Algorithms for Approximate Shortest Path Queries on Weighted Polyhedral Surfaces

Lyudmil Aleksandrov · Hristo N. Djidjev ·
Hua Guo · Anil Maheshwari · Doron Nussbaum ·
Jörg-Rüdiger Sack

Received: 21 May 2009 / Revised: 2 June 2009 / Accepted: 2 June 2009 / Published online: 7 July 2009
© Springer Science+Business Media, LLC 2009

Abstract We consider the well-known geometric problem of determining shortest paths between pairs of points on a polyhedral surface P , where P consists of triangular faces with positive weights assigned to them. The cost of a path in P is defined to be the weighted sum of Euclidean lengths of the sub-paths within each face of P .

We present query algorithms that compute approximate distances and/or approximate shortest paths on P . Our all-pairs query algorithms take as input an approximation parameter $\varepsilon \in (0, 1)$ and a query time parameter q , in a certain range, and build a data structure $APQ(P, \varepsilon; q)$, which is then used for answering ε -approximate distance queries in $O(q)$ time. As a building block of the $APQ(P, \varepsilon; q)$ data structure, we develop a single-source query data structure $SSQ(a; P, \varepsilon)$ that can answer ε -approximate distance queries from a fixed point a to any query point on P in logarithmic time. Our algorithms answer shortest path queries in weighted surfaces, which is an important extension, both theoretically and practically, to the extensively studied Euclidean distance case. In addition, our algorithms improve upon previously known query algorithms for shortest paths on surfaces. The algorithms are based on a novel graph separator algorithm introduced and analyzed here, which extends and generalizes previously known separator algorithms.

Research supported by the Natural Sciences and Engineering Research Council of Canada, SUN Microsystems, and the US Department of Energy under contract W-705-ENG-36. Preliminary research leading to this paper has appeared in [5, 10]. Research was carried out in part while the first and second authors were visiting Carleton University, Ottawa, Canada.

L. Aleksandrov
Bulgarian Academy of Sciences, Sofia, Bulgaria

H.N. Djidjev
Los Alamos National Laboratory, Los Alamos, USA

H. Guo · A. Maheshwari · D. Nussbaum · J.-R. Sack (✉)
Carleton University, Ottawa, Canada
e-mail: sack@scs.carleton.ca

Keywords Shortest path · Query algorithm · Separator · Weighted polyhedral surface

1 Introduction

Shortest path problems rank among the fundamental computer science problems, different aspects of which are studied in computational geometry, network optimization, graph algorithms, geographical information systems (GIS), and others. These problems arise naturally in various applications such as motion/route planning, navigation, graphics, seismology, injection molding (for references, see [8]). Aside from the importance of shortest path problems in their own right, they often appear in the solutions to other problems.

Unlike the well-studied graph version of the shortest path problem, computing shortest paths in geometric domains is a much more challenging problem. Existing algorithms for many geometric shortest path problems are quite complex in both design and implementation or have large time and space complexities. Hence, they are unappealing to practitioners and pose a challenge to theoreticians. Problem instances are known to be computationally “hard,” see, e.g., [13]. This, along with the fact that geometric models are approximations for reality and there is a need for fast algorithms computing high-quality but not necessarily optimal paths, makes approximation algorithms the best available option.

In many cases, the computation of shortest paths with respect to the classic Euclidean distance does not provide an adequate solution. Non-Euclidean distances naturally occur in a variety of key application domains, such as GIS, Robotics, Optics, and Seismology, where the regions are nonhomogeneous, and the cost of travel is different at different locations. For example, in GIS, a terrain could consist of different types of regions (e.g., water, forest, rocks) that are modeled by assigning suitable weights to each of them. This leads to the so-called weighted shortest path problem. Considering weighted problems adds further complexity to solutions.

Frequently, shortest path queries between different pairs of points are executed repeatedly over time for the same domain. Examples of applications employing repeated shortest path queries are tourist information systems and planning search-and-rescue strategies in terrains. Due to the relatively high time complexities for shortest path computations, in particular in weighted domains, efficient shortest path query algorithms for such applications are not only desirable but often are the only way that timely answers can be provided. This motivates our search for algorithms for answering approximate shortest path queries.

We consider paths that stay on a connected polyhedral surface¹ P in the three-dimensional Euclidean space consisting of n positively weighted triangular faces. The cost of a path lying inside a face is its Euclidean length multiplied by the weight of the face. The cost of a general path on P is the sum of the costs of the sub-paths within each face traversed. For a pair of points a and b on P , a path of least cost between

¹Surface P can be any polyhedral 2-manifold with no additional assumed geometrical and/or topological properties like convexity, being a terrain, absence of holes, etc., assumed.

them is called a *shortest path*, and its cost is called a *distance* between a and b . Clearly, this weighted scenario is a generalization of the classic Euclidean distance model, which is obtained by assigning unit weights to all faces. In this paper, we present algorithms for answering approximate distance (and/or approximate shortest path) queries between pairs of points on P .

More precisely, let ε be a fixed real number in $(0, 1)$. For a pair of points a and b on P , an ε -*approximate* (or simply *approximate* if no ambiguity arises) distance between these points is any number whose ratio with the distance between a and b is in $(1 - \varepsilon, 1 + \varepsilon)$. A path between a and b on P whose cost is an ε -approximate distance is called an ε -*approximate (approximate) shortest path*.

In this setting, the approximate shortest path query problem is: preprocess the surface P so that an approximate distance (and/or an approximate shortest path) between any query pair of points in P can be computed efficiently. We distinguish two standard variations of this problem. In the *All Pairs Query* (APQ) problem, the query consists of a pair of arbitrary points on P (as well as pointers to the faces of P containing the query points). In the *Single Source Query* (SSQ) problem, the query consists of a single point, whereas the other point, called *source*, is fixed and given in advance.

The main result of this paper is a family of approximate solutions to the APQ problem on weighted polyhedral surfaces. More precisely, we present algorithms that, for an input parameter q in a certain range, construct a data structure $\text{APQ}(P, \varepsilon; q)$, which is then used to answer approximate shortest path queries between arbitrary pairs of points in P in $O(q)$ time per query. As a building block for our solution to the APQ problem, we present an approximate solution for the SSQ problem as well. To the best of our knowledge, these are the first solutions to the APQ and SSQ problems on weighted surfaces. These problems were mentioned as open problems in [30]; it is easy to see that they are natural ones to consider once we know how to answer queries on surfaces under Euclidean distance metric. To place our work in the context of the literature, we next state some relevant results.

1.1 Overview of Previous Work

A large amount of research has been devoted to efficient computation of shortest paths in geometric environments. Here, we concentrate on results related to SSQ and APQ problems on polyhedral surfaces in the three-dimensional Euclidean space. With a few exceptions, these results have been obtained with respect to the Euclidean distance, and in the discussion below, we assume the Euclidean distance if not explicitly specified otherwise. We distinguish between exact and approximate SSQ and APQ problems, referring to the cases where finding exact or approximate distances is required.

Work on Exact Problems A solution to the exact SSQ problem on general polyhedral surfaces was presented initially in [39] and then improved in [34]. For a surface of size n and a source point s , the algorithm in [34] takes $O(n^2 \log n)$ preprocessing time to construct a diagram that can be used to answer distance queries to s in $O(\log n)$ time. Applying an alternate approach, Chen and Han [16, 17] obtained an algorithm with improved preprocessing time and controllable query time. For a surface

of size n and an input parameter $1 \leq d \leq n$, their algorithm takes $O(n^2)$ preprocessing time to construct a data structure of size $O(n \log n / \log d)$ that allows answering distance queries to the source in $O(d \log n / \log d)$.

The exact APQ problem on general polyhedral surfaces is believed to be complex, both theoretically and computationally. To the best of our knowledge, no complete solution to this problem has been published so far. It had been announced in [1, 21] that exact methods that work for special cases, such as planar polygonal domain [21] and surface of convex polytopes [1], extend to the general case. These potential solutions represent a theoretical interest only, due to their high time and space complexities.

Due to the difficulty of the exact SSQ and APQ problems in the general case, researchers have paid special attention to finding efficient solutions under certain restrictions imposed on the domain. Guibas and Hershberger [28] presented an asymptotically optimal solution of the exact APQ problem inside a simple polygon in the plane. In $O(n)$ time, their algorithm constructs a data structure that allows answering distance queries between arbitrary pairs of points inside a simple polygon of size n in $O(\log n)$ time. In addition, the corresponding shortest paths can be reported in time proportional to their combinatorial size.

The exact SSQ problem in planar polygonal domains drew considerable attention and has been studied extensively. An asymptotically optimal algorithm for the exact SSQ problem has been presented in [31]. For a planar polygonal domain of size n , it takes $O(n \log n)$ time to build a data structure of size $O(n)$ that can then be used to answer single-source queries in $O(\log n)$ time.

A number of suboptimal solutions for the exact APQ problem in planar polygonal domains have been published. For a domain defined by h polygonal obstacles of total size n , algorithms with $O(n)$, $O(h + \log n)$, $O(h \log n)$, $O(\log^2 n)$, and $O(\log n)$ query times are proposed in [21]. The bounds on the preprocessing time and the space for the corresponding data structures vary from $O(n^5)$ to $O(n^{15})$.

An algorithm for solving the exact APQ problem with $O(n^2)$ bound on the preprocessing time and space has been proposed in [18]. The bound on the query time of this algorithm depends on the position of the query points and is $O(n \log n)$ in the worst case.

An optimal solution of the exact SSQ problem on the surface of a convex polytope has been recently presented in [38]. For a polytope of size n , it takes $O(n \log n)$ preprocessing time, and distance queries to the source are answered in $O(\log n)$ time. The exact APQ problem on the surface of a convex polytope has been studied by Agarwal et al. in [1]. They proposed a scheme that, for a convex polytope of size n and an input parameter m , $1 \leq m \leq n^2$, takes $O(n^6 m^{1+\delta})$ ($\delta > 0$) preprocessing time and space to construct a data structure that serves distance queries between arbitrary pairs of points in $O(\sqrt{n} \log n / m^{1/4})$ time. See Table 1 for more details.

Work on Approximate Problems The high complexity of the available solutions for the exact problems, especially for the APQ problem, sustains the interest towards finding conceptually simpler and more efficient approximate solutions.

In [30], an algorithm for solving the approximate SSQ problem on general polyhedral surfaces is proposed. For a surface of size n , a source point s , and an approximation parameter ε , $0 < \varepsilon < 1$, the algorithm uses $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$ preprocessing

Table 1 Selected exact solutions to the Euclidean SSQ and APQ problems. In the second column, terms Planar, Convex, and General stand for Planar polygonal domain, Convex polyhedral surface, and General polyhedral surface, correspondingly

Problem	Geom. model	Preprocessing	Space	Query	Remarks	Ref.
SSQ	Planar	$O(n \log n)$	$O(n)$	$O(\log n)$		[31]
SSQ	Convex	$O(n \log n)$	$O(n)$	$O(\log n)$		[38]
SSQ	General	$O(n^2)$	$O(n \log n / \log d)$	$O(d \log n / \log d)$	$1 \leq d \leq n$	[17]
APQ	Planar	$O(n^{5+10\delta_1+\delta})$	$O(n^{5+10\delta_1+\delta})$	$O(n^{1-\delta} \log n)$	$0 < \delta_1 \leq 1, \delta > 0$	[21]
APQ	Convex	$O(n^6 m^{1+\delta})$	$O(n^6 m^{1+\delta})$	$O(\frac{\sqrt{n}}{m^{1/4}} \log n)$	$1 \leq m \leq n^2, \delta > 0$	[1]

time and builds a data structure of size $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$. Approximate distance queries to s are then answered in $O(\log \frac{1}{\varepsilon})$ time. This assumes that the face containing the query point is specified with the query; otherwise, a point location data structure needs to be used (e.g., the one in [40] can be used which requires $O(n^2 \log n)$ time and $O(n^2)$ space for preprocessing and $O(\log n)$ time for locating the face containing the query point). Moreover, a solution of the approximate SSQ problem on the surface of a convex polytope with improved preprocessing time and data structure size has been proposed in [30].

In [22], Clarkson presented an algorithm for solving the approximate APQ problem in planar polygonal domains. For an approximation parameter $0 < \varepsilon \leq \pi$ and a domain of size n , the algorithm builds, in $O(\frac{n}{\varepsilon} \log n)$ time, a data structure of size $O(\frac{n}{\varepsilon})$ that supports approximate shortest path queries in $O(n(\frac{1}{\varepsilon} + \log n))$ time. As noticed by Chen in [15], Clarkson's approach can be used to obtain an algorithm with query time reduced to $O(\frac{1}{\varepsilon}(\frac{1}{\varepsilon} + \log n))$ at the expense of a preprocessing time increased to $O(n^2(\frac{1}{\varepsilon} + \log n))$ and a size of the data structure increased to $O(n(\frac{1}{\varepsilon} + n))$. In the same paper, an alternative algorithm with $O(\sqrt{n^3/\log n})$ preprocessing time and $O(n \log n)$ size of the data structure has been presented. This algorithm is not a true approximation algorithm, since the reported distances can be up to 6 times greater than the exact ones.

In [11], Arikati et al. presented a family of algorithms for solving the approximate APQ problem in planar polygonal domains under L_p metrics ($1 \leq p \leq \infty$). These algorithms achieve various trade-offs between the quality of approximation, the query time, the size of the data structures, and the preprocessing time. Similarly to Chen's algorithm, these algorithms do not guarantee true approximation, except for the case $p = 1$.

Approximation problems on the surface of a convex polytope P have been extensively studied. A series of results have been obtained based on a result of Dudley [25] about Hausdorff approximation of convex sets. Dudley's result shows that P can be approximated by a convex polytope Q of size $O(1/\varepsilon^{\frac{3}{2}})$ so that distances on Q provide ε -approximation to the distances on P . This was used in the algorithm of [2] to compute an ε -approximate shortest path in $O(\frac{1}{\varepsilon^3} + n \log \frac{1}{\varepsilon})$ time. Later, in [29], this algorithm was extended to solve the approximate APQ problem in $O(n)$ preprocessing time and size of the data structure and in $O(\frac{\log n}{\varepsilon^{3/2}} + \frac{1}{\varepsilon^3})$ query time. Practical issues

Table 2 Selected approximate solutions to the Euclidean SSQ and APQ problems

Problem	Geom. model	Preprocessing	Space	Query	Ref.
SSQ	General	$O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$	$O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$	$O(\log \frac{1}{\varepsilon})$	[30]
APQ	Planar	$O(\frac{n}{\varepsilon} \log n)$	$O(\frac{n}{\varepsilon})$	$O(\frac{n}{\varepsilon} + n \log n)$	[22]
APQ	Planar	$O(\frac{n^2}{\varepsilon} + n^2 \log n)$	$O(n^2 + \frac{n}{\varepsilon})$	$O(\frac{n}{\varepsilon} + \frac{1}{\varepsilon^2})$	[15]
APQ	Convex	$O(n)$	$O(n)$	$O(\frac{1}{\varepsilon^3} + \frac{\log n}{\varepsilon^{3/2}})$	[29]

about this approach as well as implementation and experiments have been discussed in [3]. That algorithm finds an ε -approximate distance between any pair of points on P in $O(\frac{n}{\sqrt{\varepsilon}} + \frac{1}{\varepsilon^4})$ time. Recently, Chazelle et al. [14] proposed a randomized solution to the approximate APQ problem on convex surfaces. See Table 2 for more details.

The Weighted Case In the weighted surface model, it is assumed that positive weights are assigned to the surface faces. The cost of a path on the weighted surface is computed as the weighted sums of the Euclidean lengths of their portions inside the faces of the surface. The weighted model was introduced in [33] for planar subdivisions. Solutions to the exact problems in the weighted case seem to be unfeasible, since computing of exact distances even on very simple weighted surfaces with just a few faces requires a solution of high-degree algebraic equations.

Most of the known results in the weighted case concern restricted versions of the approximate SSQ problem. So, for the case of weighted planar subdivision of size n , the algorithm presented in [33] constructs a so-called restricted shortest path map from a fixed source point to the edges of the weighted planar subdivision of size $O(n^4)$ and in roughly $O(n^8 \log \frac{n}{\varepsilon})$ time. The map can then be used to answer approximate distance queries from the source to points on the edges of the surface in roughly $O(n^7 \log \frac{n}{\varepsilon})$ time.

In a series of papers [6–8, 35, 36], the so-called approximate Single-Source Shortest Path (SSSP) tree problem has been studied. In this problem, for a given source vertex on a weighted polyhedral surface of size n and an approximation parameter $\varepsilon \in (0, 1)$, one has to compute ε -approximate distances from the source to all other vertices of the surface. In the approximate SSSP tree problem, the query point is restricted to be a vertex rather than an arbitrary point on the surface. In the first three rows of Table 3, we list selected solutions to the approximate SSSP tree problem, which can be used for solving approximate SSQ and APQ problems on weighted surfaces, as it was sketched first in [5]. The method we develop and use in this paper can be viewed as a realization and generalization of the approach mentioned in [5].

The constants hidden in the “big-O” notation of all results, except the one in [19], presented in Table 3, depend on certain geometrical properties of the weighted surface P . The constant hidden in the “big-O” notation concerning the solutions of the SSQ and APQ problems presented in this paper is bounded by $5\Gamma \log 2L$, where Γ is the average of the reciprocals of the sines of the angles of the faces of P (see Lemma 3 here for more detail and [8] for further discussion). In theory, Γ can be big, but for this, a considerable portion of the faces of P should have big aspect ratios,

Table 3 Selected solutions to SSSP tree, SSQ, and APQ problems on weighted polyhedral surfaces. It is assumed that query consists of query points as well as pointers to the faces containing the query points

Problem	Geom. model	Preprocessing	Space	Query	Ref.
SSSP tree	General weighted	$O(\frac{n}{\epsilon^2} \log n \log \frac{1}{\epsilon})$	$O(n)$	$O(1)$	[6]
SSSP tree	General weighted	$O(\frac{n}{\epsilon} \log \frac{n}{\epsilon} \log \frac{1}{\epsilon})$	$O(n)$	$O(1)$	[35, 36]
SSSP tree	General weighted	$O(\frac{n}{\sqrt{\epsilon}} \log \frac{n}{\epsilon} \log \frac{1}{\epsilon})$	$O(n)$	$O(1)$	[7, 8]
SSQ	Planar weighted	$O(n^8 \log \frac{n}{\epsilon})$	$O(n^4)$	$O(n^7 \log \frac{n}{\epsilon})$	[33]
SSQ	Planar weighted	$O(\frac{n^4}{\epsilon^2} \log^2 \frac{n}{\epsilon})$	$O(\frac{n^4}{\epsilon^2} \log \frac{n}{\epsilon})$	$O(\log \frac{n}{\epsilon})$	[19]
SSQ	General weighted	$O(\frac{n}{\sqrt{\epsilon}} \log \frac{n}{\epsilon} \log \frac{1}{\epsilon})$	$O(\frac{n}{\sqrt{\epsilon}} \log \frac{1}{\epsilon})$	$O(\log \frac{1}{\epsilon})$	here
APQ	General weighted (genus g)	$O(\frac{(g+1)n^2}{\epsilon^{3/2}q} \log \frac{n}{\epsilon} \log^4 \frac{1}{\epsilon})$	$O(\frac{(g+1)n^2}{\epsilon^{3/2}q} \log^4 \frac{1}{\epsilon})$	$O(q)$	here
APQ	General weighted (genus 0)	$O(\frac{n^2}{\epsilon^2q} \log q \log \frac{n}{\epsilon} \log^4 \frac{1}{\epsilon})$	$O(\frac{n^2}{\epsilon^3q^2} \log^2 q \log^6 \frac{1}{\epsilon})$	$O(q)$	here

which is unrealistic in practice. For example, if none of the angles of P is smaller than 6° , then Γ is less than 10.

During the write-up of the journal version of [5, 10], the results in [19, 20], with respect to the weighted region problem in planar domains, have appeared. This work is inspired by our work and that of Reif and Sun [36], and the authors were able to remove the dependence on the geometric parameters at the expense of increasing the dependence on n as well as the ratio of the weights (max weight to min weight). Their analysis requires the bound on the maximum number of links possible in a weighted shortest path, which is $\Omega(n^2)$ as shown in [33].

1.2 Main Results

Our approach is based on three main techniques, which we develop, combine, and use in our solution of the approximate SSQ and APQ problems on weighted surfaces. We refer to these techniques as: (1) *weighted surface partitioning*, (2) *efficient solution of the approximate SSSP tree problem on weighted surfaces*, and (3) *local Voronoi diagrams*.

Partitioning techniques have been successfully applied to shortest path problems in both graphs and geometric environments. We consider graphs with nonnegative weights and positive costs assigned to their vertices. For such a graph $G(V, E)$ and a real number $t \in (0, 1)$, a t -separator is a set of vertices whose removal leaves no component of total weight exceeding $tw(G)$, where the weight of a component is the sum of the weights of its vertices, and $w(G)$ denotes the total weight of G . Any t -separator S defines a partition of the vertices of G into sets inducing the connected components of $G \setminus S$ and S itself. Let $V_1, \dots, V_k, S$ be the partition defined by a t -separator S . Then any vertex in the set V_i , for any $1 \leq i \leq k$, can be adjacent to vertices in $V_i \cup S$ only. The subset of vertices in S that are adjacent to vertices in V_i is called *boundary* of V_i (or of the component induced by V_i) and is denoted

by ∂V_i . The partition $V_1, \dots, V_k, S$ defined by a t -separator S is called *B-regular* (or simply *regular*), where B is a real number, if the total cost of the vertices in ∂V_i for $i = 1, \dots, k$ is no more than B . In Sect. 2, we propose an efficient algorithm for construction of B -regular partition for an embedded graph of degree three with weights and costs assigned to its vertices.

This graph separator problem formulation is rather general, and many of the known graph separator results can be viewed as solutions of its particular cases. For example, the well-known separator result by Lipton and Tarjan [32] can be viewed as special case of the problem formulated here, where G is planar, $t = 2/3$, and all vertices have unit cost. Another important example is the separator result by Frederickson [26] for planar graphs with unit costs, where an upper bound on the size of B is provided. Our result here extends and/or improves upon existing graph partitioning results of [4, 9, 23, 26, 27, 32].

We apply our partitioning technique to solve the approximate APQ problem on weighted polyhedral surfaces. The partition of the surface P is the first step in the construction of our APQ data structures. Different choices of the parameter t result in APQ data structures with different preprocessing times, sizes, and query times. So, we take query time as an input parameter q , which defines an appropriate choice of t , and obtain an APQ data structure supporting queries in $O(q)$ time. Our result is as follows. Let P be a weighted polyhedral surface of genus g and consisting of n triangular faces. Let $\varepsilon \in (0, 1)$ and $q \in (\frac{1}{\sqrt{\varepsilon}} \log^2 \frac{1}{\varepsilon}, \frac{(g+1)^{2/3} n^{1/3}}{\sqrt{\varepsilon}})$. There exists a data structure such that approximate distance queries in P can be answered in $O(q)$ time. The structure is constructed in $O(\frac{(g+1)n^2}{\varepsilon^{3/2}q} \log \frac{n}{\varepsilon} \log^4 \frac{1}{\varepsilon})$ time, and its size is $O(\frac{(g+1)n^2}{\varepsilon^{3/2}q} \log^4 \frac{1}{\varepsilon})$.

Our APQ data structure is supported in turn by several SSQ data structures. The central piece of the SSQ data structures are the approximate SSSP trees and local Voronoi diagrams. Recall that for an SSSP tree, we are given an approximation parameter $\varepsilon \in (0, 1)$ and a source vertex on the weighted triangulated surface P , and the goal is to construct an ε -approximate SSSP tree from the source vertex to all other vertices of P . Here, we develop and use a modification of the solution of the approximate SSSP tree problem presented in [8]. Our result is as follows. Given a point $a \in P$, a data structure of size $O(\frac{n}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$ exists, so that the approximate distance between a and a query point b in P can be computed in $O(\log \frac{1}{\varepsilon})$ time. The data structure can be constructed in $O(\frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time.

Since the weighted surface model is a generalization of the classic Euclidean distance model, our solutions are alternatives to the known solutions of approximate SSQ and APQ problems on surfaces and in planar polygonal domains with respect to Euclidean distance. Although the dependence of our solutions on the size of the surface n and on the approximation parameter ε compares favourably with some previous results (see Table 2) obtained with respect to the Euclidean distance, decisive comparison is difficult due to the fact that the parameters of our solutions depend on the geometry of the underlying surface. Our analysis reveals this dependence in detail.

For the case where the polygonal surface P has genus zero, i.e., P is homeomorphic to a sphere, we present a more elaborate algorithm providing an improved solution to the approximate APQ problem. This algorithm constructs a data structure

whose size is reduced approximately by a factor of $q\epsilon^{\frac{3}{2}}$, at the expense of an increase in preprocessing time by a factor of $\epsilon^{-\frac{1}{2}}$. An important feature of this solution is that the asymptotic dependence on n of the product of the size of the data structure and the query time q is sub-quadratic, provided that $q = \Omega(n^\delta)$ for $\delta > 0$.

1.3 Organization of the Paper

The remainder of this paper is organized as follows. In Sect. 2, we present and analyze our separator algorithm for graphs with weights and costs assigned to their vertices. In Sect. 3, we describe our discretization scheme involving the construction of a graph G_ϵ called approximation graph, show its approximation properties, and discuss the construction of SSSP tree in G_ϵ . In Sect. 4, we present our solution to the approximate SSQ problem. Next, in Sect. 5, our solution to the approximate APQ problem is presented. The important planar problem instance, i.e., weighted surfaces of genus zero, is discussed in Sect. 6. Finally, in Sect. 7, we conclude with a discussion of extensions and open problems.

2 Partitioning of Embedded Graphs with Weights and Costs

In this section, we present two new results on partitioning of embedded graphs of bounded genus with weights and costs assigned to their vertices. Let $G = (V, E)$ be an embedded graph of genus g , where each vertex $v \in V$ is endowed with nonnegative weight, denoted by $w(v)$, and nonnegative cost, denoted by $c(v)$. For a subgraph G' of G , we denote the sum of the weights of the vertices in G' by $w(G')$. Similarly, the sum of the costs of the vertices of G' is denoted by $c(G')$. We define the size of G to be $|V| + |E|$ and denote it by $|G|$. Throughout this section, t is a real number in $(0, 1)$. A set of vertices S of G is called a t -separator if its removal from G leaves no component of weight exceeding $tw(G)$. Recall that *genus* of a graph G is the minimum number of handles that must be added to a sphere so that G can be embedded on the resulting surface. Moreover, $g = 0$ if and only if G is planar.

Our results, presented as Theorems 1 and 2 below, show the existence and construction of small-cost t -separators in embedded graphs of genus g . The results are obtained by extending a number of graph partitioning techniques that appeared in [4, 9, 24, 26]. In the core is a technique developed in [4], in which an embedded and triangulated graph K of genus γ is partitioned by means of fundamental cycles² with respect to a spanning tree T of K . The choice of an appropriate set of fundamental cycles is done by construction and manipulation of an auxiliary graph, referred to as *separation graph*. The edges of the separation graph correspond to fundamental cycles in K , and the removal of an edge from the separation graph corresponds, in terms of the connectivity and the weight, to the removal of the corresponding fundamental cycle from K . Consequently, we are able to construct a t -separator of K consisting of fundamental cycles by finding a set of edges that partitions the separation graph in

²A fundamental cycle is a cycle consisting of a single non-tree edge (v_1, v_2) plus the two paths in T from v_1 and v_2 to their lowest common ancestor.

the required way. If the genus of K is small, then its separation graph is much simpler (e.g., for planar graphs, it is a tree), and its partitioning is easy and efficient. The next lemma states results obtained by applying the separation graph technique on K . The lemma follows directly from the presentation in [4].

Lemma 1 *Let K be an embedded and triangulated graph of genus γ with nonnegative weights on its vertices and a spanning tree T . There exists a t -separator C of G that satisfies the following:*

- (a) *The separator C consists of at most $2\gamma + 3/t$ fundamental cycles.*
- (b) *Any of the components of $K \setminus C$ is adjacent to at most $2\gamma + 3$ cycles in C .*

Such a separator C can be constructed in $O(|K| \log |K|)$ time.

Next, we present our first result concerning existence and construction of “low-cost” t -separators for embedded graphs of genus g with weights and costs. Lemma 1 by itself is not sufficient for constructing such separators, since fundamental cycles in the graph to be partitioned may have large cost. To handle this, we use a technique called *slicing*, in which the input graph is “sliced” into subgraphs with “short” spanning trees in terms of cost (see [9]). Then, the resulting subgraphs are further partitioned by means of fundamental cycles applying Lemma 1(a).

We denote the sum of the squares of the costs of the vertices of G by $\sigma(G)$, i.e., $\sigma(G) = \sum_{v \in V} c(v)^2$.

Theorem 1 *Let G be an embedded graph of genus g with weights and costs assigned to its vertices. For any $t \in (0, 1)$, there exists a t -separator S whose cost is at most $4\sqrt{(2g + 3/t)\sigma(G)}$. Such a separator can be constructed in $O(|G| \log |G|)$ time.*

Proof The theorem is proved by constructing a t -separator S whose cost is as required. The separator S is constructed in two phases. In the first phase, we “slice” the graph into subgraphs. In the second phase, we use Lemma 1 to obtain a t -separator. Initially, we set the separator $S = \emptyset$.

Phase I: (Slicing) We add a dummy vertex ρ of zero weight and zero cost to G into one of the faces of G and connect it to the vertices of this face. Then, we convert the graph into a directed one by replacing each edge by a pair of edges with opposite directions. Using Dijkstra’s algorithm, we construct a single-source shortest path (SSSP) tree T rooted at ρ , assuming that the cost of an oriented edge (u, v) equals to $c(v)$. The radius of the tree T is defined as the maximum distance between ρ and any vertex of T and is denoted by $r(T)$. For any real $x \in (0, r(T))$, we define a set of vertices $L(x)$ called *level* as follows. A vertex v is in $L(x)$ if its distance to ρ is at least x and the distance of its predecessor in T to ρ is less than x . Intuitively, a level in T consists of the vertices that are at the same cost-distance from the root ρ . The removal of the vertices of a level divides the graph into two disjoint subgraphs induced by the vertices “below” and “above” that level.

Next we define a set of levels in T which we use for slicing the graph into several components, where each component has a “small radius.” Let $h = \sqrt{\frac{\sigma(G)}{4(2g+3/t)}}$. We

apply the method described in [9] and compute a set of levels $\mathcal{L}_h$, so that their removal partitions G into components with SSSP trees of radius not exceeding $2h$. The total cost of the vertices in the set of levels $\mathcal{L}_h$ does not exceed $\sigma(G)/h$. The vertices in the levels $\mathcal{L}_h$ are inserted into S .

Phase II: (t -separator) In this phase, each “heavy” component K , i.e., $w(K) > tw(G)$, of the graph $G \setminus \mathcal{L}_h$, is further partitioned by fundamental cycles as stated in Lemma 1 with a parameter $t_K = tw(G)/w(K)$. The resulting separator $S(K)$ is inserted in S . By the construction in Phase I and by Lemma 1, the cost of the separator $S(K)$ is bounded by $c(S(K)) \leq 4h(2\gamma(K) + 3/t_K)$, where $\gamma(K)$ is the genus of the embedding of K . Therefore, for the cost of the obtained separator, we have

$$\begin{aligned} c(S) &\leq \sigma(G)/h + \sum_{w(K) > tw(G)} 4h(2\gamma(K) + 3w(K)/tw(G)) \\ &\leq \sigma(G)/h + 4h(2g + 3/t) = 4\sqrt{(2g + 3/t)\sigma(G)}. \end{aligned} \quad (1)$$

We have used that the genus of a graph is at most the sum of the genres of its components (see [12]). The time required for the construction of the separator S is dominated by the time for the construction of the SSSP tree T of G , which can be done in $O(|G| \log |G|)$ time, e.g., using Dijkstra’s algorithm. $\square$

In many applications, including our query algorithms presented later in this paper, it is useful to construct separators possessing certain additional properties. We need t -separators that partition the graph into components with “small-cost” boundaries. To obtain such separators, we first construct a “low-cost” t -separator using the approach described in Theorem 1 and then further partition components with costly boundaries applying separation graph technique (Lemma 1(b)). Similar separators were first constructed and used for planar graphs with uniform costs by Frederickson in [26]. The separators there are obtained recursively using $2/3$ -separators from [32]. The separation graph technique allows for direct and more efficient construction of the required separators and is applicable for graphs with costs.

Recall the following from Sect. 1.2. Let $V_1, \dots, V_k, S$ be the partition defined by a t -separator S . Note that a vertex in a set V_i , for some $1 \leq i \leq k$, can be adjacent to vertices in $V_i \cup S$ only. The subset of vertices in S that are adjacent to vertices in V_i is called *boundary* of V_i (or of the component induced by V_i) and is denoted by ∂V_i . The partition $V_1, \dots, V_k, S$ defined by a t -separator S is called *B-regular* (or simply *regular*), where B is a real number, if $c(\partial V_i) \leq B$ for $i = 1, \dots, k$.

Theorem 2 *Let G be an embedded graph of genus g with maximum degree three and with weights and costs assigned to its vertices. For any $t \in (0, 1)$, there exists a t -separator S , that defines a $2B$ -regular partition of G with $B = \sqrt{(g+1)t\sigma(G)}$, whose cost is $O(\sqrt{(g+1)\sigma(G)/t})$. Such a separator can be constructed in $O(|G| \log |G|)$ time.*

Proof We set $18h = B/(g+1) = \sqrt{t\sigma(G)/(g+1)}$ and apply Phase I as described in the proof of Theorem 1 above. Thus, we compute a set of levels $\mathcal{L}_h$, whose removal

partitions the graph into components, whose spanning trees have radii (in terms of cost) bounded by $2h$. For the total cost of the vertices in $\mathcal{L}_h$, we have

$$c(\mathcal{L}_h) \leq \sigma(G)/h. \quad (2)$$

Then, we apply Phase II and obtain a set C_1 of fundamental cycles such that the set of vertices in $S_1 = \mathcal{L}_h \cup C_1$ is a t -separator for G . For the cost $c(C_1)$ of the vertices in the cycles C_1 , we have

$$c(C_1) \leq 4h(2g + 3/t). \quad (3)$$

In addition, from Lemma 1(b) and the bound on the radii of the components resulting after the removal of $\mathcal{L}_h$, it follows that the cost of the vertices on the boundary of any component defined by S_1 which are not in $\mathcal{L}_h$ is at most $4h(2g + 3) \leq (2/3)B$.

The set S_1 is a t -separator, but it may not induce a $2B$ -regular partition since there might be components in $G \setminus S_1$ with boundaries whose costs exceed $2B$. To obtain a $2B$ -regular partition, we consider each component K of $G \setminus S_1$ with a “high-cost” boundary and further partition it obtaining $2B$ -regular partition as follows.

Let K be a component of $G \setminus S_1$ such that $c(\partial K) > 2B$. By our construction, we have

$$c(\partial K \setminus \mathcal{L}_h) \leq (2/3)B, \quad \text{and so} \quad c(\partial K \cap \mathcal{L}_h) > (4/3)B. \quad (4)$$

We consider the subgraph of G induced by the set of vertices $V(K) \cup \partial K$ and denote it by $\tilde{K}$. Let the embedding of $\tilde{K}$ have genus $\gamma(K)$. We assign new weights $w_1(v)$ and costs $c_1(v)$ to the vertices of $\tilde{K}$ as follows. We denote by $\partial' K$ the set of the vertices in ∂K that belong to $\mathcal{L}_h$, i.e., $\partial' K = \partial K \cap \mathcal{L}_h$. The new costs of the vertices in ∂K are set to zero, and the new costs of the vertices in K equal its original cost, i.e., for $v \in K$, we have $c_1(v) = c(v)$ and for $v \in \partial K$, $c_1(v) = 0$.

The new weight of a vertex v in K is the sum of the costs of the vertices in $\partial' K$ that are adjacent to v . The weights of the vertices in ∂K and the vertices in K not adjacent to $\partial' K$ are set to zero. By this definition, and since the maximum degree of G is three, we have $w_1(\tilde{K}) \leq 3c(\partial' K)$.

Then, we set $t_K = (2/3)B/w_1(\tilde{K})$ and compute a t_K -separator of $\tilde{K}$ using Lemma 1. We denote this separator by $C(K)$. For the cost of this separator, we have

$$c(C(K)) \leq 4h(2\gamma(K) + 9w_1(\tilde{K})/2B) \leq 8h\gamma(K) + (c(\partial K \cap \mathcal{L}_h)/(g + 1)). \quad (5)$$

Let us consider any component K_1 of $\tilde{K} \setminus C(K)$ and estimate the cost of its boundary in G . The boundary of K_1 can contain vertices from the cycles in $C(K)$ and vertices from ∂K only. We estimate first the cost of the vertices in ∂K_1 that belong to ∂K . By (4), it follows that the cost of the vertices in $(\partial K_1 \cap \partial K) \setminus \mathcal{L}_h$ is at most $(2/3)B$. By our definitions, the total cost of the vertices from $\mathcal{L}_h$ in the boundary ∂K_1 is bounded by $w_1(K_1)$, and thus it does not exceed $t_K w_1(\tilde{K}) = (2/3)B$.

From Lemma 1(b) and the estimate on the radius of the spanning tree in K it follows that the total cost of the vertices from $C(K)$ in the boundary ∂K_1 is at most $4h(2\gamma(K) + 3) \leq (2/3)B$. We have shown that $c(\partial K_1) \leq 2B$. Thereby, we define the separator S to be the union of S_1 and the separators $C(K)$ computed for all components K with $c(\partial K) > 2B$. Clearly, S is a t -separator that induces a $2B$ -regular partition of G .

Next, we estimate the cost of the separator S . We use (2), (3), (5), and the definitions of B and h and obtain

$$\begin{aligned} c(S) &\leq c(S_1) + \sum_{c(\partial K) > 2B} c(C(K)) \\ &\leq \sigma(G)/h + 4h(2g + 3/t) + \sum_{c(\partial K) > 2B} 8h\gamma(K) + (c(\partial K \cap \mathcal{L}_h)/(g + 1)) \\ &\leq \sigma(G)/h + 16hg + 12h/t + 3c(\mathcal{L}_h)/(g + 1) \\ &< \left(\frac{19g + 77}{g + 1}\right) \sqrt{(g + 1)\sigma(G)/t} \leq 77\sqrt{(g + 1)\sigma(G)/t}. \end{aligned}$$

The estimate on the time for the construction of the separator S is straightforward. $\square$

3 Approximating Shortest Paths

First, we adapt the discretization scheme presented in [8] and establish properties which are used later for answering shortest path queries. Let P be a polyhedral surface in three-dimensional Euclidean space consisting of n triangular faces $f_1, \dots, f_n$. Each face f_i has an associated positive weight w_i , representing the cost of travelling a unit Euclidean distance inside it. The cost of travelling 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 from which they inherit their weight. 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 of π in f_i . Given two points a and b in P , a path of minimum cost joining a and b is called *shortest path* between a and b and is denoted by $a \overset{P}{\rightsquigarrow} b$. It is known that the shortest paths in P consist of segments with end-points on the edges of P . The cost $\|a \overset{P}{\rightsquigarrow} b\|$ of this path is referred to as *distance* between a and b and is denoted by $\text{dist}_P(a, b)$.

3.1 Approximation Graph

Let ε be a real number in $(0, 1)$. As part of our algorithms, we will be constructing an *approximation graph* $G_\varepsilon = (V_\varepsilon, E_\varepsilon)$, which is a supergraph of the approximation graph constructed in [8], whose nodes correspond to geometric objects, namely, Steiner points and vertex vicinities of “small” radius in P (Fig. 1). Costs will be assigned to the edges of G_ε so that the distances between nodes in G_ε approximate the distances between corresponding objects in P .

The construction of the graph G_ε closely follows the scheme described in [8]. The approximation graphs we define here and in [8] have the same set of nodes, but the graph here has some additional edges. Moreover, the definition of the costs assigned to the edges of G_ε is slightly different. To emphasize these differences, we give a short description of the approximation graph we use here.

As detailed in [8], around each vertex v of P , we define a “small” star-shaped polygon $\mathcal{E}(v)$ called *vertex vicinity*. More precisely, $\mathcal{E}(v)$ is contained within the

Fig. 1 Steiner points on bisectors of a face f and the vertex vicinities of the vertices of f are illustrated

union of the triangles incident to v , and its intersection with each of these triangles is a “small” isosceles triangle with side length $\varepsilon r(v)$, where $r(v)$ is called *weighted radius* of v . The weighted radius $r(v)$ is defined by

$$r(v) = \frac{w_{\min}(v)}{7w_{\max}(v)}d(v), \quad (6)$$

where $d(v)$ 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 nodes of G_ε are of two types depending on the object in P they represent (Fig. 1). For each vertex of P and its vertex vicinity, we define a node representing them in G_ε and call them vertex vicinity nodes. Steiner point nodes (or simply Steiner points) represent Steiner points inserted in P . Steiner points are placed along the bisectors of the angles of the faces of P forming a geometric progression with ratios depending on ε and on the geometry of P as detailed in [8]. The approximation properties of Steiner points are stated in the following lemma.

Lemma 2 [8] *Let x and y be points lying on two different edges of a face f of P and outside vertex vicinities. There is a Steiner point p in f such that $|xp| + |py| \leq (1 + \varepsilon/2)|xy|$.*

Upper bounds on the number of nodes of G_ε , i.e., Steiner points inserted in P and vertex vicinity nodes are given by the following lemma.

Lemma 3 [8]

- (a) *The number of nodes of G_ε incident to a bisector ℓ of an angle α at a vertex v is bounded by $C(\ell) \frac{1}{\sqrt{\varepsilon}} \log_2 \frac{2}{\varepsilon}$, where the constant $C(\ell) < \frac{2}{\sin \alpha} \log_2 \frac{2|\ell|}{r(v)}$.*

- (b) The total number of nodes of G_ε is less than $C(P) \frac{n}{\sqrt{\varepsilon}} \log \frac{2}{\varepsilon}$, where $C(P)$ is the average of the constants $C(\ell)$ over all $3n$ bisectors of the angles of the faces of P . The constant $C(P)$ is less than $5\Gamma \log_2 2L$, where L is the maximum of the ratios $|\ell|/r(v(\ell))$ over all bisectors ℓ ($v(\ell)$ being the vertex incident to ℓ), and Γ is the average of the reciprocals of the sines of the face angles in P , i.e., $\Gamma = \frac{1}{3n} \sum_{i=1}^{3n} \frac{1}{\sin \alpha_i}$.

To define the edges of G_ε , we introduce the notion of *face neighbourhood* for points in P . The face neighbourhood of a vertex of P is the union of the triangles incident to that vertex. The face neighbourhood of a point in a face f of P consists of the union of f and its *neighbouring* faces, where two faces are neighbours if they share an edge of P . The face neighbourhood of a point a is denoted by $\mathcal{N}(a)$. The face neighbourhood of a node p of G_ε is the face neighbourhood of its representation (Steiner point or vertex) in P .

The edges of G_ε are defined as follows. A node p of G_ε is connected to all nodes, whose representations are incident with its face neighbourhood $\mathcal{N}(p)$. To define costs of the edges of G_ε , we use the notion of *local paths*. A path in P is called *local* if it intersects at most two faces. The cost $c(p, q)$ of an edge (p, q) in G_ε is defined as the cost of the *local shortest path* restricted to lie in the intersection of their face neighbourhoods $\mathcal{N}(p) \cap \mathcal{N}(q)$. Thereby, the cost $c(p, q)$ of an edge (p, q) joining a pair of Steiner points is defined as the cost of the shortest path restricted to lie in the union of the triangles³ containing p and q . The cost of an edge between a vertex vicinity node and a Steiner point is the cost of the shortest path restricted to lie in the triangle containing the Steiner point. The cost of an edge between two vertex vicinity nodes is the cost of the segment along the edge of P joining these two vertex vicinities.

3.2 Approximation Properties of G_ε

The paths in the approximation graph G_ε are called *discrete paths*. The cost $c(\pi_{G_\varepsilon}(p, q))$ of a discrete path $\pi_{G_\varepsilon}(p, q)$ is the sum of the costs of its edges. Let $p \xrightarrow{G_\varepsilon} q$ denote a shortest path in G_ε between a pair of nodes p and q of G_ε . As is shown below, in general, the cost $c(p \xrightarrow{G_\varepsilon} q)$ of a shortest discrete path is an ε -approximation of the distance $\|\tilde{p} \xrightarrow{P} \tilde{q}\|$, where $\tilde{p}$ and $\tilde{q}$ are points in P incident to the objects represented by p and q . A discrete path $\pi_{G_\varepsilon}(p, q)$ between nodes p and q can be embedded naturally in P as follows. First, each node on that path is embedded into the object it represents, i.e., either a Steiner point or a vertex vicinity. Then, each edge of $\pi_{G_\varepsilon}(p, q)$ is embedded into the local shortest path between the objects representing its end-nodes. As a result, we obtain a sequence of vertex vicinities joined by polygonal paths in P . Finally, each vertex vicinity is replaced with a two (or one) segment path through the vertex of P in that vicinity. We refer to this embedding of a discrete path $\pi_{G_\varepsilon}(p, q)$ into P as *natural embedding* and denote it by $\tilde{\pi}_{G_\varepsilon}(p, q)$. By our definitions, the cost in P of the natural embedding $\tilde{\pi}_{G_\varepsilon}(p, q)$ of a discrete path

³Recall that an edge of P belongs to the triangle from which it inherits its weight.

minus the cost of its portions inside vertex vicinities equals the cost of $\pi_{G_\varepsilon}(p, q)$ in G_ε .

The following theorem states the time complexity for solving SSSP problem in the approximation graph G_ε .

Theorem 3 [8] *The SSSP problem in the approximation graph G_ε can be solved in $O(|V_\varepsilon| \log |V_\varepsilon|) = O(\frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time.*

Next, we discuss how the approximation graph G_ε can be used to approximate distances and shortest paths in P . For the sake of this discussion, we assume that the distances and shortest paths between any pair of nodes of G_ε are given to us. Let a and b be arbitrary points in P . If a and b lie in neighbouring triangles and the shortest path $a \overset{P}{\rightsquigarrow} b$ between them is a local path (i.e., stays inside the quadrilateral formed by the union of their triangles), then we can report the exact path in constant time. So, we concentrate on the approximation of shortest paths that cross more than two faces.

Naturally, we consider paths of the form $\{a \overset{P}{\rightsquigarrow} p \overset{G_\varepsilon}{\rightsquigarrow} q \overset{P}{\rightsquigarrow} b\}$ and then approximate $\text{dist}_P(a, b)$ by the minimum of $\|a \overset{P}{\rightsquigarrow} p\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|q \overset{P}{\rightsquigarrow} b\|$ taken over all choices of nodes p and q in G_ε . As shown below, we can obtain the desired approximation by taking the minimum not over all pairs of nodes in G_ε but only over pairs p and q such that $p \in \mathcal{N}(a)$ and $q \in \mathcal{N}(b)$. Moreover, we show that it suffices to compute the local shortest paths between a and p and between b and q . We denote these local shortest paths by $a \overset{\mathcal{N}(a)}{\rightsquigarrow} p$ and $q \overset{\mathcal{N}(b)}{\rightsquigarrow} b$ and define *approximate discrete paths* between pairs of points in P as follows.

Definition 1 A path between a pair of points a and b in P is called *approximate discrete path* if it is either a shortest local path joining a and b or a path of the form

$$\left\{ a \overset{\mathcal{N}(a)}{\rightsquigarrow} p \overset{G_\varepsilon}{\rightsquigarrow} q \overset{\mathcal{N}(b)}{\rightsquigarrow} b \right\}, \quad (7)$$

where $p \in \mathcal{N}(a)$, $q \in \mathcal{N}(b)$. The cost of an approximate discrete path is $\|a \overset{\mathcal{N}(a)}{\rightsquigarrow} p\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|q \overset{\mathcal{N}(b)}{\rightsquigarrow} b\|$ or its cost in P if it is a local path. The cost of a shortest approximate discrete path between a and b is called *approximate distance* between a and b and is denoted by $\text{dist}_{G_\varepsilon}(a, b)$.

Note that by this definition, the approximate distance $\text{dist}_{G_\varepsilon}(a, b)$ between points a and b lying in neighbour triangles is the minimum of the cost of the shortest local path between a and b and the cost of any path of the form (7). The next theorem establishes the relation between approximate distances and the distances in P .

Theorem 4 *For any pair of points a and b in P , one of the following two holds, either*

$$(a) \quad (1 - 2\varepsilon)\text{dist}_P(a, b) \leq \text{dist}_{G_\varepsilon}(a, b) \leq (1 + 2\varepsilon)\text{dist}_P(a, b). \quad (8)$$

- (b) *There is a vertex v of P such that the points a and b , together with any shortest path between them in P , are in the face neighbourhood $\mathcal{N}(v)$. Moreover,*

$$\text{dist}_P(a, b) - \varepsilon(w(a) + w(b))r(v) < \text{dist}_{G_\varepsilon}(a, b) < (1 - \varepsilon)\text{dist}_P(a, b), \quad (9)$$

where $w(a)$ and $w(b)$ are the weights of the faces containing a and b , respectively, and $r(v)$ is the weighted radius of v defined by (6).

Proof There are three possible cases depending on the coincidence of the points a and b with representatives of the nodes of G_ε .

Case 1: Let a and b be incident with the representations of nodes p and q of G_ε . Then, by our definitions, $\text{dist}_{G_\varepsilon}(a, b) = c(p \xrightarrow{G_\varepsilon} q)$. Following the proof of Theorem 3.2 in [8], we obtain

$$c(p \xrightarrow{G_\varepsilon} q) \leq (1 + \varepsilon)\text{dist}_P(a, b), \quad (10)$$

which is stronger than the right inequality in (8).

To estimate $\text{dist}_{G_\varepsilon}(a, b)$ from below, we assume that $\text{dist}_{G_\varepsilon}(a, b) < \text{dist}_P(a, b)$, otherwise, the lower bound in (a) holds. We consider the natural embedding of the path $p \xrightarrow{G_\varepsilon} q$. As discussed above, the cost of the path $p \xrightarrow{G_\varepsilon} q$ in G_ε equals the cost of its natural embedding in P minus the total cost of the portions of this embedding inside vertex vicinities. By the definitions of the vertex vicinities and the cost of the edges in G_ε , the total cost of these portions does not exceed $\varepsilon c(p \xrightarrow{G_\varepsilon} q)$, provided that the path $p \xrightarrow{G_\varepsilon} q$ contains none or at least two vertex vicinity nodes. Thus, in these cases, we have

$$\begin{aligned} \text{dist}_P(a, b) &\leq c(p \xrightarrow{G_\varepsilon} q) + \varepsilon c(p \xrightarrow{G_\varepsilon} q) = \text{dist}_{G_\varepsilon}(a, b) + \varepsilon \text{dist}_{G_\varepsilon}(a, b) \\ &< \text{dist}_{G_\varepsilon}(a, b) + \varepsilon \text{dist}_P(a, b), \end{aligned}$$

where we used the assumption $\text{dist}_{G_\varepsilon}(a, b) < \text{dist}_P(a, b)$. The latter implies

$$(1 - \varepsilon)\text{dist}_P(a, b) < \text{dist}_{G_\varepsilon}(a, b), \quad (11)$$

which is stronger than the left inequality in (8).

It remains to consider the case where the path $p \xrightarrow{G_\varepsilon} q$ contains exactly one vertex vicinity node, say corresponding to a vertex v of P . In this case, from the definitions it follows that $\text{dist}_P(a, b) \leq \text{dist}_{G_\varepsilon}(a, b) + \varepsilon(w(a) + w(b))r(v)$. This implies the lower bound in (a), provided that $\text{dist}_P(a, b) \geq (w(a) + w(b))r(v)$. If $\text{dist}_P(a, b) < (w(a) + w(b))r(v)$, then (b) holds.

Case 2: Consider now the case where a and b are neither Steiner points nor inside vertex vicinities. If the points a and b lie in neighbour triangles and the shortest path between them is a local path, then by our definitions, $\text{dist}_{G_\varepsilon}(a, b) = \text{dist}_P(a, b)$, and the theorem holds. So, we assume below that any shortest path between a and b intersects more than two faces.

First, we prove the upper bound on $\text{dist}_{G_\varepsilon}(a, b)$ by constructing an approximate discrete path $\{a \xrightarrow{N(a)} p \xrightarrow{G_\varepsilon} q \xrightarrow{N(b)} b\}$ whose cost is at most $(1 + 2\varepsilon)\text{dist}_P(a, b)$. Let

Fig. 2 A shortest path $\tilde{\pi} = a \xrightarrow{P} b = \{a, x(a), y(a) \xrightarrow{P} x(b), y(b), b\}$ between a and b is approximated by an approximate discrete path (dashed) $\{a \xrightarrow{N(a)} p \xrightarrow{G_\varepsilon} q \xrightarrow{N(b)} b\}$

$\tilde{\pi} = a \xrightarrow{P} b$ be a shortest path between a and b in P . The path $\tilde{\pi}$ consists of segments with end-points on the edges of P . We call these points *bending points* of the path $\tilde{\pi}$. By our assumption, the path $\tilde{\pi}$ intersects more than two faces. There are two subcases to consider; one where $\tilde{\pi}$ remains entirely inside $\mathcal{N}(a) \cup \mathcal{N}(b)$, and the other where the path $\tilde{\pi}$ is not entirely in $\mathcal{N}(a) \cup \mathcal{N}(b)$. Here, we discuss the latter case, and the other one will trivially follow from its proof.

We denote by $y(a)$ the first bending point on $\tilde{\pi}$ when traversed from a to b , where the path exits $\mathcal{N}(a)$ (see Fig. 2). Similarly, let $x(b)$ be the last bending point on $\tilde{\pi}$ when traversed from a to b , where the path enters $\mathcal{N}(b)$. Furthermore, we denote by $x(a)$ the bending point on $\tilde{\pi}$ preceding $y(a)$ and by $y(b)$ the bending point succeeding $x(b)$.

Let us consider first, the case where none of the points $x(a)$, $y(a)$, $x(b)$, and $y(b)$ is inside vertex vicinity. By their choice, $x(a)$ and $y(a)$ lie on two different sides of the face f containing the segment $(x(a), y(a))$. We define p to be a Steiner point in f such that

$$|x(a)p| + |py(a)| \leq (1 + \varepsilon/2)|x(a)y(a)|. \quad (12)$$

The existence of such a Steiner point is shown in Lemma 2. Similarly, let q be a Steiner point inside the face containing $(x(b), y(b))$ and such that

$$|x(b)q| + |qy(b)| \leq (1 + \varepsilon/2)|x(b)y(b)|. \quad (13)$$

If $x(a) = x(b)$ and $y(a) = y(b)$, we choose $p = q$. Now we have defined an approximate discrete path $\pi(a, b) = \{a \xrightarrow{N(a)} p \xrightarrow{G_\varepsilon} q \xrightarrow{N(b)} b\}$ and estimate its cost.

We consider the case where $(x(a), y(a)) \neq (x(b), y(b))$ (Fig. 2). From the definition of the points $x(a)$ and $y(b)$ and from the triangle inequality we have

$$\|a \xrightarrow{N(a)} p\| \leq \|a \xrightarrow{P} x(a)\| + \|x(a)p\| \quad (14)$$

and

$$\|q \xrightarrow{N(b)} b\| \leq \|y(b) \xrightarrow{P} b\| + \|qy(b)\|. \quad (15)$$

Again, from the triangle inequality and the validity of (10) for the path $p \xrightarrow{G_\varepsilon} q$ we obtain

$$c(p \xrightarrow{G_\varepsilon} q) \leq (1 + \varepsilon)(\|y(a) \xrightarrow{P} x(b)\| + \|py(a)\| + \|x(b)q\|). \quad (16)$$

Then, the cost of the path $\pi(a, b)$ is estimated by summing (14), (16), and (15) and using (12) and (13):

$$\begin{aligned} c(\pi(a, b)) &= \|a \overset{N(a)}{\rightsquigarrow} p\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|q \overset{N(b)}{\rightsquigarrow} b\| \\ &\leq \|a \overset{P}{\rightsquigarrow} x(a)\| + (1 + \varepsilon)\|y(a) \overset{P}{\rightsquigarrow} x(b)\| + \|y(b) \overset{P}{\rightsquigarrow} b\| \\ &\quad + (1 + \varepsilon/2)\|x(a)y(a)\| + (1 + \varepsilon/2)\|x(b)y(b)\| \\ &\quad + \varepsilon(\|py(a)\| + \|x(b)q\|) \leq (1 + 2\varepsilon)\text{dist}_P(a, b). \end{aligned}$$

The case where segments $(x(a), y(a)) = (x(b), y(b))$ is simpler. The cases where some of the points $x(a)$, $y(a)$, $x(b)$, and $y(b)$ are inside vertex vicinities are handled by the same technique.

Next, we estimate $\text{dist}_{G_\varepsilon}(a, b)$ from below. Again, we assume that $\text{dist}_{G_\varepsilon}(a, b) < \text{dist}_P(a, b)$. Otherwise, statement (a) of the theorem holds. We consider any approximate discrete path of the form $\{a \overset{N(a)}{\rightsquigarrow} p \overset{G_\varepsilon}{\rightsquigarrow} q \overset{N(b)}{\rightsquigarrow} b\}$. If p is a vertex vicinity node, we denote by $\tilde{p}$ the vertex in that vicinity. Correspondingly, $r(\tilde{p})$ is the weighted radius of $\tilde{p}$ defined by (6). If p is a Steiner point node, then $\tilde{p}$ denotes the corresponding Steiner point, and we assume $r(\tilde{p}) = 0$. We use analogous notation $\tilde{q}$ and $r(\tilde{q})$ for q . From our definitions and using this notation, we have

$$\|a \overset{N(a)}{\rightsquigarrow} p\| \geq \|a \overset{P}{\rightsquigarrow} \tilde{p}\| - \varepsilon w(a)r(\tilde{p}) \quad \text{and} \quad \|q \overset{N(b)}{\rightsquigarrow} b\| \geq \|\tilde{q} \overset{P}{\rightsquigarrow} b\| - \varepsilon w(b)r(\tilde{q}). \quad (17)$$

Then, for the cost of the path $\{a \overset{N(a)}{\rightsquigarrow} p \overset{G_\varepsilon}{\rightsquigarrow} q \overset{N(b)}{\rightsquigarrow} b\}$, which is $\text{dist}_{G_\varepsilon}(a, b)$, we obtain

$$\begin{aligned} &\|a \overset{N(a)}{\rightsquigarrow} p\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|q \overset{N(b)}{\rightsquigarrow} b\| \\ &\geq \|a \overset{P}{\rightsquigarrow} \tilde{p}\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|\tilde{q} \overset{P}{\rightsquigarrow} b\| - \varepsilon(w(a)r(\tilde{p}) + w(b)r(\tilde{q})). \end{aligned} \quad (18)$$

In the case where

$$w(a)r(\tilde{p}) + w(b)r(\tilde{q}) \leq \|a \overset{P}{\rightsquigarrow} \tilde{p}\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|\tilde{q} \overset{P}{\rightsquigarrow} b\|,$$

we use (18) and (11) and obtain

$$\begin{aligned} &\|a \overset{N(a)}{\rightsquigarrow} p\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|q \overset{N(b)}{\rightsquigarrow} b\| \\ &\geq (1 - \varepsilon)(\|a \overset{P}{\rightsquigarrow} \tilde{p}\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|\tilde{q} \overset{P}{\rightsquigarrow} b\|) \\ &\geq (1 - \varepsilon)(\|a \overset{P}{\rightsquigarrow} \tilde{p}\| + (1 - \varepsilon)\text{dist}_P(\tilde{p}, \tilde{q}) + \|\tilde{q} \overset{P}{\rightsquigarrow} b\|) \\ &\geq (1 - \varepsilon)^2(\|a \overset{P}{\rightsquigarrow} \tilde{p}\| + \text{dist}_P(\tilde{p}, \tilde{q}) + \|\tilde{q} \overset{P}{\rightsquigarrow} b\|) \geq (1 - 2\varepsilon)\text{dist}_P(a, b), \end{aligned} \quad (19)$$

which implies the lower bound in (a).

Consider now the case where

$$w(a)r(\tilde{p}) + w(b)r(\tilde{q}) > \|a \overset{P}{\rightsquigarrow} \tilde{p}\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|\tilde{q} \overset{P}{\rightsquigarrow} b\|. \quad (20)$$

We show that, in this case, (b) holds. This inequality is possible only if one of the points $\tilde{p}$ or $\tilde{q}$ is a vertex of P , denoted by v , and the other is either a Steiner point inside the face neighbourhood of v or the same vertex. Furthermore, (20) implies that the points a and b and any shortest path $a \overset{P}{\rightsquigarrow} b$ stay in $\mathcal{N}(v)$.

In the case where $\tilde{p} = \tilde{q} = v$, we use (18) and obtain

$$\begin{aligned} \text{dist}_{G_\varepsilon}(a, b) &\geq \|a \overset{P}{\rightsquigarrow} v\| + \|v \overset{P}{\rightsquigarrow} b\| - \varepsilon(w(a) + w(b))r(v) \\ &\geq \text{dist}_P(a, b) - \varepsilon(w(a) + w(b))r(v), \end{aligned}$$

which proves the lower bound in (b).

If $\tilde{p} = v$ and $\tilde{q}$ is a Steiner point in $\mathcal{N}(v)$, then from (17) we have

$$\begin{aligned} \text{dist}_{G_\varepsilon}(a, b) &\geq \|a \overset{P}{\rightsquigarrow} v\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|\tilde{q} \overset{\mathcal{N}(b)}{\rightsquigarrow} b\| - \varepsilon w(a)r(v) \\ &\geq \|a \overset{P}{\rightsquigarrow} v\| + \|b \overset{P}{\rightsquigarrow} v\| - \varepsilon(w(a) + w(b))r(v), \end{aligned}$$

and the lower bound in (b) holds again.

The cases where just one of the points a or b is a Steiner point or incident to a vertex vicinity are treated analogously. $\square$

Next, we make some remarks on the quality of the approximation given by the approximate distances in the cases (a) and (b) of Theorem 4. If case (a) of the theorem applies, then the approximate distance $\text{dist}_{G_\varepsilon}(a, b)$ is an approximation of the distance $\text{dist}_P(a, b)$ with relative error of $\frac{|\text{dist}_{G_\varepsilon}(a, b) - \text{dist}_P(a, b)|}{\text{dist}_P(a, b)}$, which is bounded by 2ε . Case (b) of the theorem can be viewed as an exception covering a special situation where points a and b are “close” to each other and “near” a vertex v of P , meaning that any shortest path in P between them stays in the face neighbourhood $\mathcal{N}(v)$. Furthermore, in case (b), the approximate distance $\text{dist}_{G_\varepsilon}(a, b)$ is less than the actual distance $\text{dist}_P(a, b)$, and it is an approximation with relative error not exceeding $\varepsilon(w(a) + w(b))r(v)/\text{dist}_{G_\varepsilon}(a, b)$. Therefore, if $(w(a) + w(b))r(v) \leq 2\text{dist}_{G_\varepsilon}(a, b)$, the quality of the approximation is 2ε , same as in case (a). If the ratio $r(v)(w(a) + w(b))/\text{dist}_{G_\varepsilon}(a, b)$ is larger than $1/\varepsilon$, then the relative error could be as big as 1. For example, if points a and b are inside the vertex vicinity $\mathcal{E}(v)$, then $\text{dist}_{G_\varepsilon}(a, b)$ is zero, and the relative error is 1. Note that the conditions for the occurrence of case (b) and the presence of eventually large (compared to ε) relative error are easily detected by the position of the points a and b , the structure of the approximate discrete path, and the ratio $(w(a) + w(b))r(v)/\text{dist}_{G_\varepsilon}(a, b)$. Thus, if the approximation is not satisfactory, the exact shortest path restricted to lie inside $\mathcal{N}(v)$ can be computed. Straightforwardly, such computation can be done in time quadratic to the number of faces incident to v , which is efficient if the degree of v is small. Alternatively, additional preprocessing can be done for vertices with large degrees so that exact shortest paths in their face neighbourhoods can be answered efficiently. The above discussion is summarized in the next corollary.

Corollary 1 *The distance $\text{dist}_{G_\varepsilon}(a, b)$ approximates $\text{dist}_P(a, b)$ with relative error 2ε , except possibly when the case (b) of Theorem 4 applies and $(w(a) + w(b))r(v) > 2\text{dist}_{G_\varepsilon}(a, b)$.*

We conclude this section by a remark on the computation of approximate distances. The approximate distance $\text{dist}_{G_\varepsilon}(a, b)$ between a pair of points a and b can be computed as follows. We consider the approximate discrete paths of the form (7) and compute the minimum

$$\min(\|a \overset{\mathcal{N}(a)}{\rightsquigarrow} p\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|q \overset{\mathcal{N}(b)}{\rightsquigarrow} b\|) \quad (21)$$

over all pairs of nodes $p \in \mathcal{N}(a)$ and $q \in \mathcal{N}(b)$. In the case where a and b do not lie in neighbouring faces, this minimum is the approximate distance $\text{dist}_{G_\varepsilon}(a, b)$. In the case where the points a and b lie in neighbouring faces, we also need to consider the shortest local path between them.

4 Fixed Source Shortest Path Queries

In this section, we describe an algorithm that takes as input a point a in P , called *source*, and an approximation parameter $\varepsilon \in (0, 1)$ and constructs a data structure, called *Single-Source Queries* (SSQ), such that for any query point b , called *target*, the approximate distance $\text{dist}_{G_\varepsilon}(a, b)$ (and/or an approximate shortest path) from a to b is computed efficiently. (It is assumed that the query consists of the query point b and a pointer to the face of P containing b .) The algorithm uses the approximation graph G_ε defined and analyzed in the previous section as follows.

First, we construct the set of nodes of the approximation graph G_ε . Then, we augment G_ε so that a becomes a representation of a new node in G_ε . That is, we add a new node in G_ε whose representation in P is a and connect it to all nodes of G_ε , whose representations are incident to the face neighbourhood $\mathcal{N}(a)$ of the face containing a . A cost equal to the cost of the shortest local path between a and the representation of the corresponding node in $\mathcal{N}(a)$ is assigned to each of the new edges. This augmentation of the approximation graph is denoted by G_ε^a or simply G_ε , when no ambiguity arises. The augmentation takes time proportional to the number of the new edges, which by Lemma 3 is $O(\frac{1}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$. The result of Theorem 4 is extended straightforwardly to the graph G_ε^a and to the approximate distances $\text{dist}_{G_\varepsilon^a}(a, b)$, where b is an arbitrary point in P .

Next, we run the SSSP algorithm on G_ε^a with source a and store the SSSP tree and the distances as required. Then, for a query point b , we will find the shortest approximate discrete path from a to b and output its cost $\text{dist}_{G_\varepsilon^a}(a, b)$ as an approximation to the distance $\text{dist}_P(a, b)$ and the natural embedding of the path, if required.

Approximate discrete paths between the node a and a point b are either local shortest paths or paths of the form $\{a \overset{G_\varepsilon^a}{\rightsquigarrow} p \overset{\mathcal{N}(b)}{\rightsquigarrow} b\}$, where p is a node of G_ε^a incident to the face neighbourhood of b . Hence, the computation of the approximate distance $\text{dist}_{G_\varepsilon^a}(a, b)$ requires finding

$$\min_{p \in \mathcal{N}(b)} (\text{dist}_{G_\varepsilon^a}(a, p) + \|p \overset{\mathcal{N}(b)}{\rightsquigarrow} b\|) \quad (22)$$

and the cost of the shortest local path between a and b if they lie in neighbouring faces. The cost of the shortest local path, when necessary, is computed in constant time. So we concentrate on the computation of (22).

Fig. 3 An illustration of local Voronoi diagram $\text{LVD}(a, f', f)$. The face $f = ABC$ and the edge $e = AB$ adjacent to f and f' are shown. Dashed horizontal segments correspond to “events.” The query point b lies in the Voronoi cell (shown shaded) of the node $p(b)$

We have already computed and stored the distances $\text{dist}_{G_\varepsilon^a}(a, p)$ from a to all nodes $p \in G_\varepsilon^a$, and thus our task is reduced to finding a node $p(b)$ that minimizes (22) for a query point b . Next, we introduce the notion of Local Voronoi Diagrams between a pair of adjacent faces and show how that can be used in finding the node $p(b)$ as required in (22). More precisely, let f be a face of P , and let $\mathcal{N}(f)$ be its face neighbourhood consisting of at most three faces. Each of these faces in turn has Steiner points which have been placed on their corresponding bisectors. Our objective is to locate that Steiner point that satisfies (22). For each face $f' \in \mathcal{N}(f)$, we compute the Steiner point that satisfies this equation. We compute an additively weighted Voronoi Diagram of the set of Steiner points in f' , with respect to the distance $\text{dist}_{G_\varepsilon^a}$, and restrict this diagram to f . We call the resulting diagram as the *Local Voronoi Diagram* and denote it by $\text{LVD}(a, f', f)$, and its computation is illustrated next.

Let $f = ABC$, and, without loss of generality, let $e = AB$ be horizontal, where f is above AB , and A is to the left of B . Let f' be the face adjacent to e . Without loss of generality, let $p_1, p_2, \dots, p_k$ be the set of Steiner points in f' ordered according to their increasing distance from a . First, we will compute LVD of these points on e , denoted by $\text{LVD}(a, f', e)$, and then propagate it to the interior of the face f (Fig. 3).

The Voronoi diagram on e is computed in an incremental fashion, by considering Steiner points in increasing order of their distances to a . Assume that we have already computed $\text{LVD}(a, f', e)$ with respect to Steiner points $p_1, \dots, p_i$ (where, $i < k$) and now we wish to update this diagram to include the effect of the next Steiner point p_{i+1} . Due to the convexity and the continuity of the distance measure, Voronoi cell of p_{i+1} on e , with respect to Voronoi cells of $p_1, \dots, p_i$, will either be empty or will consist of a single interval; denote this interval by $[x^-, x^+]$. Let M be an endpoint of one of the intervals characterizing the Voronoi diagram of $p_1, \dots, p_i$. The relative position of M with respect to the interval $[x^-, x^+]$ on e can be computed in constant time. Hence, using binary search, we can find the location of the endpoints x^- and x^+ on e and update the diagram within this interval. Since each newly inserted element occupies at most one interval, it is easy to see that there are, in all, at most

$2k - 1$ intervals on e . They can be maintained in a search tree structure by naturally following their containment order.

Next, we propagate the Voronoi diagram to the interior of the face f . Note that $\text{LVD}(a, f', f)$ has the following combinatorial structure: Voronoi cells are connected, each cell has a nonempty intersection with e , and it has at most $2k - 1$ cells. We use plane sweep in a vertical direction starting at e . In place of computing the full diagram, we compute the diagram for a set of horizontal segments in f , parallel to e , that correspond to “events.” These events represent combinatorial changes in LVD, which in turn corresponds to the end of a Voronoi cell for a Steiner point. The events are maintained in a priority queue with respect to their increasing distance from e . Initially, the events are derived from adjacent intervals on e ; each interval endpoint is the intersection point of e and the bisector (i.e., locus of all points that are equidistant) between two Steiner points. During the sweep, intervals disappear one by one and the adjacency between the intervals changes, which gives rise to new events. For horizontal lines corresponding to two consecutive event points, there are only $O(1)$ changes in the LVDs, and the corresponding interval structure can be maintained using the persistent tree structure of [37]. It can be seen that LVD for horizontal lines corresponding to events in f can be encoded in $O(k \log k)$ time.

Given a query point $b \in f$, we locate the Voronoi region of $\text{LVD}(a, f', f)$ containing b as follows. First, we perform a binary search with respect to its distance from e to locate the two adjacent horizontal lines corresponding to event points in which b lies. Let these lines be e' and e'' , where e'' is further than e' with respect to distance from e (see Fig. 3). Using the persistent tree structure, we know the interval structure on e' and e'' , and we also know the set of Steiner points in f' that contribute these intervals. Note that the interval structure is the same between e' and e'' , except that one of the intervals has disappeared on e'' . Consider an interval endpoint in e'' and its corresponding endpoint in e' . Furthermore, we know the equation of the curve (bisector) that joins these two endpoints. In constant time, we can test whether b is to the left or to the right of this curve (for example, this can be accomplished by intersecting this curve with the horizontal line passing through b). Therefore, using binary search, we can locate the Voronoi cell containing b in $O(\log k)$ time.

As stated above, our objective was to locate that Steiner point that satisfies (22). To do this for each face $f' \in N(f)$, using the $\text{LVD}(a, f', f)$, we compute the Steiner point that satisfies this equation and then take the minimum among all the neighbouring faces. In the next lemma, we summarize the above discussion.

Lemma 4 *Let $p_1, \dots, p_k$ be the set of Steiner points and the vertices of P incident to $N(f)$, and let $\delta_i = \text{dist}_{G_e}(a, p_i)$ for $i = 1, \dots, k$. A data structure exists so that for a point $b \in f$, the minimum $\min_{1 \leq i \leq k} (\delta_i + \|p_i \overset{N(b)}{\rightsquigarrow} b\|)$ and the point for which it is achieved can be computed in $O(\log k)$ time. The size of the data structure is $O(k)$, and it can be constructed in $O(k \log k)$ time.*

We define $\text{SSQ}(a; P, \varepsilon)$ as a data structure to consist of the SSSP tree rooted at a in the approximation graph G_e^a , plus the collection of local Voronoi diagrams $\text{LVD}(a, f', f)$ for all pairs of neighbouring faces $f, f' \in P$. This data structure can

be used to answer single-source queries as follows. We assume that each query consists of a query point b on P and the face $f(b)$ containing b .

First, using the LVD data structure, we find the point $p(b)$ for which the minimum (22) is achieved. Then, if a and b lie in neighbour faces, we compute the shortest local path between a and b and output the approximate distance $\text{dist}_{G_\varepsilon^a}(a, b)$, which is the smaller of the two values. If an approximation path is required, we output the natural embedding of the approximate discrete path whose cost is $\text{dist}_{G_\varepsilon^a}(a, b)$. The quality of this approximation follows from Theorem 4 and Corollary 1. The above discussion is summarized in the next theorem.

Theorem 5 *Given a triangulated, weighted surface P with n faces, a point $a \in P$, and the set of nodes V_ε of G_ε , a data structure $\text{SSQ}(a; P, \varepsilon)$ of size $O(|V_\varepsilon|) = O(\frac{n}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$ exists, so that the approximate distance between a and a query point b in P can be found in $O(\log \frac{1}{\varepsilon})$ time. The data structure can be constructed in $O(|V_\varepsilon| \log |V_\varepsilon|) = O(\frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time.*

Proof Follows from the above discussion and using Lemma 3, Lemma 4, and Theorem 3. $\square$

5 Arbitrary Shortest Path Queries

In this section, we describe and analyze an algorithm for constructing a data structure called *All Pairs Queries* (APQ), such that approximate distance (and/or approximate shortest paths) queries between pairs of arbitrary points in P can be answered efficiently. In addition to the weighted polyhedral surface P and the approximation parameter ε , the algorithm takes as input a query time parameter q and outputs a data structure $\text{APQ}(P, \varepsilon; q)$, which can answer approximate distance queries in $O(q)$ time (the parameter q is within a range of values, and it is made precise in the theorem at the end of this section).

Our approach is based on the results obtained in the previous two sections. First, we compute the set of Steiner points and vertex vicinities defining the set of nodes V_ε of the approximation graph G_ε . Then, we consider the dual graph P^* of P , defined as follows. The set of nodes of P^* corresponds to the set of faces of P , and two nodes in P^* are joined by an edge if their corresponding faces are neighbours. To each node u of P^* , we assign the weight $w(u)$ equal to the number of nodes of G_ε that are incident to the face $f(u)$ in P . Furthermore, we assign the cost $c(u)$ equal to the number of nodes of G_ε that are incident to the face neighbourhood $\mathcal{N}(f(u))$. The total weight $w(P^*)$ of P^* and the value $\sigma(P^*)$, defined in Sect. 2, are estimated using Lemma 3 by

$$w(P^*) \leq C(P) \frac{n}{\sqrt{\varepsilon}} \log \frac{2}{\varepsilon} \quad \text{and} \quad \sigma(P^*) \leq \Gamma(P) \frac{n}{\varepsilon} \log^2 \frac{2}{\varepsilon}, \quad (23)$$

where $C(P) = \frac{1}{n} \sum_{f \in P} C(f)$ and $\Gamma(P) \leq \frac{16}{n} \sum_{f \in P} C^2(f)$. We observe that the weight of P^* and $\sigma(P^*)$ are related by

$$\sigma(P^*) \leq 16w^2(P^*) \leq 16n\sigma(P^*). \quad (24)$$

Table 4 All Pairs Queries Preprocessing algorithm**APQ_Preprocessing**

Input: A weighted polyhedral surface P of genus g and consisting of n triangular faces; an approximation parameter $\varepsilon \in (0, 1)$; query time parameter $q \leq \bar{q} = \frac{(g+1)^{2/3} n^{1/3}}{\sqrt{\varepsilon}}$.

Output: Data structure $\text{APQ}(P, \varepsilon; q)$.

Step 1. Define the set of nodes of G_ε by computing the Steiner points and vertex vicinities of P .

Step 2. Construct the dual graph P^* and assign weights and costs to its nodes as specified above.

Step 3. Set $t = \frac{q^2}{4(g+1)\sigma(P^*)\log^2 \frac{1}{\varepsilon}}$. Using Theorem 2, compute a t -separator S and construct the region partition of P induced by S .

Step 4. For each p that is a Steiner point or vertex of P incident to a face, which neighbours a face in S , compute and store $\text{SSQ}(p; P, \varepsilon)$ data structure.

Step 5. For each region R and for each Steiner point or vertex of P incident to a face in R , compute and store $\text{SSQ}(p; R, \varepsilon)$ data structure restricted to the faces in R .

Next, we choose a value $t \in (0, 1)$ depending on the input query time q and use Theorem 2 to construct a t -separator S that induces a regular partition of P^* . The separator S of P^* corresponds to a set of faces in P , which we refer to as *face separator* (or simply separator) and denote again by S . The face separator S partitions the surface P into *regions* that are unions of faces corresponding to the connected components of $P^* \setminus S$. The boundary ∂R of a region R is the set of triangles in S that neighbour faces in R .

Finally, we use Theorem 5 and construct a collection of SSQ data structures depending on the region partition defined by the face separator S . The collection of SSQ data structures and the region partition induced by S constitute the $\text{APQ}(P, \varepsilon; q)$ data structure. We refer to the construction of the APQ data structure as *preprocessing* and present it in more detail in Table 4. We denote the genus of the surface P by g . The query time parameter q will not exceed an upper bound $\bar{q}$ defined by $\bar{q} = \frac{(g+1)^{2/3} n^{1/3}}{\sqrt{\varepsilon}}$. In general, in shortest path query data structures based on separators, the query time bound is proportional to the total number of nodes present in the boundary of the largest region, as well as the number of nodes within a region, and the preprocessing time is proportional to the number of SSQ data structures that need to be constructed for the nodes in the separator.

In the next lemma, we estimate the time for the construction and the size of $\text{APQ}(P, \varepsilon; q)$ data structure.

Lemma 5 For any $q \leq \bar{q} = \frac{(g+1)^{2/3} n^{1/3}}{\sqrt{\varepsilon}}$, the construction of the data structure $\text{APQ}(P, \varepsilon; q)$ by the preprocessing algorithm takes $O(\frac{(g+1)n^2}{\varepsilon^{3/2}q} \log \frac{n}{\varepsilon} \log^4 \frac{1}{\varepsilon})$ time. The size of $\text{APQ}(P, \varepsilon; q)$ is $O(\frac{(g+1)n^2}{\varepsilon^{3/2}q} \log^4 \frac{1}{\varepsilon})$.

Proof We estimate the time following the steps as presented in Table 4. By Lemma 3, Step 1 takes $O(\frac{n}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$ time. The time for the execution of Step 2 is $O(n)$. According to Theorem 2, Step 3 is executed in $O(n \log n)$ time.

Recall that $t = q^2 / (4(g+1)\sigma(P^*) \log^2 \frac{1}{\varepsilon})$. To estimate the times for the execution of Steps 4 and 5, we use Theorems 2 and 5.

In Step 4, we compute $O(\sqrt{(g+1)\sigma(P^*)/t})$ SSQ data structures in $O(w(P^*) \times \log w(P^*))$ time each. This amounts to $O(\frac{(g+1)n^2}{\varepsilon^{3/2}q} \log \frac{n}{\varepsilon} \log^4 \frac{1}{\varepsilon})$ time, using the estimates in (23). The sizes of these data structures are bounded by $O(w(P^*))$.

In Step 5, for each region R defined by the separator S , we construct SSQ data structures restricted to R . The number of these structures in R equals the total weight of the faces in R , which we denote by $w(R)$. By Theorem 3, the construction of each of these SSQ data structures requires $O(w(R) \log w(R))$ time, and their size is bounded by $O(w(R))$. Overall, the time for the execution of Step 5 is bounded by $O(tw^2(P^*) \log w(P^*))$. The total size of the SSQ data structures stored in Step 5 is $O(tw^2(P^*))$.

Finally, we use $w^2(P^*) \leq n\sigma(P^*)$ and observe that under the assumption $q \leq \bar{q}$, the time used by Step 4 dominates the execution time used by Step 5 and, consequently, the total time of the whole preprocessing algorithm. Similarly, the total size of the SSQ data structures stored during Step 4 is bigger than the total size of the structures stored during Step 5. Thus, the asymptotic bounds on the time required by the algorithm and the size of the output are obtained by evaluation of the time and size used by Step 4 in terms of n and ε . We use estimates (23). $\square$

The APQ data structure built by the preprocessing algorithm can be used to answer approximate distance queries in the following way. Let a and b be points in P , and let $f(a)$ and $f(b)$ be the faces containing a and b , respectively. We use $\text{APQ}(P, \varepsilon; q)$ to compute the approximate distance $\text{dist}_{G_\varepsilon}(a, b)$, defined as the cost of the shortest approximate discrete path between a and b .

By definition, approximate discrete paths are either local shortest paths or paths of the form (7), which is $\{a \xrightarrow{\mathcal{N}(a)} p \xrightarrow{G_\varepsilon} q \xrightarrow{\mathcal{N}(b)} b\}$. In the case where $f(a)$ and $f(b)$ are neighbours, we compute the shortest local path between a and b . Then, we concentrate on finding the cost of a shortest path of the form (7), i.e., computing the minimum (21). Recall that $\text{APQ}(P, \varepsilon; q)$ contains a collection of SSQ data structures, as specified in Steps 4 and 5 in the preprocessing algorithm. If a data structure $\text{SSQ}(p'; P, \varepsilon)$ is present in $\text{APQ}(P, \varepsilon; q)$ for some node p' of G_ε , then we can use it and find the minimum cost of an approximate discrete path between a and b among the paths containing p' . We simply compute $\text{dist}_{G_\varepsilon}(a, p')$ and $\text{dist}_{G_\varepsilon}(p', b)$, using $\text{SSQ}(p'; P, \varepsilon)$ twice, and sum them up. By Theorem 5, this computation takes $O(\log \frac{1}{\varepsilon})$. Thereby, we introduce the notion of a *separating set* of nodes of G_ε for a pair of points a and b . A set of nodes of G_ε is called a separating set for a and b if any approximate discrete path of the form (7) between a and b contains a node from that set. Our query algorithm specifies a separating set A for a and b such that, for any $p' \in A$, the data structure $\text{SSQ}(p'; P, \varepsilon)$ is present in $\text{APQ}(P, \varepsilon; q)$, and then computes the minimum $\min_{p' \in A} (\text{dist}_{G_\varepsilon}(a, p') + \text{dist}_{G_\varepsilon}(p', b))$. Clearly, this minimum is the cost of the shortest approximate discrete path of the form (7). The time for this computation is $O(|A| \log \frac{1}{\varepsilon})$. In Table 5, we present the query algorithm.

Table 5 The APQ_Query algorithm answers approximate shortest path queries between arbitrary points on P in $O(q)$ time using APQ($P, \varepsilon; q$)

APQ_QUERY

Input: Query points a and b and faces $f(a)$ and $f(b)$ such that $a \in f(a)$ and $b \in f(b)$.

Output: The approximate distance $\text{dist}_{G_\varepsilon}(a, b)$.

Set $M_0 = M_1 = M_2 = \infty$.

Step 1. If $f(a)$ and $f(b)$ are neighbour faces, then compute the shortest local path $a \xrightarrow{f(a) \cup f(b)} b$ and assign its cost to M_0 .

Step 2. If either of the faces $f(a)$ or $f(b)$ is in the separator S , then define A to be the set of nodes of G_ε incident to the face neighbourhood $\mathcal{N}(a)$ or $\mathcal{N}(b)$, respectively.

Step 3. If neither of the faces $f(a)$ and $f(b)$ is in S , then define A to be the set of nodes of G_ε incident to the faces in the boundary $\partial R(a)$ of the region $R(a)$ containing $f(a)$.

Step 4. Use data structures $SSQ(p'; P, \varepsilon)$ and compute $M_1 = \min_{p' \in A} (\text{dist}_{G_\varepsilon}(a, p') + \text{dist}_{G_\varepsilon}(p', b))$.

Step 5. If $f(b) \in R(a)$, then define A_1 to be the set of nodes of G_ε incident to the face neighbourhood $\mathcal{N}(b)$. Use data structures $SSQ(p'; R(a), \varepsilon)$ and compute $M_2 = \min_{p' \in A_1} (\text{dist}_{G_\varepsilon}(a, p') + \text{dist}_{G_\varepsilon}(p', b))$.

Set $\text{dist}_{G_\varepsilon}(a, b) = \min(M_0, M_1, M_2)$ and output it.

If an approximate shortest path between a and b is required, we output the natural embedding of the approximate discrete path for which the minimum $\text{dist}_{G_\varepsilon}(a, b)$ is achieved. This can be done by using the SSSP trees stored in the corresponding SSQ data structure in time proportional to the size of this path. The next lemma establishes the correctness of the query algorithm and evaluates its running time.

Lemma 6 *The algorithm APQ_Query correctly computes the approximate distance $\text{dist}_{G_\varepsilon}(a, b)$. The running time of the algorithm is $O(\max(q, \frac{1}{\sqrt{\varepsilon}} \log^2 \frac{1}{\varepsilon}))$.*

Proof The correctness of the query algorithm follows from the observation that the cost of any approximate discrete path between a and b is given by the values M_0 , M_1 , and M_2 , depending on the position of the faces $f(a)$ and $f(b)$ with respect to the partition defined by the separator S . The running time of the algorithm is dominated by the times for the execution of Steps 4 and 5. As discussed above, these times are bounded by $O(|A| \log \frac{1}{\varepsilon})$ and $O(|A_1| \log \frac{1}{\varepsilon})$. By Lemma 3, $|A_1| = O(\frac{1}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$, and by Theorem 2 and the choice of t in the APQ_Preprocessing algorithm, we obtain $|A| \leq 2\sqrt{(g+1)t\sigma(P^*)} \leq \frac{q}{\log \frac{1}{\varepsilon}}$. $\square$

The results obtained in this section are summarized in the next theorem.

Theorem 6 *Let P be a weighted polyhedral surface of genus g and consisting of n triangular faces. Let $\varepsilon \in (0, 1)$ and $q \in (\frac{1}{\sqrt{\varepsilon}} \log^2 \frac{1}{\varepsilon}, \bar{q}]$, $\bar{q} = \frac{(g+1)^{2/3} n^{1/3}}{\sqrt{\varepsilon}}$. There exists a data structure APQ($P, \varepsilon; q$) such that approximate distance queries*

in P can be answered in $O(q)$ time. The structure $APQ(P, \varepsilon; q)$ is constructed in $O(\frac{(g+1)n^2}{\varepsilon^{3/2}q} \log \frac{n}{\varepsilon} \log^4 \frac{1}{\varepsilon})$ time, and its size is $O(\frac{(g+1)n^2}{\varepsilon^{3/2}q} \log^4 \frac{1}{\varepsilon})$.

6 Improved APQ Data Structure for Surfaces of Genus Zero

In this section, we consider a polyhedral surface P of genus zero⁴ (i.e., P is homeomorphic to a planar domain). We refer to this as P being planar. We show that when P is planar, it is possible to construct a data structure which is improved in terms of its size, for answering distance queries between arbitrary pairs of points on P . We refer to this data structure as *planar* APQ data structure and denote it by $PAPQ(P, \varepsilon; q)$. Roughly speaking, the size of the planar APQ data structure is reduced by a factor of $q\varepsilon^{3/2}$, whereas the preprocessing time is increased by a factor of $\varepsilon^{-1/2}$ compared to the general APQ data structure described in the previous section. More precisely, we are going to show that the following theorem holds:

Theorem 7 *Let P be a weighted polyhedral surface of genus zero consisting of n triangular faces. Let $\varepsilon \in (0, 1)$ and $q \in (\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon}, \frac{n^{1/4}}{\varepsilon})$. There exists a data structure $APQ(P, \varepsilon; q)$ such that approximate distance queries in P can be answered in $O(q)$ time. The structure $PAPQ(P, \varepsilon; q)$ is constructed in $O(\frac{n^2}{\varepsilon^2 q} \log q \log \frac{n}{\varepsilon} \log^4 \frac{1}{\varepsilon})$ time, and its size is $O(\frac{n^2}{\varepsilon^3 q^2} \log^2 q \log^6 \frac{1}{\varepsilon})$.*

The key in the construction and analysis of the planar APQ data structure, as stated in Theorem 7, is the observation that shortest paths in the approximation graph G_ε inherit some planar properties from the surface P . These properties allow us to answer queries efficiently while using less stored information.

First, we describe the $PAPQ$ data structure and present a preprocessing algorithm, called *PAPQ_Preprocessing*, for its construction. Next, we analyze the preprocessing algorithm and establish upper bounds on the size of the data structure and time required for its construction in terms of the input parameters. Then, we describe, analyze, and prove the correctness of a query algorithm. The algorithms are obtained following the approach used in the general case; hence, in our presentation, we emphasize the differences.

6.1 Preprocessing

As in the general case, the preprocessing algorithm takes as input an approximation parameter $\varepsilon \in (0, 1)$ and a query parameter q not exceeding $\bar{q}' = \frac{n^{1/4}}{\varepsilon}$, and produces a data structure that is later used for answering shortest path queries in P . The algorithm is presented formally in Table 6. The first three steps of the $PAPQ_Preprocessing$ algorithm are the same as in the $APQ_Preprocessing$ algorithm, except for a slightly different separation parameter t' in Step 3. At the end of Step 3,

⁴The genus of P can be easily computed using Euler's formula.

we obtain a partition of the faces of P into sets called regions and a special single set S called separator. Each region corresponds to a connected subgraph of P^* , and no two faces from different regions are neighbours. The number of nodes of G_ε incident to the faces in any region R is bounded by $t'w(P^*)$, i.e. $w(R) \leq t'w(P^*)$. The boundary of a region R is defined as the set of faces in S that neighbour faces in R and is denoted by ∂R . In Step 3, the preprocessing algorithm computes and stores the boundary ∂R of each region, following the order in which the triangles forming that boundary appear around R . This order is well defined in the planar case, since the edges shared by any of our regions and its boundary form a simple cycle.

According to Theorem 2, the cost of the boundaries of the regions is bounded by $2\sqrt{t'\sigma(P^*)}$, and the cost $c(S)$ of the separator is $O(\sqrt{\sigma(P^*)/t'})$, where $\sigma(P^*)$ is the sum of the squares of the costs of the faces in P .

Our PAPQ_Preprocessing algorithm consists of three more steps. In these, we compute a collection of shortest path distances and a collection of *partial SSQ* data structures that will form the PAPQ($P, \varepsilon; q$) data structure.

A partial SSQ data structure $\text{SSQ}(p; F, \varepsilon)$, from p to a set of faces $F \subset P$, is defined as the set of Local Voronoi Diagrams $\text{LVD}(p, f', f)$, from p to all faces $f \in F$, where $f' \in N(f)$. Note that the partial SSQ data structure does not store the shortest paths from p to the nodes of G_ε incident to faces neighbouring faces in F . It stores diagrams $\text{LVD}(p, f', f)$ for $f \in F$ only. From Lemma 4 it follows, straightforwardly, that the partial SSQ data structures, $\text{SSQ}(p; F, \varepsilon)$, have size $O(w(F))$ and can be constructed in $O(w(F) \log w(F))$ time, provided that the distances from p to the nodes in the faces neighbouring F are computed in advance.

We denote by $\mathcal{N}(S)$ the set of faces that neighbour faces in S , i.e., $\mathcal{N}(S) = \bigcup_{f \in S} \mathcal{N}(f)$. In Step 4, the algorithm computes distances and partial SSQ data structures related to nodes in G_ε that are incident to faces in $\mathcal{N}(S)$. That is, for each node $p \in \mathcal{N}(S)$, the following three sub-steps are implemented. First (in Step 4.1), the SSSP tree in G_ε , rooted at p , is computed. The distances $\text{dist}_{G_\varepsilon}(p, q)$ from p to all nodes $q \in \mathcal{N}(S)$ are stored in the PAPQ($P, \varepsilon; q$) data structure. Then, in Step 4.2, the algorithm computes and stores partial SSQ data structures $\text{SSQ}(p; S, \varepsilon)$ from nodes $p \in \mathcal{N}(S)$ to faces in S . Step 4.3 is implemented for nodes that are incident to faces in S only. For any such node p , up to three more partial SSQ data structures with source p are computed and stored. That is, for each of the regions $R(p)$ neighbouring the face containing p (recall that p is incident to a face in S), the partial SSQ data structure $\text{SSQ}(p; R(p), \varepsilon)$ is computed and stored.

Note that, although SSSP trees rooted at nodes in $\mathcal{N}(S)$ were computed, they are not stored in the planar APQ data structure. Instead, carefully selected collection (related to the separator S) of partial SSQ data structures have been computed and stored. As a result, the size of the obtained APQ data structure is considerably reduced compared to the one in the general case.

To describe Step 5, we introduce the notion of *restricted distances* with respect to a region in G_ε . A node p of G_ε is said to be *inside* a region R if p is incident to faces in R only. Nodes that are not inside R are said to be *outside* of R .

Definition 2 For a region R and a pair of nodes p and q outside of R , we define the restricted distance between p and q , with respect to R , to be the least cost of a path

Table 6 Planar All Pairs Queries Preprocessing algorithm**PAPQ_Preprocessing**

Input: A weighted polyhedral surface P of genus zero and consisting of n triangular faces; an approximation parameter $\varepsilon \in (0, 1)$; query time parameter $q \leq \bar{q}' = \frac{n^{1/4}}{\varepsilon}$.

Output: Data structure $\text{PAPQ}(P, \varepsilon; q)$.

Implement Steps 1 and 2 of APQ_Preprocessing algorithm (Table 4).

Step 3. Set $t' = \frac{q^2 \varepsilon}{4\sigma(P^*) \log^2 q \log^2 \frac{1}{\varepsilon}}$. Using Theorem 2, compute a t' -separator S . Construct the region partition of P induced by S . Compute and store the boundary of each region in sorted order.

Step 4. For each node $p \in G_\varepsilon$ that is incident to a face in $\mathcal{N}(S)$, implement Steps 4.1–4.3 below:

Step 4.1. Compute SSSP tree in G_ε rooted at p .

Step 4.2. Compute and store partial SSQ data structures $\text{SSQ}(p; S, \varepsilon)$.

Step 4.3. If $p \in S$, then for any region $R(p)$ neighbouring the face incident to p , compute and store $\text{SSQ}(p; R(p), \varepsilon)$.

Step 5. For each node p incident to a face $f \in S$ and for each region R neighbouring f , compute and store restricted distances $c(p \overset{G_\varepsilon \setminus R}{\rightsquigarrow} q)$ from p to all nodes q incident to faces in S .

Step 6. For each region R and for each node p incident to a face in R , compute and store $\text{SSQ}(p; R, \varepsilon)$ data structure.

in G_ε between p and q that avoids nodes inside R . We denote the restricted distance between p and q with respect to R by $c(p \overset{G_\varepsilon \setminus R}{\rightsquigarrow} q)$.

Clearly, the restricted distances from a node p outside of R to all other nodes q outside of R with respect to R can be computed by running the SSSP algorithm from p in the subgraph of G_ε , obtained by removal of all nodes inside R . The upper bound on the running time of the SSSP algorithm as stated in Theorem 3 applies in this case too. In Step 5, the PAPQ_Preprocessing algorithm considers all pairs $(p, R(p))$, where p is a node incident to a face $f(p) \in S$, and $R(p)$ is a region neighbouring $f(p)$ (or equivalently $f(p) \in \partial R(p)$), and computes the restricted distances with respect to $R(p)$ from p to all nodes q outside of $R(p)$. The restricted distances from p to nodes q that are incident to faces in S are stored in the $\text{PAPQ}(P, \varepsilon; q)$ data structure.

Step 6 of the PAPQ_Preprocessing algorithm is the same as Step 5 of the preprocessing algorithm in the general case (see Table 4). Recall that in this step, the algorithm computes and stores the SSQ data structures $\text{SSQ}(p; R, \varepsilon)$ for all regions R computed in Step 3 and nodes p incident to faces in R .

The next lemma establishes upper bounds on the running time of the PAPQ_Preprocessing algorithm and on the size of the $\text{PAPQ}(P, \varepsilon; q)$ data structure.

Lemma 7 For $q \leq \frac{n^{1/4}}{\varepsilon}$, the PAPQ_Preprocessing algorithm takes $O(\frac{n^2}{\varepsilon^2 q} \log q \log \frac{n}{\varepsilon} \times \log^4 \frac{1}{\varepsilon})$ time. The size of the $\text{PAPQ}(P, \varepsilon; q)$ data structure is $O(\frac{n^2}{\varepsilon^3 q^2} \log^2 q \log^6 \frac{1}{\varepsilon})$.

Proof First, we estimate the running time of the preprocessing algorithm. As we have shown in the proof of Lemma 5, the total time for the execution of the first three steps is bounded by $O(\frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon})$. The data stored in the PAPQ data structure during the first three steps is $O(\frac{n}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$.

According to Theorem 2, the number of the SSSP trees computed in Step 4.1 is $O(\sqrt{\sigma(P^*)/t'})$, that is, $O(\frac{\sigma(P^*) \log q \log \frac{1}{\varepsilon}}{(q\sqrt{\varepsilon})})$. By Theorem 3, each of these computations takes $O(w(P^*) \log w(P^*))$, and hence, the time for the execution of Step 4.1 is $O(\frac{\sigma(P^*)w(P^*) \log w(P^*) \log q \log \frac{1}{\varepsilon}}{(q\sqrt{\varepsilon})})$.

As stated above, the time for computation of a partial SSQ data structure $\text{SSQ}(p; F, \varepsilon)$ is proportional to $w(F) \log w(F)$, where, by our definitions, $w(F)$ is the number of nodes incident to faces in F . The size of this data structure is $O(w(F))$. Thereafter, the running time of Step 4.2 is $O(w(\mathcal{N}(S))w(S) \log w(S))$. By our definitions, this is $O(c(S)w(S) \log w(S))$, which, using the bound on $c(S)$, is $O(\frac{\sigma(P^*)w(S) \log w(S) \log q \log \frac{1}{\varepsilon}}{(q\sqrt{\varepsilon})})$. Clearly, this bound is asymptotically less than the bound obtained for Step 4.1. The size of the SSQ data structures stored during Step 4.2 is $O(w(\mathcal{N}(S))w(S))$, which is bounded by $c^2(S)$. The latter by Theorem 2 is $O(\sigma(P^*)/t')$, which after substitution for t' is $O(\frac{\sigma^2(P^*) \log^2 q \log^2 \frac{1}{\varepsilon}}{\varepsilon q^2})$.

Step 4.3 runs in $O(w(S)t'w(P^*) \log(t'w(P^*)))$ time, which, as shown below, is dominated by the running time of Step 6. Similarly, the size of the SSQ data structures stored during this step is asymptotically less than the size of the SSQ data structures stored by Step 6.

Restricted distances in Step 5 are computed using SSSP algorithm, and hence, the time for its execution satisfies the same asymptotic upper bound as Step 4.1. The number of the restricted distances stored in Step 5 is bounded by $c^2(S)$, which coincides with the bound obtained for the data stored during Step 4.2.

Step 6 is the same as Step 5 in APQ_Preprocessing algorithm for the general case, and its running time was estimated, in the proof of Lemma 5, to be $O(t'w^2(P^*) \log w(P^*))$. The size of the SSQ data structures stored during this step is $O(t'w^2(P^*))$. By substitution of t' and using (24), we obtain that both the running time and the size of the data stored in Step 6 are $O(\varepsilon q^2 n)$. For $q \leq \tilde{q} = \frac{n^{1/4}}{\varepsilon}$, this is asymptotically less than the running time of Step 4.1 and also asymptotically less than the size of the data structures stored in Step 4.2.

Overall, the running time of the PAPQ_Algorithm is $O(\frac{\sigma(P^*)w(P^*) \log w(P^*) \log q \log \frac{1}{\varepsilon}}{q\sqrt{\varepsilon}})$. The size of the $\text{PAPQ}(P, \varepsilon; q)$ data structure is $O(\frac{\sigma^2(P^*) \log^2 q \log^2 \frac{1}{\varepsilon}}{\varepsilon q^2})$. The estimates stated in the lemma are obtained from these using (23). $\square$

6.2 Query

Next, we describe our query algorithm, called PAPQ_Query. The algorithm resembles the structure of the APQ_Query algorithm as presented in Table 5 but differs in the way some of the steps are implemented. Thereby, in our presentation below,

we follow and refer to Table 5. The PAPQ_Query algorithm takes as input query points a and b together with the faces $f(a)$ and $f(b)$ containing a and b , respectively. The output of the algorithm is the approximate distance $\text{dist}_{G_\varepsilon}(a, b)$ between a and b in P (see Definition 1).

As in the general case, the distance $\text{dist}_{G_\varepsilon}(a, b)$ is found as the minimum of three values M_0 , M_1 , and M_2 corresponding to the cost of the shortest path among particular types of discrete paths between a and b . Initially M_0 , M_1 , and M_2 are set to infinity. The value M_0 corresponds to the cost of the shortest local path between a and b , i.e., if a and b lie in the neighbouring faces $f(a)$ and $f(b)$, respectively, then the shortest path between a and b lying inside the union $f(a) \cup f(b)$ is computed, and its cost is assigned to M_0 .

The value M_1 corresponds to the cost of the shortest approximate discrete path of the form (7), i.e.,

$$M_1 = \min_{p \in \mathcal{N}(a), q \in \mathcal{N}(b)} (\|a \overset{\mathcal{N}(a)}{\rightsquigarrow} p\| + c(p \overset{G_\varepsilon}{\rightsquigarrow} q) + \|q \overset{\mathcal{N}(b)}{\rightsquigarrow} b\|).$$

The query algorithm uses the PAPQ data structure and properly selected separating sets, depending on the positions of a and b with respect to the separator S , to compute M_1 efficiently. Recall that a set A of nodes in G_ε is called a separating set for a and b if any approximate discrete path between a and b contains a node in A .

In the case where both faces $f(a)$ and $f(b)$ are in S , the algorithm sets the separating set A to be the set of nodes incident to $\mathcal{N}(b)$ and computes the minimum M_1 using the formula $M_1 = \min_{q \in A} (\text{dist}_{G_\varepsilon}(a, q) + \|q \overset{\mathcal{N}(b)}{\rightsquigarrow} b\|)$. Observe that the SSQ data structures $\text{SSQ}(q; S, \varepsilon)$ from nodes $q \in A$ to the faces in S are present in the PAPQ data structure. Therefore, the distances $d_{G_\varepsilon}(a, q)$ are computed in $O(\log \frac{1}{\varepsilon})$ time. The shortest local path, and correspondingly the distance $\|q \overset{\mathcal{N}(b)}{\rightsquigarrow} b\|$, is computed in constant time. We have $|A| = O(\frac{1}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$, and thus M_1 is computed in $O(\frac{1}{\sqrt{\varepsilon}} \log^2 \frac{1}{\varepsilon})$ time.

The case where just one of the faces $f(a)$ and $f(b)$ is in S is treated similarly. Without loss of generality, assume that $f(a)$ is in S and $f(b)$ is not. Then, $f(b)$ must be in one of the regions defined by S . We denote this region by $R(b)$. In this case, the algorithm uses separating set A consisting of all nodes in G_ε that are incident to faces in the boundary $\partial R(b)$ of $R(b)$. By Theorem 2, the size of A does not exceed $\frac{q\sqrt{\varepsilon}}{\log \sigma(P^*) \log \frac{1}{\varepsilon}}$. Then, the algorithm computes M_1 using the formula $M_1 = \min_{q \in A} (\text{dist}_{G_\varepsilon}(a, q) + \text{dist}_{G_\varepsilon}(q, b))$. For any node q in A , both SSQ data structures $\text{SSQ}(q; S, \varepsilon)$ and $\text{SSQ}(q; R(b), \varepsilon)$ are stored in the PAPQ data structure, and therefore M_1 can be computed in $O(|A| \log \frac{1}{\varepsilon})$ time, which is at most $O(q)$.

The case where the faces $f(a)$ and $f(b)$ belong to different regions, say $R(a)$ and $R(b)$, respectively, is central to our presentation. Planarity of P is essentially used by our query algorithm in this case. To find the minimum M_1 in this case, we use a pair of separating sets $A(a)$ and $A(b)$ consisting of the nodes in G_ε that are incident to the faces in $\partial R(a)$ and $\partial R(b)$, respectively. Then, the algorithm finds the minimum

M_1 by computing

$$\min_{p \in A(a), q \in A(b)} \left(\text{dist}_{G_\varepsilon}(a, p) + c(p \overset{G_\varepsilon \setminus R(a)}{\rightsquigarrow} q) + \text{dist}_{G_\varepsilon}(q, b) \right), \quad (25)$$

where $c(p \overset{G_\varepsilon \setminus R(a)}{\rightsquigarrow} q)$ is the restricted distance between p and q with respect to the region $R(a)$. By our definitions, it is easily seen that this minimum equals to M_1 . Next, we describe how our query algorithm computes (25).

First, we observe that for any pair of nodes $p \in A(a)$ and $q \in A(b)$, the SSQ data structures $\text{SSQ}(p; R(a), \varepsilon)$ and $\text{SSQ}(q; R(b), \varepsilon)$, as well as the distance $c(p \overset{G_\varepsilon}{\rightsquigarrow} q)$ are present in the $\text{PAPQ}(P, \varepsilon; q)$ data structure. Thus, the three terms in (25), for any fixed pair of nodes p and q , can be computed in $O(\log \frac{1}{\varepsilon})$ time. The number of such pairs, however, can be large, and searching over all pairs to compute (25) is unacceptable. Therefore, we use a more elaborate approach and devise a recursive procedure called $\text{SEARCH_MIN}(a, b)$, to compute M_1 . The procedure uses the circular structure of the boundaries $\partial R(a)$ and $\partial R(b)$ and properties of shortest approximate discrete paths related to the planarity of P . First, we introduce some notation.

For a given face $f \in \partial R(a)$, let $M_1(f)$ be the minimum defined by

$$M_1(f) = \min_{p \in A(f), q \in A(b)} \left(\text{dist}_{G_\varepsilon}(a, p) + c(p \overset{G_\varepsilon \setminus R(a)}{\rightsquigarrow} q) + \text{dist}_{G_\varepsilon}(q, b) \right), \quad (26)$$

where $A(f)$ is the subset of $A(a)$ consisting of the nodes of G_ε incident to f . Let (p^*, q^*) be the pair with smallest restricted distance $c(p^* \overset{G_\varepsilon \setminus R(a)}{\rightsquigarrow} q^*)$ satisfying the above equation. We call such a pair an *optimal pair* of nodes for f . The face f^* containing q^* is called *optimal face* for f . The corresponding restricted path, with respect to $R(a)$, $\pi^*(f) = \pi(p^*, q^*)$ is called *optimal path* for f . Observe that by this definition, an optimal path $\pi(p^*, q^*)$ does not contain nodes which are inside $R(a)$ or $R(b)$. (It does not contain nodes inside $R(a)$, since it is a restricted shortest path with respect to $R(a)$. It does not contain nodes inside $R(b)$, since q^* has been chosen to minimize the restricted distance $c(p \overset{G_\varepsilon \setminus R(a)}{\rightsquigarrow} q)$ among pairs (p, q) for which the minimum (26) is achieved.)

The next lemma establishes an important property of optimal paths. Let f_1 and f_2 be two different faces in $\partial R(a)$, and let $\pi^*(f_1)$ and $\pi^*(f_2)$ be optimal paths for f_1 and f_2 , respectively. Let f_1^* and f_2^* be the optimal faces for f_1 and f_2 determined by $\pi^*(f_1)$ and $\pi^*(f_2)$, respectively. Furthermore, let $\tilde{\pi}^*(f_1)$ and $\tilde{\pi}^*(f_2)$ be the natural embeddings of the paths $\pi^*(f_1)$ and $\pi^*(f_2)$ in P . Roughly speaking, the natural embedding of a discrete path is obtained by embedding its edges into their corresponding local shortest paths (see Sect. 3.2 for more details).

Lemma 8 *If the natural embeddings $\tilde{\pi}^*(f_1)$ and $\tilde{\pi}^*(f_2)$ intersect, then the face f_1^* is optimal for f_2 , and f_2^* is optimal for f_1 .*

Proof Presented in [Appendix](#). □

Recall that a region R is a connected component of the dual graph P^* , i.e., it is a collection of faces forming a connected component in the dual. Hence, the edges that

Fig. 4 Illustration of an optimal chain

are incident to the faces on the boundary of R form a cycle. Furthermore, in Step 3 of the preprocessing algorithm, the faces in each boundary have been stored in the PAPQ data structure sorted according to their appearance around R . So, any pair of different faces f_1 and f_2 in the boundary ∂R defines a *clockwise chain* $C(f_1, f_2; \partial R)$ (or simply $C(f_1, f_2)$ if no ambiguity arises) consisting of all faces, including f_1 and f_2 , between f_1 and f_2 ordered in clockwise direction starting at f_1 . We also use the notation $C(f, f)$ for a chain consisting of a single face f and $\bar{C}(f, f)$ for the chain consisting of all faces in ∂R ordered in clockwise direction starting at f . The size of a chain $C(f_1, f_2)$ is the number of faces in it and is denoted by $|C(f_1, f_2)|$. We define *median* of a chain $C(f_1, f_2)$ as a face $f \in C(f_1, f_2)$ such that the difference between the sizes of the chains $C(f_1, f)$ and $C(f, f_2)$ is at most one.

Let f_1^* and f_2^* be the optimal faces for f_1 and f_2 , respectively, where f_1, f_2 are on the boundary of $R(a)$, and f_1^*, f_2^* are on the boundary of $R(b)$. Then, for a chain $C = C(f_1, f_2)$ on the boundary of $R(a)$, the chain $C^* = C(f_2^*, f_1^*)$, on the boundary of $R(b)$, is said to be the *optimal chain* (see Fig. 4). Furthermore, optimal chains and faces satisfy the following useful relation.

Lemma 9 *Let C be a chain on $\partial R(a)$, and let C^* be the optimal chain on $\partial R(b)$ for C . Then, any face $f \in C$ has an optimal face in C^* .*

Proof Follows from the planarity of P and Lemma 8. See Fig. 4. $\square$

Next, we describe our procedure $\text{SEARCH_MIN}(a, b)$ that computes the minimum (25). The procedure first computes approximate distances $\text{dist}_{G_\varepsilon}(a, p)$ and $\text{dist}_{G_\varepsilon}(q, b)$ for all nodes p and q incident to faces in $\partial R(a)$ and $\partial R(b)$, respectively. Then, the procedure picks an arbitrary face f_0 on $\partial R(a)$ and computes the minimum $M_1(f_0)$ and an optimal face $f_0^* \in \partial R(b)$ for f_0 . At the end, $\text{SEARCH_MIN}(a, b)$ calls a recursive procedure $\text{SEARCH_MIN}(\bar{C}(f_0, f_0), \bar{C}(f_0^*, f_0^*))$ and outputs M_1 as the smaller of $M_1(f_0)$ and the value returned by that procedure. Below, we describe the recursive procedure $\text{SEARCH_MIN}(C, C^*)$.

The procedure $\text{SEARCH_MIN}(C, C^*)$ takes as input a chain $C = C(f_1, f_2)$ in the boundary of $R(a)$ and an optimal chain $C^* = C(f_2^*, f_1^*)$ for C and computes the minimum $M_1(C, C^*)$ defined by (25), but over the subsets $A(C) \subset A(a)$ and $A(C^*) \subset A(b)$, where $A(C)$ consists of all nodes in G_ε incident to faces in C . The set $A(C^*)$ is defined analogously with respect to b . The procedure $\text{SEARCH_MIN}(C, C^*)$ checks whether any of the chains C and C^* consists of a single face and, if so, computes $M_1(C, C^*)$ by checking all pairs of nodes $p \in A(C)$ and $q \in A(C^*)$. If both chains contain more than one face, then the procedure finds the median face f of C

and computes the minimum

$$M_1(f) = \min_{p \in A(f), q \in A(C^*)} (\text{dist}_{G_\varepsilon}(a, p) + c(p \overset{G_\varepsilon \setminus R(a)}{\rightsquigarrow} q) + \text{dist}_{G_\varepsilon}(q, b)) \quad (27)$$

and an optimal face f^* for f . Then, the procedure $\text{SEARCH_MIN}(C, C^*)$ calls recursively $\text{SEARCH_MIN}(C(f_1, f), C(f^*, f_1^*))$ and $\text{SEARCH_MIN}(C(f, f_2), C(f_2^*, f^*))$ and outputs

$$M_1(C, C^*) = \min(M_1(C(f_1, f), C(f^*, f_1^*)), M_1(C(f, f_2), C(f_2^*, f^*))). \quad (28)$$

The latter is valid because of Lemma 9.

Lemma 10 *The procedure $\text{SEARCH_MIN}(a, b)$ correctly computes the minimum (25) in $O(\max(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon}, q))$ time.*

Proof The fact that $\text{SEARCH_MIN}(C, C^*)$ computes $M_1(C, C^*)$ follows by induction on the size of C and using the validity of (28). Then, by definition, we have $\bar{C}(f_0, f_0) = \partial R(a)$ and $\bar{C}(f_0^*, f_0^*) = \partial R(b)$, and the correctness of $\text{SEARCH_MIN}(a, b)$ follows.

We first estimate the running time of $\text{SEARCH_MIN}(C, C^*)$. From its description, it is easy to see that the running time of $\text{SEARCH_MIN}(C, C^*)$ is bounded by $T(C^*) \log |C|$, where $T(C^*)$ is an upper bound on the time for computing the minimum $M_1(f)$ by (27) for any face f in C . By Lemma 3, the size of the set $A(f)$ for any face f is $O(\frac{1}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$, and hence $T(C^*) = O(\frac{|A(C^*)|}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$. We have obtained that the running time of $\text{SEARCH_MIN}(C, C^*)$ is $O(\frac{|A(C^*)|}{\sqrt{\varepsilon}} \log |C| \log \frac{1}{\varepsilon})$. On the other hand, if each of the chains consists of a single face, the running time of $\text{SEARCH_MIN}(C, C^*)$ is $\Omega(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})$.

The running time of $\text{SEARCH_MIN}(a, b)$ is the sum of the times for the initialization and the time taken by the recursive procedure $\text{SEARCH_MIN}(\bar{C}(f_0, f_0), \bar{C}(f_0^*, f_0^*))$. The latter is $O(\frac{w(\partial R(b))}{\sqrt{\varepsilon}} \log |\partial R(a)| \log \frac{1}{\varepsilon})$, where $w(\partial R(b))$ is the number of nodes of G_ε incident to faces in $\partial R(b)$. By Theorem 2 and the value of the separation parameter t' , we have $w(\partial R(b)) \leq \frac{q\sqrt{\varepsilon}}{\log q \log \frac{1}{\varepsilon}}$. From the same estimate on $w(\partial R(a))$ we have that $|\partial R(a)| \leq q$. Therefore, procedure $\text{SEARCH_MIN}(\bar{C}(f_0, f_0), \bar{C}(f_0^*, f_0^*))$ runs in $O(\max(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon}, q))$ time.

Partial SSQ data structures $\text{SSQ}(p; R(a), \varepsilon)$ and $\text{SSQ}(q; R(b), \varepsilon)$ are present in the PAPQ data structure for all nodes p incident to faces in $\partial R(a)$ and for nodes q incident to faces in $\partial R(b)$. Therefore, each of the distances $\text{dist}_{G_\varepsilon}(a, p)$ and $\text{dist}_{G_\varepsilon}(q, b)$ is computed in $O(\log \frac{1}{\varepsilon})$ time. The number of these distances is bounded by the sum $w(\partial R(a)) + w(\partial R(b))$, which, as discussed above, is less than $2 \frac{q\sqrt{\varepsilon}}{\log q \log \frac{1}{\varepsilon}}$. Hence, this computation takes less than $O(q)$ time.

Computing $M_1(f_0)$ and an optimal face $f_0^* \in \partial R(b)$ takes $O(\max(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon}, \frac{q}{\log q}))$ time. The lemma follows. $\square$

Finally, we consider the case where faces $f(a)$ and $f(b)$ belong to a single region $R = R(a) = R(b)$. To find the distance $\text{dist}_{G_\varepsilon}(a, b)$ in this case, we partition approximate discrete paths between a and b into two groups. The first group consists of approximate discrete paths that stay within R , and the second group consists of those that contain nodes outside R . Then, the distance $\text{dist}_{G_\varepsilon}(a, b)$ is equal to the smaller of the two distances found with respect to the paths in each group.

To find the distance with respect to the paths inside R , we define A to be the set of nodes in G_ε incident to the faces in $\mathcal{N}(f(a))$ and compute $\min_{p \in A} (\|a \overset{\mathcal{N}(a)}{\rightsquigarrow} p\| + \text{dist}_R(p, b))$. For any $p \in A$, the distance $\text{dist}_R(p, b)$ is computed using the SSQ data structure $\text{SSQ}(p; R, \varepsilon)$ in $O(\log \frac{1}{\varepsilon})$ time. The shortest local path is computed in constant time, and therefore the minimum and the distance inside R is computed in $O(\frac{1}{\sqrt{\varepsilon}} \log^2 \frac{1}{\varepsilon})$. The distance with respect to paths that leave R is computed in $O(q)$ time, since $w(\partial R(a)) \leq \frac{q\sqrt{\varepsilon}}{\log q \log \frac{1}{\varepsilon}}$.

This concludes the description of the PAPQ_Query algorithm. We summarize the above discussion in the following lemma.

Lemma 11 *The algorithm PAPQ_Query correctly computes the approximate distance $\text{dist}_{G_\varepsilon}(a, b)$ in $O(\max(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon}, q))$ time.*

The validity of Theorem 7 stated in the beginning of the section follows from Lemmas 7 and 11.

7 Extensions and Conclusions

In this paper, we present novel solutions to fundamental shortest path query problems. The algorithms improve and generalize previous solutions in terms of (1) setting: (a) Euclidean to weighted and (b) arbitrary genus g , (2) preprocessing time, and/or (3) size of query data structure. We also develop a new graph partitioning algorithm for embedded graphs of genus g with weights and costs on vertices, which extends and/or generalizes previously known separator algorithms. Our techniques also enable us to obtain improved results with space-query time tradeoffs for the planar case, i.e., when the genus is 0.

A natural question arises whether the range $(\frac{1}{\sqrt{\varepsilon}} \log^2 \frac{1}{\varepsilon}, \bar{q})$ of query time parameter q , in Theorem 6, can be widened while keeping the efficiency of the algorithm. This question can be answered affirmatively by constructing a hierarchical APQ data structure in which the second-level APQ data structures are built and stored for each region of the partition. For each region R of the partition, construct an $\text{APQ}(R, \varepsilon; \bar{q}_R)$ restricted to R , where $\bar{q}_R$ is the corresponding upper bound for R . Then, in the APQ-Query algorithm, we use $\text{APQ}(R, \varepsilon; \bar{q}_R)$ to answer approximate shortest path queries in the case where both query points lie in R in $O(\bar{q}_R)$ time. The latter is $O(q)$ since $\bar{q} \leq q$. Our analysis shows that if $q \leq \bar{q}_1 = \frac{(g+1)^{5/9} n^{2/3}}{\varepsilon^{2/3}} \log^{7/3} \frac{1}{\varepsilon}$, the result of Theorem 6 extends to the case $q \in (\bar{q}, \bar{q}_1)$. By the same approach, the interval for q in the planar case can be extended from above.

Next, we briefly describe how our technique can be used to build an APQ data structure with query time parameter $q = \log \frac{1}{\varepsilon}$. We build an ε -mesh consisting of $O(\frac{1}{\varepsilon^2})$ additional points in each triangle f of P . The mesh is constructed so that for each point a in f , there is a mesh point p which is closer to a than $\varepsilon d(p)$, where $d(p)$ is the minimum distance from p to any node of G_ε outside f . Then, for each mesh point, we compute in $O(\frac{n^2}{\varepsilon^{5/2}} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time the SSQ data structure. For a pair of query points a and b , we first find the mesh point $p(a)$ closest to a and then use the structure $\text{SSQ}(p(a); P, \varepsilon)$ to find $\text{dist}_{G_\varepsilon}(p(a), b)$. Each of these two steps is carried out in $O(\log \frac{1}{\varepsilon})$ time. It can be shown that $\|ap(a)\| + \text{dist}_{G_\varepsilon}(p(a), b)$ is an ε -approximation of the distance $\text{dist}_P(a, b)$.

Appendix

Lemma 8 *If the natural embedding $\tilde{\pi}^*(f_1)$ and $\tilde{\pi}^*(f_2)$ intersect, then the face f_1^* is optimal for f_2 , and f_2^* is optimal for f_1 .*

Proof Let z be the first point of intersection between $\tilde{\pi}^*(f_1)$ and $\tilde{\pi}^*(f_2)$ (following the paths starting at f_1 and f_2 , respectively). Let (p_1^*, q_1^*) and (p_2^*, q_2^*) be the optimal pair for f_1 and f_2 , respectively, for which the minimum (26) is achieved. Let $\pi_1(a, b) = (a \overset{G_\varepsilon}{\rightsquigarrow} p_1^* \overset{G_\varepsilon}{\rightsquigarrow} q_1^* \overset{G_\varepsilon}{\rightsquigarrow} b)$ denote the shortest path from a to b passing through the optimal pair (p_1^*, q_1^*) . Similarly, let $\pi_2(a, b)$ denote the shortest path from a to b passing through (p_2^*, q_2^*) . Let $\pi_1(i, j)$ (or $\pi_2(i, j)$) denote the subpath of $\pi_1(a, b)$ (respectively, $\pi_2(a, b)$) between (nodes) i and j .

Case 1: Consider the case where z corresponds to a node in G_ε . In that case, the subpaths $\pi_1(z, b)$ and $\pi_2(z, b)$ have equal cost. Construct a new path $\pi_{12}(a, b) = \{\pi_1(a, z), \pi_2(z, b)\}$. We have $\|\pi_1(a, b)\| = \|\pi_{12}(a, b)\|$, thus $\pi_{12}(a, b)$ is a shortest path intersecting f_1 and f_2^* via an optimal pair (p_1^*, q_2^*) . Hence, the face f_2^* is optimal for f_1 . (So does f_1^* for f_2 .)

Case 2: Consider the case where z does not correspond to a node in G_ε . It belongs to the intersection of the embedding of two edges $e_1 = (w_1, v_1)$ and $e_2 = (w_2, v_2)$ from $\pi^*(f_1)$ and $\pi^*(f_2)$, respectively.

We say that point z is *isolated* if there exists a disk of nonzero radius centered at z that does not contain any other point that is common to $\pi^*(f_1)$ and $\pi^*(f_2)$. We consider the case where z is an isolated point in Cases 2a and 2b, and otherwise in Case 2c.

Case 2a: Let z be an isolated point in the interior of a face denoted by $f(z)$. Then z is the intersection of two segments corresponding to embeddings of edges e_1 and e_2 . Let the embedding $\tilde{e}_1$ of e_1 be represented by two segments $\tilde{e}_1 = \{\overline{w_1 x_1}, \overline{x_1 v_1}\}$, where w_1 and v_1 correspond to nodes in G_ε , and x_1 is a bending point. Likewise, let the embedding $\tilde{e}_2$ of e_2 be $\tilde{e}_2 = \{\overline{w_2 x_2}, \overline{x_2 v_2}\}$, where w_2 and v_2 correspond to nodes in G_ε , and x_2 is a bending point. Without loss of generality, assume that z is the intersection of $\overline{x_1 v_1}$ and $\overline{x_2 v_2}$ (see Fig. 5(a)). Note that the points (or the corresponding nodes in G_ε) w_1, v_1, w_2, v_2 are incident to the face neighbourhood $\mathcal{N}(z)$ of z . Thus, by definition, (w_1, v_2) and (w_2, v_1) are edges of G_ε . Also, the cost

Fig. 5 An illustration of the case where a point z whose representation is not a node in G_ε . **(a)** A point z is an isolated point in the interior of a face (Case 2a). **(b)** and **(c)** A point z is an isolated point on an edge of P (Case 2b). **(d)** A point z is not isolated (Case 2c)

of the edge (w_1, v_2) corresponds to the cost of shortest path connecting the end-points which is restricted to lie within the face neighbourhood $\mathcal{N}(z)$. Same argument holds for the cost of edge (w_2, v_1) . By triangle inequality and the cost of edges, we have $w_1 \xrightarrow{\mathcal{N}(z)} v_2 \leq \overline{w_1 x_1} + \overline{x_1 v_2}$ and $w_2 \xrightarrow{\mathcal{N}(z)} v_1 \leq \overline{w_2 x_2} + \overline{x_2 v_1}$. Since $\overline{x_1 v_2} + \overline{x_2 v_1} < \overline{x_1 v_1} + \overline{x_2 v_2}$, it follows that

$$\begin{aligned} \|w_1 \xrightarrow{\mathcal{N}(z)} v_2\| + \|w_2 \xrightarrow{\mathcal{N}(z)} v_1\| &< \|\overline{w_1 x_1}\| + \|\overline{w_2 x_2}\| + \|\overline{x_1 v_1}\| + \|\overline{x_2 v_2}\| \\ &= \|e_1\| + \|e_2\|. \end{aligned} \quad (29)$$

In case x_1 and x_2 are also nodes in G_ε , w_1 coincides with x_1 , w_2 coincides with x_2 , and (29) is still valid. We construct two paths denoted by $\pi_{12}(a, b)$ and $\pi_{21}(a, b)$, from a to b , passing through f_1 and f_2 , respectively, as follows: $\pi_{12}(a, b) = \{\pi_1(a, w_1), (w_1, v_2), \pi_2(v_2, b)\}$ and $\pi_{21}(a, b) = \{\pi_2(a, w_2), (w_2, v_1), \pi_1(v_1, b)\}$. Applying (29), we have $\|\pi_{12}(a, b)\| + \|\pi_{21}(a, b)\| < \|\pi_1(a, b)\| + \|\pi_2(a, b)\|$. Hence, this case contradicts the optimality of the paths that we started with.

Case 2b: Let z be an isolated point on an edge of P . Then z must be a bending point as shown in Figs. 5(b) and (c). Note that w_1, w_2, v_1, v_2 are nodes in G_ε , and the edges e_1 and e_2 are denoted by $e_1 = (w_1, v_1)$ and $e_2 = (w_2, v_2)$, respectively. In all possible cases, we have $\|w_1 \xrightarrow{\mathcal{N}(z)} v_2\| + \|w_2 \xrightarrow{\mathcal{N}(z)} v_1\| < \|w_1 \xrightarrow{\mathcal{N}(z)} v_1\| + \|w_2 \xrightarrow{\mathcal{N}(z)} v_2\| = \|e_1\| + \|e_2\|$, which follows the same argument as above. This proves that z cannot be an isolated point on an edge of P .

Case 2c: Consider now the case where z is not isolated. (The only possibility for z in this case is shown in Fig. 5(d).) The embedding of e_1 is represented by three segments $\{\overline{w_1 x_1}, \overline{x_1 y_1}, \overline{y_1 v_1}\}$, where w_1, v_1 are nodes of G_ε , and they are the end-points of e_1 . Similarly, the embedding of e_2 is represented by $\{\overline{w_2 x_2}, \overline{x_2 y_2}, \overline{y_2 v_2}\}$. Then, x_1, x_2, y_1, y_2 are bending points, and z is their first common point. Without loss of generality, let x_2 coincide with z . We have already shown that (w_1, v_2) and (w_2, v_1) are edges of G_ε . Since we are dealing with paths within a triangle, observe that $\|w_1 \xrightarrow{\mathcal{N}(z)} v_2\| + \|w_2 \xrightarrow{\mathcal{N}(z)} v_1\| = \|w_1 \xrightarrow{\mathcal{N}(z)} v_1\| + \|w_2 \xrightarrow{\mathcal{N}(z)} v_2\|$. As in Case 2a, we define two paths: $\pi_{12}(a, b) = \{\pi_1(a, w_1), (w_1, v_2), \pi_2(v_2, b)\}$ and $\pi_{21}(a, b) = \{\pi_2(a, w_2), (w_2, v_1), \pi_1(v_1, b)\}$. Thus, we have $\|\pi_{12}(a, b)\| + \|\pi_{21}(a, b)\| = \|\pi_1(a, b)\| + \|\pi_2(a, b)\|$. Hence, $\|\pi_{12}(a, b)\| = \|\pi_1(a, b)\|$ and $\|\pi_{21}(a, b)\| = \|\pi_2(a, b)\|$ must hold. Otherwise, without loss of generality, let

$\|\pi_{12}(a, b)\| = \|\pi_1(a, b)\| + \delta$ and $\|\pi_{21}(a, b)\| = \|\pi_2(a, b)\| - \delta$, where $\delta > 0$. This contradicts the optimality of the optimal paths $\pi^*(f_1)$ and $\pi^*(f_2)$. Hence, we proved the claim. $\square$

References

1. Agarwal, P.K., Aronov, B., O'Rourke, J., Schevon, C.A.: Star unfolding of a polytope with applications. *SIAM J. Comput.* **26**(6), 1689–1713 (1997)
2. Agarwal, P.K., Har-Peled, S., Sharir, M., Varadarajan, K.: Approximate shortest paths on a convex polytope in three dimensions. *J. ACM* **44**(4), 567–584 (1997)
3. Agarwal, P.K., Har-Peled, S., Karia, M.: Computing approximate shortest paths on convex polytopes. *Algorithmica* **33**(2), 227–242 (2002)
4. Aleksandrov, L., Djidjev, H.: Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Discrete Math.* **9**(1), 129–150 (1996)
5. Aleksandrov, L., Lanthier, M., Maheshwari, A., Sack, J.-R.: An ε -approximation algorithm for weighted shortest path queries on polyhedral surfaces. In: *Proc. 14th Euro CG Barcelona*, pp. 19–21 (1998)
6. Aleksandrov, L., Lanthier, M., Maheshwari, A., Sack, J.-R.: An ε -approximation algorithm for weighted shortest paths on polyhedral surfaces. In: *Proceedings of SWAT. LNCS*, vol. 1432, pp. 11–22. Springer, Berlin (1998)
7. Aleksandrov, L., Lanthier, M., Maheshwari, A., Sack, J.-R.: An improved approximation algorithm for computing geometric shortest paths. In: *Proc. Foundations of Computation Theory. LNCS*, vol. 2751, pp. 246–257. Springer, Berlin (2003)
8. Aleksandrov, L., Maheshwari, A., Sack, J.-R.: Determining approximate shortest paths on weighted polyhedral surfaces. *J. ACM* **52**(1), 25–53 (2005)
9. Aleksandrov, L., Djidjev, H., Guo, H., Maheshwari, A.: Partitioning planar graphs with costs and weights. *ACM J. Exp. Algorithmics* **11**, 1.5 (2006)
10. Aleksandrov, L., Djidjev, H., Guo, H., Maheshwari, A., Nussbaum, D., Sack, J.-R.: Approximate shortest path queries on weighted polyhedral surfaces. In: *Proc. 31st Int. Symp. Mathematical Foundations of Computer Science. LNCS*, vol. 4162, pp. 98–109. Springer, Berlin (2006)
11. Arikati, S., Chen, D., Chew, L., Das, G., Smid, M., Zaroliagis, C.: Planar spanners and approximate shortest path queries among obstacles in the plane. In: *Proc. of the Forth Annual Fourth Annual European Symposium on Algorithms ESA'96. LNCS*, vol. 1136, pp. 514–528. Springer, Berlin (1996)
12. Battle, J., Harary, F., Kodama, Y., Youngs, J.W.T.: Additivity of the genus of a graph. *Bull. Am. Math. Soc.* **68**(6), 565–568 (1962)
13. Canny, J., Reif, J.H.: New lower bound techniques for robot motion planning problems. In: *Proc. 28th Annu. IEEE Symposium on Foundations of Computer Sciences*, pp. 49–60 (1987)
14. Chazelle, B., Liu, D., Magen, A.: Sublinear geometric algorithms. *SIAM J. Comput.* **35**, 627–646 (2006)
15. Chen, D.: On the all-pairs Euclidean short path problem. In: *Proc. 6th ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pp. 292–301 (1995)
16. Chen, J., Han, Y.: Storing shortest paths for a polyhedron. In: *Proc. 1991 Int. Conf. on Comp. and Information. LNCS*, vol. 497, pp. 169–180. Springer, Berlin (1991)
17. Chen, J., Han, Y.: Shortest paths on a polyhedron. In: *Proceedings of 6th ACM Symposium on Computational Geometry*, pp. 360–369 (1990). Full version *IJCGA* **6**, 127–144 (1996)
18. Chen, D.Z., Daescu, O., Klenk, K.S.: On geometric path query problems. *Int. J. Comput. Geom. Appl.* **11**(6), 617–645 (2001)
19. Cheng, S.-W., Na, H.-S., Vigneron, A., Wang, Y.: Querying approximate shortest paths in anisotropic regions. In: *Erickson, J. (ed.) Symposium on Computational Geometry*, pp. 84–91. ACM, New York (2007)
20. Cheng, S.-W., Na, H.-S., Vigneron, A., Wang, Y.: Approximate shortest paths in anisotropic regions. *SIAM J. Comput.* **38**(3), 802–824 (2008)
21. Chiang, Y.-J., Mitchell, J.: Two-point Euclidean shortest path queries in the plane. In: *Proc. 10th ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pp. 215–224 (1999)
22. Clarkson, K.: Approximation algorithms for shortest path motion planning. In: *Proc. 19th Annu. Symp. Theory Comput. (STOC)*, pp. 56–65 (1987)

23. Djidjev, H.N.: Linear algorithms for graph separation problems. In: SWAT'88. LNCS, vol. 318, pp. 216–222. Springer, Berlin (1988)
24. Djidjev, H.: Partitioning planar graphs with vertex costs: algorithms and applications. *Algorithmica* **28**(1), 51–75 (2000)
25. Dudley, R.M.: Metric entropy of some classes of sets with differentiable boundaries. *J. Approx. Theory* **10**(3), 227–236 (1974)
26. Frederickson, G.N.: Fast algorithms for shortest paths in planar graphs. *SIAM J. Comput.* **16**, 1004–1022 (1987)
27. Gilbert, J.R., Hutchinson, J.P., Tarjan, R.E.: A separator theorem for graphs of bounded genus. *J. Algorithms* **5**, 391–407 (1984)
28. Guibas, L., Hershberger, J.: Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.* **39**, 126–152 (1989)
29. Har-Peled, S.: Approximate shortest paths and geodesic diameters on convex polytopes in three dimensions. *Discrete Comput. Geom.* **21**, 217–231 (1999)
30. Har-Peled, S.: Constructing approximate shortest path maps in three dimensions. *SIAM J. Comput.* **28**(4), 1182–1197 (1999)
31. Hershberger, J., Suri, S.: An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.* **28**, 2215–2256 (1999)
32. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. *SIAM J. Appl. Math.* **36**, 177–189 (1979)
33. Mitchell, J., Papadimitriou, C.: The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM* **38**, 18–73 (1991)
34. Mitchell, J., Mount, D., Papadimitriou, C.: The discrete geodesic problem. *SIAM J. Comput.* **16**, 647–668 (1987)
35. Reif, J.H., Sun, Z.: An efficient approximation algorithm for weighted region shortest path problem. In: Proceedings of the 4th Workshop on Algorithmic Foundations of Robotics (WAFR2000), pp. 191–203. AK Peters, Wellesley (2000)
36. Reif, J.H., Sun, Z.: On finding approximate optimal paths in weighted regions. *J. Algorithms* **58**, 1–32 (2006)
37. Sarnak, N., Tarjan, R.E.: Planar point location using persistent search trees. *Commun. ACM* **29**, 669–679 (1986)
38. Schreiber, Y., Sharir, M.: An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. *Discrete Comput. Geom.* **39**(1–3), 500–579 (2008)
39. Sharir, M., Schorr, A.: On shortest paths in polyhedral spaces. *SIAM J. Comput.* **15**, 193–215 (1986)
40. Tan, X., Hirata, T., Inagaki, Y.: Spatial point location and its applications. In: Asano, T., Ibaraki, T., Imai, H., Nishizeki, T. (eds.) SIGAL International Symposium on Algorithms. LNCS, vol. 450, pp. 241–250. Springer, Berlin (1990)