphs. CoRR abs/1011.5549 (2010)
12. Sun, Z., Reif, J.H.: On finding approximate optimal paths in weighted regions. J. Algorithms 58(1), 1–32 (2006)
13. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM 51(6), 993–1024 (2004)
14. Thorup, M., Zwick, U.: Approximate distance oracles. J. ACM 52(1), 1–24 (2005)