EAR-Oracle: On Efficient Indexing for Distance Queries between Arbitrary Points on Terrain Surface Technical Report

ABSTRACT

Due to the advancement of geo-positioning technology, the terrain data has become increasingly popular and has drawn a lot of research effort from both academia and industry. The distance computation on the terrain surface is a fundamental and important problem that is widely applied in geographical information systems and 3D modeling. As could be observed from the existing studies, online computation of the distance on the terrain surface is very expensive. All existing index-based methods are only efficient under the case where the distance query must be performed among a small set of predefined points-of-interest known apriori. But, in general cases, they could not scale up to sizable datasets due to their intolerable oracle building time and space consumption.

In this paper, we studied the arbitrary point-to-arbitrary point distance query on the terrain surface in which no assumption is imposed on the query points, and the distance query could be performed between any two arbitrary points. We propose an indexing structure, namely *Efficient Arbitrary Point-to-Arbitrary Point Distance Oracle* (*EAR-Oracle*), with theoretical guarantee on the accuracy, oracle building time, oracle size and query time. Our experiments demonstrate that our oracle enjoys excellent scalability and it scales up to enormous terrain surfaces but none of the existing index-based methods could be able to. Besides, it significantly outperforms all existing online computation methods by orders of magnitudes in terms of the query time.

1 INTRODUCTION

With the development of geo-positioning technology, terrain data has become popular and query processing on terrain surface becomes an emerging research topic in both academia and industry [11, 13, 20, 21, 27, 32, 39, 41, 42]. As Figure 1 shows, a terrain surface is a planar graph consisting of faces, edges and vertices. Each face on a terrain surface is a triangle with three adjacent vertices and three adjacent edges. Each edge connects two adjacent vertices and each vertex is a 3D point. As the figure shows, the terrain in this example contains 18 vertices (represented by solid dots), 39 edges and 23 faces (i.e., triangles).

The shortest distance between two points on the terrain surface is called *geodesic shortest distance*. It is defined to be the length of the shortest path between the two points on the terrain surface, namely *geodesic shortest path*. Consider the example in this figure. There are two arbitrary points on the terrain surface, namely s and t, defined by two hollow dots. The dashed line GP which consists of many line segments denotes the geodesic path between s and t. The length of the path is their geodesic distance. The dashed line EP corresponds to their shortest path in the Euclidean space and the length of this path is their Euclidean distance. It is worth mentioning that the geodesic distance is significantly different from the traditional Euclidean distance. According to the existing studies [11], the difference between the geodesic distance and the Euclidean distance could be as large as 300%.

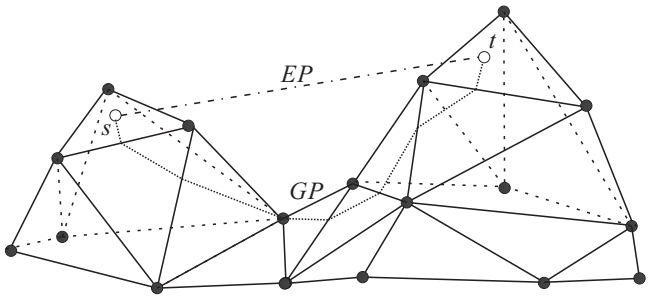


Figure 1: An Example of Terrain Surface

The distance query on terrain surfaces has many applications. (1) With the rise of Metaverse [25, 26], 3D modeling of buildings and infrastructures in the urban area and the objects such as mountains, hills and valleys in the rural areas become more and more popular. The shortest geodesic distance computation [25, 26] is a fundamental building block for location-based services such as the POI recommendation in its own virtual world. (2) In Geographical Information System (GIS), the geodesic distances are used to compute the travel cost (e.g., travel time and energy consumption) from one place to another [31]. Besides, life scientists study the travel patterns of animals by using the geodesic distances between their residential sites [14, 28]. (3) In the scientific 3D modeling [1, 18, 23, 33], each 3D object such as an organ has many reference points and the geodesic distances between the reference points are used as their key features for scientific analysis. (4) In spatial data-mining, many techniques frequently invoke geodesic distance queries. One typical example is clustering where the inner-cluster geodesic distances and inter-cluster geodesic distances are used frequently [7, 38]. In spatial co-location pattern mining, the geodesic distance queries are also largely invoked for checking co-location patterns [17, 30].

Although there are a lot of existing studies on distance queries on terrain surfaces [2, 4, 8, 15, 20, 21, 24, 27, 29, 39, 40], they are not efficient enough for distance query processing between two arbitrary points. The first category of the existing studies consists of the *on-the-fly algorithms* [8, 29, 40]. The time complexity of each exact on-the-fly algorithm is larger than $O(N^2)$, where N is the number of vertices on the terrain surface. This complexity is not tolerable for sizable datasets and could not scale up. Each of the existing approximate algorithms [20, 21] takes $O((N + N') \log(N + N'))$ time, where N' is the number of additional points added for the distance

computation. The time complexity is still too large to apply for many real-world applications. According to the experiment of [39], the running time of the algorithm is more than 300 seconds for a small dataset with only 300K vertices which is very slow and prevents its usage in many real-time applications. The second category of the existing studies contains the *index-based methods* [4, 39] which accelerate the query processing with a pre-computed indexing structure. To the best of our knowledge, there is no index-based method for computing the exact distances and thus, all algorithms in this category are approximate algorithms. The state-of-the-art algorithm in this category [39] focuses on the POI-to-POI query where the query must be performed among a small set of points-of-interests known apriori. Besides, the set of POIs is assumed to be typically significantly smaller than the number of vertices on the terrain surface. In the general cases where there is no assumption on the locations of the query points and the query could be performed between two arbitrary points, all existing index-based algorithms [4, 39] introduce a large number of auxiliary points, namely *Steiner points*, whose amount is typically orders of magnitudes larger than the number of vertices on the terrain surface. Their indexing structures are built upon these Steiner points in place of the original POIs or vertices. Since the number of Steiner points introduced is orders of magnitudes larger than that of original vertices on the terrain surface, the existing index-based algorithms inevitably have a prohibitively large building time and space consumption which renders them unscalable to sizable terrain datasets. According to our experiments, their building time is more than 3 hours and their memory usage exceeds our memory budget (i.e., 256GB) even for a small terrain with 10,243 vertices only.

Motivated by this, in this paper, we proposed an indexing structure, namely *Efficient Arbitrary Point-to-Arbitrary Point Distance Oracle* (*EAR-Oracle*), for the arbitrary point-to-arbitrary point distance query processing on the terrain surface. It integrates the *online search* and a *highway network-based indexing structure*. The *highway network* is a sparse graph which indexes the pairwise distances between the highway nodes (which are carefully selected "hub" points on the terrain surface) and bridges the distant regions. As such, the distance query between two points in a local area could be answered by a local online search and the distance between two points in the distant areas is processed by using the highway network. Our theoretical analysis and empirical study demonstrates that the highway network-based approach is *lightweight* (which incurs very small indexing overhead and query cost) and could scale up and it provides highly accurate results. Our contributions are three-folds. Firstly, we proposed a highway network-based approach for the arbitrary point-to-arbitrary point distance query on terrain surfaces. To the best of our knowledge, it is the first index-based algorithm which could answer this kind of query efficiently with accuracy guarantee. Secondly, we provided theoretical analysis on the complexities of its building time, space consumption, query time and also the error bound. Thirdly, we conducted empirical study and our experimental results show that our algorithm outperforms the existing algorithms significantly by orders of magnitudes in terms of building time, space consumption and query time.

The remainder of the paper is organized as follows. Section 2 presents the problem definition. Section 3 reviews the related studies and compares them with this paper. Section 4 demonstrates our proposed indexing structure, *Efficient Arbitrary Point-to-Arbitrary Point Oracle*. Section 5 reports the experimental results. Finally, Section 6 concludes this paper.

2 PROBLEM DEFINITION

Consider a terrain surface T. Let V denote the set consisting of all vertices of T and let E denote the set of all edges of T. The size of the terrain, denoted by N, is defined to be the cardinality of V (i.e., $N = |V|$). Consider a vertex v in V. We denote the three coordinate values of v as x_v, y_v and z_v, respectively. Let F denote the set of all faces of T. Each face f in F is associated with a weight, denoted by $w(f)$, which is a positive real number. The weights capture the different travel costs (e.g., time and energy consumption) for a distance unit on different faces (e.g., it takes more time to travel through a sandy area or wetland than a normal ground even though the travel distances are the same) and all terrain properties regarding the travel cost such as slopes, terrain types (e.g., wetland, sand and rock) and obstacles could be encoded as weights [3, 15, 24]. It is worth mentioning that in the general case where the weight of each face could be any positive real number, we call such a terrain surface a *weighted terrain surface* and in the case where the weight of each face is equal to 1, the terrain surface degenerates to an *unweighted terrain surface*.

Consider two arbitrary surface points s, t and let $\pi_g(s,t)$ denote a path between them on the terrain surface. Formally, $\pi_g(s,t)$ consists of a sequence X of line segments and each segment lies on a unique face of the terrain surface. Given a line segment $x \in X$, we denote the unique face that x lies on by f_x and we denote the length of x by $l(x)$. The length of $\pi_g(s,t)$, denoted by $l(\pi_g(s,t))$, is defined to be $\sum_{x \in X}(w(f_x) \cdot l(x))$ which is the sum over the product of the length of each line segment in X and the weight of the face it lies on. The *geodesic shortest path* between s and t, denoted by $\Pi_g(s,t)$, is defined to be the path between s and t on the terrain surface with the smallest length (i.e., $\Pi_g(s,t) = \arg\min_{\pi_g(s,t)}\{l(\pi_g(s,t))\}$). The *geodesic distance* between s and t, denoted by $d_g(s,t)$, is defined to be the length of $\Pi_g(s,t)$. Note that $d_g(\cdot,\cdot)$ is a metric and the triangle inequality could be applied.

Problem 1 (Distance Oracle). *We would like to design an indexing structure, namely distance oracle, for the distance query processing on terrain surfaces which provides approximate distance and has a small building time, a small space consumption, a small query time and an accuracy guarantee.*

To the best of our knowledge, there is no existing work which could answer the distance queries between arbitrary points efficiently with accuracy guarantee on the terrain surface.

3 RELATED WORK

In this section, we review the existing studies about the query processing on the terrain surface including the distance queries, k nearest neighbor queries, reverse k nearest neighbor queries, etc. Since this paper focuses on the distance query, Section 3.1 and Section 3.2 present the existing studies on the distance query, where Section 3.1 surveys *on-the-fly algorithms* (which compute the distance online without any preprocessing) and Section 3.2 summarizes the existing work on *index-based algorithms* (which accelerate the distance query processing with a pre-computed indexing structure). Section 3.3 introduces some other related studies. Section 3.4 finally compares the existing studies with this paper.

3.1 On-the-fly Algorithms

There are three existing on-the-fly algorithms of finding the exact geodesic shortest distance between two arbitrary points on terrain surfaces [8, 29, 40]. Their time complexities are $O(N^2 \log N)$, $O(N^2)$ and $O(N^2 \log N)$, respectively, which are prohibitively large and prevent their usage in many real-world applications.

Motivated by this, a lot of research effort was put into the development of the approximate distance computation which could have a smaller time complexity [20, 21, 24]. The major idea of all existing approximate algorithms is introducing many additional auxiliary points, called Steiner points, into the terrain surface and then connecting auxiliary points with many auxiliary edges. Finally, they obtain a graph by introducing the auxiliary points and edges. They find the approximate geodesic distance through the distance computation in the graph. The running time is $O((N + N') \log(N + N'))$, where N' is the number of auxiliary points introduced. Their differences lie on their methods of the auxiliary points selection and auxiliary edges construction. The first attempt in this category, namely *Fixed Scheme* [24], is to introduce m auxiliary points uniformly in each edge of the terrain surface and establish an auxiliary edge between each pair of auxiliary points on the same face. However, its error bound has a terrain-related term. Although its empirical error is fairly small when a proper value of m is provided, it still takes a quite long time for distance query processing. The second one, namely *K-Algo* [20], adopts a similar method but has a different strategy of introducing auxiliary points on each edge. As such, it provides ϵ-approximate distance, where ϵ is a user-defined error parameter. The error is achieved through placing auxiliary points by a terrain-related geometry property. However, this property may place a large amount of auxiliary points and thus, more time is required for query processing in *K-Algo*. In addition, *K-Algo* could only be applied to unweighted terrain surfaces and does not support weighted terrain surfaces. The third one, namely *Unfixed Scheme* [4], introduces auxiliary points in each bisection of each face by using a geometry-based scheme and introduces an auxiliary edge between each pair of auxiliary points in the same vicinity. The same as the second one, it also provides ϵ-approximate distance. However, for the faces with sharp angles, the *Unfixed Scheme* will also place quantities of auxiliary points which causes slow query response. Compared with the exact methods, the approximate ones have a highly boosted running time but the running time is still too large to be used in real-time applications.

3.2 Index-Based Algorithms

To further boost the distance query processing, the index-based methods were proposed [4, 19, 39]. [19] proposed a *Single-Source All-Destination* (SSAD in short) indexing structure, which indexes all distances between a fixed source point and any other point on the terrain surface. It has a $O(N^2 \log N)$ building time, $O(N)$ space consumption and a $O(1)$ query time. But, it is limited to a fixed source point which must be given before the building of the index and could not be changed in the query phase. Then, [4] proposed an indexing structure, namely *Steiner Point-Based Oracle* (*SP-Oracle* in short), which is an index-based version of *Unfixed Scheme*. Note that, in this paper, the additional auxiliary points added are also called *Steiner points* and the graph obtained from the auxiliary points and edges is called *Steiner graph*. *SP-Oracle* builds an indexing structure for the exact distance query processing on the Steiner graph. It provides ϵ-approximate distance and the complexities of its building time, space consumption and query time are $O(\frac{N \log \frac{1}{\epsilon}}{sin^2(\theta)\epsilon^{2.5}} \log^3(\frac{N \log \frac{1}{\epsilon}}{sin(\theta)\epsilon^{1.5}}) \log^2 \frac{1}{\epsilon})$, $O(\frac{N}{\sin(\theta)\epsilon^{1.5}} \log^2(\frac{N}{\epsilon}) \log^2 \frac{1}{\epsilon})$ and $O(\frac{1}{\sin(\theta)\epsilon} \log \frac{1}{\epsilon} + \log\log N)$, respectively, where θ is the minimum inner angle among all faces on the terrain surface. The state-of-the-art algorithm, namely *SE-Oracle* [39], focuses on POI-to-POI distance query where the distance query must be performed among several pre-defined POIs known apriori. It proposed an oracle for the POI-to-POI query based on a technique called *Well-Separated Pair Decomposition*. But, for the arbitrary point-to-arbitrary point query, it must introduce the Steiner points in the same way as *SP-Oracle*. In this case, *SE-Oracle* indexes the Steiner points in place of POIs and builds their oracle on the Steiner points. As such, it inevitably has a prohibitively large building time and space consumption. According to our experiments, it consumes more than 256GB of memory and takes more than 3 hours to build in a dataset with only 10,243 vertices. Besides, it only applies to an unweighted terrain and could not be able to apply to a weighted terrain. It also provides ϵ-approximate distance. The complexities of its building time, space consumption and query time are $O(\frac{N \log^2 N}{\epsilon^{2\beta}} + \frac{Nh \log \frac{1}{\epsilon}}{sin(\theta)\sqrt{\epsilon}} \log \frac{N \log \frac{1}{\epsilon}}{sin(\theta)\sqrt{\epsilon}} + \frac{Nh \log \frac{1}{\epsilon}}{sin(\theta)\sqrt{\epsilon}\epsilon^{2\beta}})$, $O(\frac{Nh \log \frac{1}{\epsilon}}{sin(\theta)\sqrt{\epsilon}\epsilon^{2\beta}})$ and $O(\frac{h}{sin(\theta)\epsilon})$, respectively. Here h and β are two terrain related parameters in [39] and in practice, $h < 30$ and $\beta \in [1.5, 2]$. In conclusion, all existing index-based methods suffer from their unscalable building time and space consumption.

3.3 Other Related Studies

We review some other related studies [12, 13, 20, 21, 32, 41, 42] on the terrain surface in this section. Specifically, [12, 13] studies the kNN query processing and it proposed to use a simplified low-resolution terrain surface constructed from the original data for pruning some irrelevant regions so that the

Table 1: Comparison of Our Algorithm and Existing Algorithms

Algorithm	Building Time	Space Consumption	Query Time
Fixed Scheme [24]	-	-	$O(mN\log(mN))$
K-Algo [20]	-	-	$O(\frac{L_{max}^3 N}{(L_{min}\epsilon\cdot\sqrt{1-cos\theta})^3}+\frac{L_{max}N}{L_{min}\epsilon\cdot\sqrt{1-cos\theta}}\log(\frac{L_{max}N}{L_{min}\epsilon\cdot\sqrt{1-cos\theta}}))$
SP-Oracle [4]	$O(\frac{N\log\frac{1}{\epsilon}}{sin^2(\theta)\cdot\epsilon^{2.5}}\log^3(\frac{N\log\frac{1}{\epsilon}}{sin(\theta)\cdot\epsilon^{1.5}})\log^2\frac{1}{\epsilon})$	$O(\frac{N\log\frac{1}{\epsilon}}{sin^2(\theta)\cdot\epsilon^2}\log^2\frac{N\log\frac{1}{\epsilon}}{sin(\theta)\cdot\epsilon^{1.5}}\log^2\frac{1}{\epsilon})$	$O(\frac{1}{sin(\theta)\cdot\epsilon}\log\frac{1}{\epsilon}+\log\log(\frac{N}{sin(\theta)\cdot\sqrt{\epsilon}}))$
SE-Oracle [39]	$O(\frac{N\log^2 N}{\epsilon^{2\beta}}+\frac{Nh\log\frac{1}{\epsilon}}{sin(\theta)\cdot\sqrt{\epsilon}}\cdot\log\frac{N\log\frac{1}{\epsilon}}{sin(\theta)\sqrt{\epsilon}}+\frac{Nh\log\frac{1}{\epsilon}}{sin(\theta)\cdot\sqrt{\epsilon}\cdot\epsilon^{2\beta}})$	$O(\frac{Nh\log\frac{1}{\epsilon}}{sin(\theta)\cdot\sqrt{\epsilon}\cdot\epsilon^{2\beta}})$	$O(\frac{h}{sin(\theta)\cdot\epsilon})$
EAR-Oracle (Proposed)	$O(\lambda\gamma\zeta mN\log(mN))+\frac{N\log(N)}{\epsilon^{2\beta}}+Nh\log(N)+\frac{Nh}{\epsilon^{2\beta}})$	$O(\frac{m\lambda\gamma N}{\zeta}+\frac{Nh}{\epsilon^{2\beta}})$	$O(\lambda\gamma\zeta\log(\lambda\gamma\zeta))$

Remark: δ is the additive error of Fixed Scheme and according to the existing studies [2, 6, 10] and our experiment, δ is a significantly small value which is very close to 0. The additive error of EAR-Oracle is 2δ. The additive error of any other algorithm is 0. The multiplicative error of Fixed Scheme is 0 and the multiplicative error of any other algorithm is ϵ, where ϵ is a user-defined error parameter.

kNN queries could be accelerated. [32] proposed a Voronoi diagram-based method for kNN queries and [41] studies the monitoring of the kNN in a dynamic setting and [42] studies reverse k nearest neighbors queries. [21] studies the problem of finding the shortest geodesic path satisfying a slope constraint. Note that all algorithms in this section have a different problem setting to our problem.

3.4 Comparison

We compare our oracle with the existing works in terms of building time, space consumption, query time, error bound and if they support weighted terrains or not for the arbitrary point-to-arbitrary point distance query on the terrain surface. The results are shown in Table 1. The explanations of some notations could be found in Section 4.6.1 and d_g denotes the exact geodesic distance. In the table, m denotes the number of Steiner points introduced in each edge on the terrain surface by *Fixed Scheme* and our oracle. ζ is the square root of the amount of boxes in *EAR-Oracle*. Let L_{max} (resp. L_{min}) denotes the length of the longest (resp. shortest) edges of the terrain surface. θ is the minimum inner angle of any face on the terrain surface. ϵ is a user-defined parameter regarding the error bound. h is the height of a tree structure called *partition tree* in *SE-Oracle* [39] and its value is a small constant smaller than 30 in their experiments. β is the intrinsic dimension of the terrain surface and according to the experiments in [39], $\beta \in [1.5, 2]$. d_M^+ (resp. d_M^-) denotes the largest (resp. smallest) Minkowski distance when p reaching infinity (resp. negative infinity) among all vertex pairs. λ is the maximum number of vertices covered by the minimum square with side length d_M^- and γ is the ratio of d_M^+ and d_M^-. λ and γ are both determined by the terrain property.

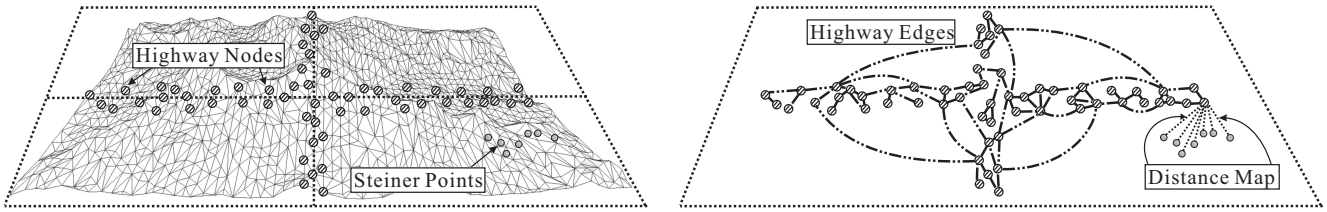
We highlight some comparisons in Table 1 as follows. (1) For the building time and space consumption, *SP-Oracle* and *SE-Oracle* are not scalable since they build their indexing structure upon all Steiner points introduced whose amount is orders of magnitudes more than the vertices on the terrain surface. But our oracle has a small building time and space consumption. Besides, their building time and space are quite sensitive to the parameter θ and will be prohibitively large on the terrain surface with many "sharp" faces (i.e., the faces with very small inner angles). They are more sensitive to ϵ than our oracle and our oracle has scalable building time and space consumption. (2) For the query time, the *Fixed Scheme*, *Unfixed Scheme* and *K-Algo* have very query time complexities (i.e., larger than $O(N)$) which is obviously much larger than those of the other three oracles (which are small constants). (3) For the error bound, we observe that each algorithm has a very small error bound given that the parameter ϵ is small. Note that δ is the additive error bound of *Fixed Scheme* which is negligible according to the existing studies [6, 10, 24] and our experiment also verified this. (4) *Fixed Scheme*, *Unfixed Scheme*, *SP-oracle* and our oracle support weighted terrains but the other two do not support them.

4 OUR PROPOSED ALGORITHM: EAR-ORACLE

In this section, we detail our proposed indexing structure, namely *Efficient Arbitray Point-to-Arbitrary Point Geodesic Distance Oracle* (*EAR-Oracle* in short), for the geodesic distance query processing. In Section 4.1, we briefly give an overview of the algorithm framework. In Section 4.2 and Section 4.3, we present the two components in our oracle, namely *highway nodes* and *highway edges* of our highway network-based algorithm. In Section 4.4, we demonstrate our distance query processing algorithm. Section 4.5 presents the implementation details involved. Finally, in Section 4.6, we present the theoretical analysis on the building time, space consumption, query time and error bound of our oracle.

4.1 Overview: A Highway Network-based Approach

In a nutshell, our oracle is a highway network-based indexing structure, in which a highway network is proposed to bridge the distant local regions and boost the query processing. Figure 2 shows the framework of our technique. As Figure 2(a) shows, our algorithm partitions the terrain surface into disjoint regions (i.e., boxes) and carefully selects several "hub" points (marked by the shaded points), namely *highway nodes*, on the boundary of each region and we introduce several auxiliary points, namely *Steiner points*, in each region. Each highway node has a component called *distance map* which stores the distances between the highway node and all Steiner points introduced in the same region (as shown in Figure 2(b)). It is worth mentioning that although the Steiner points are introduced, they are only visible to the highway nodes in the



(a) Highway Nodes and Steiner Points (b) Highway Edges and Distance Map

Figure 2: Framework of EAR-Oracle

local box and the indexing cost of them will be *lightweight*. The placement of the Steiner points and the distance map will be detailed later. All highway nodes form a *highway node set* which is a subset of the terrain vertex set V. Figure 2(b) also shows the whole highway network which is a sparse graph consisting of the *highway nodes* (defined by the boundaries of the boxes) and the *highway edges*, each of which connects two highway nodes.

In the query phase, we consider two types of queries. The first one is called *inner-box query* where the distance query is performed between two points in the same box. The other is called *inter-box query* where the distance query is performed between two points in two different boxes.

For the first type, the inner-box query, we adopt an online local search between the two query points in the local region defined by the box (which will be presented in detail later). Since the two query points are quite close to each other, the online local search is efficient.

For the second type, the inter-box query, we utilized the highway network and the distance map to accelerate the query processing. As such, in the query phase, the search starting from the source point s could be able to 'jump' to a distant region by using the highway network and the query processing could be largely accelerated. Specifically, we adopt the highway network and the distance map to construct a graph called *query graph* (which also contains s and t as two of its vertices) and perform Dijkstra's algorithm on this query graph to obtain the distance.

As could be understood from the above description, our oracle contains two components, namely *highway nodes* and *highway edges*, which will be present in Section 4.2 and Section 4.3, respectively.

4.2 Highway Nodes

To extract the highway nodes, we first impose a grid on the 2D x-y plane to partition the terrain surface evenly into ζ^2 boxes (or cells) and each box corresponds to a local rectangle region on the original terrain surface.

We adopt a quadtree-based approach to generate such a grid with ζ^2 boxes . A quadtree [16] is a well-known tree-like data structure for processing 2D spatial data. Each leaf node (i.e., quad) in the quadtree corresponds to a local region in our algorithm. Each internal node has exactly four children which separate the region of their parent. Thus, a quadtree with depth ψ will generate 4^ψ leaf boxes (i.e., quads). In our case, the ζ^2 boxes could be generated by building a limited-depth quadtree with $\psi = \log(\zeta)$. In this grid, there are ζ rows and ζ columns.

For each box, we select its corresponding highway nodes as follows. We find the faces which have overlap with the boundaries of the box and select all vertices of the faces as the highway nodes of this box. The highway node set $\mathcal{V}$ consists of the highway nodes of all boxes.

Each highway node has a *distance map*. The distance map of highway node v, denoted by $M(v)$, stores the approximate distance $\tilde{d}_g(v, p)$ for each Steiner point p (i.e., each auxiliary point introduced) in the same box containing v. The introduction of Steiner points is detailed in Implementation Detail 1 in Section 4.5. The approximate distance $\tilde{d}_g(c, p)$ is the distance between c and p in a graph called *base graph* to be shown later in Implementation Detail 1 in Section 4.5. The distance $\tilde{d}_g(c, p)$ is always larger than or equal to $d_g(c, p)$. Note that the reason why we use the approximate distance here instead of the exact distance is that there is no existing algorithm for finding the exact geodesic distance on weighted terrain surfaces. According to existing studies [6, 10, 24], $\tilde{d}_g(c, p)$ is very close to $d_g(c, p)$. With this distance map, we could quickly find the approximate distance between v and each arbitrary point in the same box as v. Given a highway node v and the box containing v, we construct the *distance map* of v by using the A-SSAD-Box mentioned in the Implementation Detail 2 in Section 4.5.

4.3 Highway Edges

Given a set $\mathcal{V}$ containing all highway nodes, we build a sparse graph, namely highway network, by introducing highway edges which is a connected graph and the distance on the highway network between any two highway nodes is very close to that on the terrain surface. Before presenting the details of highway edges, we first illustrate two important concepts, namely *surface disk* and *shrunk surface disk*. Given an arbitrary point c on the terrain surface and a non-negative real value r, a surface disk, denoted by $D(c, r)$, is the set of all points whose geodesic distances to c is at most r (i.e., $D(c, r) = \{p | d_g(c, p) \leq r\}$). In Figure 3, there are two points c_1 and c_2 and there is one disk $D(c_1, r_1)$ which centers at c_1 and has radius r_1 and another disk $D(c_2, r_2)$ which centers at c_2 and has radius r_2. For the weighted terrain surfaces, it is intractable to compute the exact

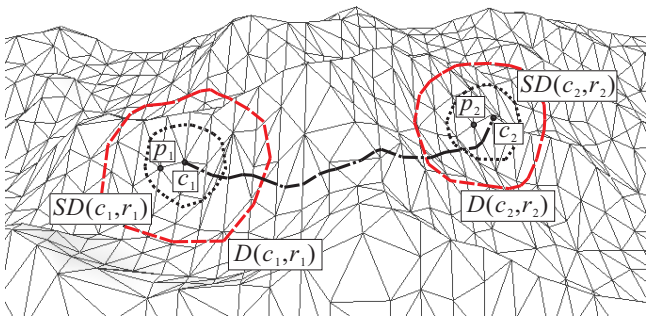
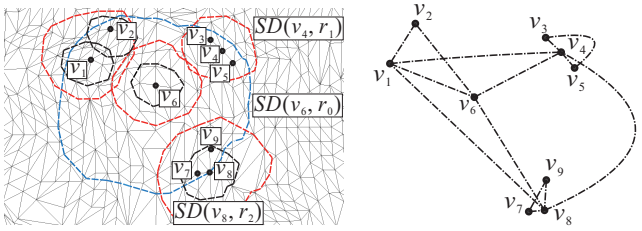


Figure 3: Surface Disks and Shrunk Surface Disks

value of $d_g(c,p)$. Thus, we use an approximate single-source all-destination algorithm to find the approximate distance $\tilde{d}_g(c,p)$ between the center c and another point p. Recall that $\tilde{d}_g(c,p)$ is the distance on the base graph which is always larger than or equal to $d_g(c,p)$. Based on the distance $\tilde{d}_g(c,p)$, we could define the *shrunk surface disk*. Given an arbitrary point c on the terrain surface and a non-negative real value r, a shrunk surface disk, denoted by $SD(c,r)$, is the set of all points whose approximate distance to c is at most r (i.e., $SD(c,r) = \{p|\tilde{d}_g(c,p) \leq r\}$). Figure 3 shows two shrunk surface disks $SD(c_1,r_1)$ and $SD(c_2,r_2)$. They are respectively contained by the surface disks $D(c_1,r_1)$ and $D(c_2,r_2)$ which respectively have the same center and radius with them. The approximate single-source all-destination algorithm to be described later in Section 4.5 will also allow us to find all highway nodes in the shrunk surface disk $SD(c,r)$. Given two shrunk disks $SD(c_1,r_1)$ and $SD(c_2,r_2)$, we call them *well-separated* if $\tilde{d}_g(c_1,c_2) \geq (\frac{2}{\epsilon}+2)\max(r_1,r_2)$, where ϵ is the error control parameter defined by the user. For any point $p_1 \in SD(c_1,r_1)$ and $p_2 \in SD(c_2,r_2)$, the approximate distance of these two points $\tilde{d}_g(p_1,p_2)$ could be approximate by the approximate distance of the two shrunk surface disk centers $\tilde{d}_g(c_1,c_2)$ with an ϵ bounded error guarantee according to existing works [5, 34, 39].

Based on the concept of shrunk surface disk, we present a key concept called *well-separated disk pair decomposition* (*WSDPD*) next. Given a set $\mathcal{V}$ of highway nodes on the terrain surface, we call a set $\mathcal{O}$ a *well-separated disk pair decomposition* (*WSDPD*) of $\mathcal{V}$ if $\mathcal{O}$ satisfies the following properties: (1) $\mathcal{O}$ consists of several disk pairs, each of which is a pair of shrunk surface disks on the terrain surface in the format of $(SD(c_1,r_1), SD(c_2,r_2))$. (2) For each pair $(SD(c_1,r_1), SD(c_2,r_2))$ of shrunk surface disks contained in $\mathcal{O}$, the two disks are well-separated and the distance $\tilde{d}_g(c_1,c_2)$ is stored in this pair. (3) For any pair (v_1,v_2) of highway nodes, each of which is from $\mathcal{V}$, there is exactly one pair $(SD(c_1,r_1), SD(c_2,r_2))$ in $\mathcal{O}$ such that (i) v_1 is in $SD(c_1,r_1)$ and (ii) v_2 is in $SD(c_2,r_2)$.

Recall that the vertices of the highway network consist of all highway nodes. Given a WSDPD $\mathcal{O}$ of $\mathcal{V}$, we build the highway edges as follows. For each pair $(SD(c_1,r_1), SD(c_2,r_2))$ in $\mathcal{O}$, we establish an edge between c_1 and c_2 and the weight of the edge is assigned to be the associated distance $\tilde{d}_g(c_1,c_2)$. We will show in Section 4.6 that the highway network is a connected



(a) Shrunk Surface Disks **(b) Highway Network**

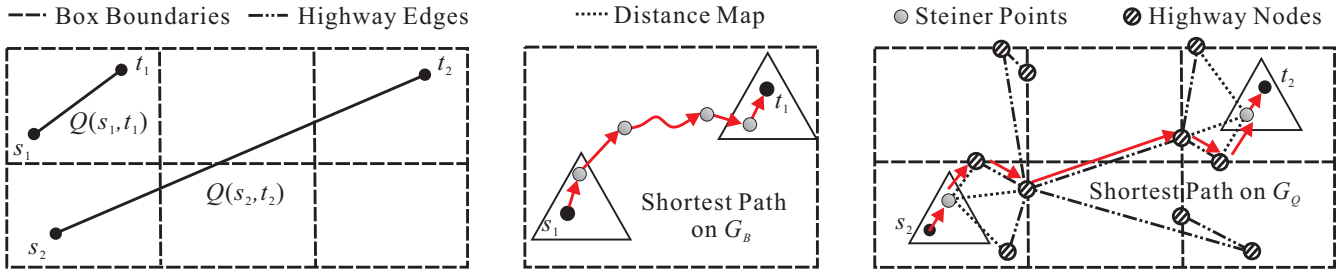
Figure 4: WSDPD and Corresponding Highway Network

and sparse graph and the distance between any two highway nodes on the highway network is very close to their distance on the base graph with a very small error bound. Consider the example in Figure 4. There are totally 9 highway nodes, namely $v_1, v_2, ..., v_9$. Figure 4(a) shows all shrunk surface disks involved in the WSDPD of all highway nodes. For each disk pair in the WSDPD, we establish a highway edge and the final highway network has all highway nodes as its vertices and has all highway edges established as its own edges. The highway network in the example is shown in Figure 4(b).

The only issue left is how to build the WSDPD for the highway node set $\mathcal{V}$. We adopt an existing method in *SE-Oracle* [39] to obtain this WSDPD. In our case, it treats all highway nodes as the POIs when we apply the building method of *SE-Oracle* and we adopt the shrunk surface disk in place of the surface disk in the original method and we replace the SSAD algorithm in the original method with our A-SSAD-Disk (to be introduced in Implementation Detail 2 later). Then, we extract the second component in *SE-Oracle* which is a pair set containing several pairs of shrunk surface disks. Finally, we put all pairs of disks in the pair set into our WSDPD. Our theoretical analysis in Section 4.6 shows that the size of the WSDPD is linear to the number of highway nodes.

4.4 Distance Query Processing

For the inner-box query where the source point s and the destination point t are located in the same box, we apply the A-SSAD-Query method mentioned in the Implementation Detail 2 in Section 4.5 to obtain the distance $\tilde{d}_g(s,t)$ as the output. For the inter-box query where the source point s and the destination point t are located in different boxes, we perform the query processing as follows. First, we find the face f_s containing s and build a graph G_s as follows. Let $\kappa = \lfloor(\frac{4(m+1)}{\sqrt{3}}\rfloor \approx \lfloor 2.309(m+1))\rfloor$ be a constant. To obtain G_s, we first insert s as a new vertex into G_B and establish an edge between s and each Steiner point p on f_s into G_B and the weight of the edge is assigned to be $d_e(s,p) \cdot w(f_s)$, where $d_e(s,p)$ denotes the Euclidean distance between s and p. Recall that we introduce the Steiner points in each face with the method in Implementation Detail 1 in Section 4.5. The vertices of G_s consist of s, all Steiner points within κ hops from s on G_B (with s and aforementioned edges inserted) and all highway nodes in the box containing s. After that, we perform a Breadth First Search (BFS) with κ hops on base graph G_B which takes s as the source point and add all



(a) Shortest Distance Queries (b) Inner-Box Query (c) Inter-Box Query

Figure 5: Query Processing for the Inner-Box and Inter-Box Queries

visited edges to G_s. Finally, for each highway node v in the box containing s and each Steiner points p on f_s, we establish an edge connecting v and p on G_s and the weight of the edge is the distance between v and p in the distance map of v. Similarly, we could obtain a graph G_t for the destination point t but all edges in the reverse direction. Then, we merge the highway network $\mathcal{G}$, G_s and G_t together and obtain a graph G_Q. Finally, we perform a Dijkstra's algorithm with s as the source and t as the destination and return the s-t distance on G_Q.

Consider the example as shown in Figure 5. Figure 5(a) shows two shortest distance queries. The first query $Q(s_1, t_1)$ is an inner-box query. Figure 5(b) shows the path that A-SSAD-Query finds in the base graph G_B which contains a sequence of Steiner points and the length of the path will be the distance returned. The second query $Q(s_2, t_2)$ is an inter-box query and the shortest path of them passes through the highway network. Figure 5(c) shows the shortest path from s_2 to t_2 in the graph G_Q (which is merged from G_s, G_t and the highway network $\mathcal{G}$) and the length of the path will be the distance returned. This path could be efficiently found by performing a shortest path query on G_Q since the size of G_Q is very small compared with the base graph G_B.

4.5 Implementation Details

This section presents the implementation details of the *EAR-Oracle* as follows.

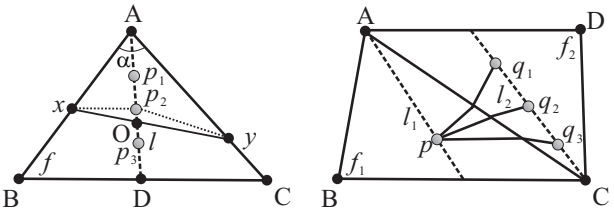
Implementation Detail 1: The introduction of Steiner points and the construction of base graph. The base graph G_B is constructed as follows. We first introduce the Steiner points on each face f through a *bisector-fixed scheme*. Each bisector of the face is defined by two adjacent edges who share a common vertex on the terrain surface. We call two bisectors share a common edge if and only if there exists an edge defines both of them. For each bisector of the given face f, we uniformly introduce m points on the bisector. After placing m Steiner points on each bisector, for each pair of adjacent faces f and f', we introduce a Steiner edge between each Steiner point v in f and each Steiner point v' in f' if the bisectors that they lie on share a common edge. The weight of each Steiner edge is calculated by using the so-called *Snell's Law* method in [4]. For the sake of space, we refer the readers to [4] for the detailed operations of building the Steiner edges. It is worth mentioning that although we build a base graph with Steiner points introduced, we do not build any indexing structure upon the base graph. This is the reason why our oracle is scalable in terms of building time and space.

Implementation Detail 2: The approximate single-source all destination algorithm (A-SSAD). The algorithm first introduces a new vertex which is the source point c into the base graph G_B. Then, it introduces a Steiner edge between each Steiner point p on f and c and the weight of the edge is assigned to be $d_e(c, p) \cdot w(f)$, where f is the face containing c and $d_e(c, p)$ denotes the Euclidean distance between c and p. Then, this algorithm simply visits all vertices on G_B by using a Dijkstra's algorithm with c as the source vertex. The A-SSAD algorithm has three variants, namely *A-SSAD-Disk*, *A-SSAD-Box* and *A-SSAD-Query*. A-SSAD-Disk starts from a given source point c and terminates when a given disk $SD(c, r)$ is fully visited and returns the list containing all highway nodes within the disk. A-SSAD-Box starts from a given source point c and terminates when all Steiner points in a given box are fully visited and returns the distance $\tilde{d}_g(c, p)$ between c and each Steiner point p in the box. The A-SSAD-Query starts from a given source point s and terminates when the given destination point t is visited. In the A-SSAD-Query, when the face containing t is visited, besides all Steiner points and Steiner edges introduced in the original A-SSAD algorithm, it also introduces an additional Steiner point t and establishes a Steiner edge connecting t and each Steiner point in the face. It finally returns the distance $\tilde{d}_g(s, t)$ in the graph G_B by using Dijkstra's algorithm.

4.6 Theoretical Analysis

We present the theoretical analysis on *EAR-Oracle* in this section. Section 4.6.1 highlights the important notations to be used for reference. Section 4.6.2 and Section 4.6.3 present the analysis of the building time and space consumption, and the analysis of query time and accuracy, respectively.

4.6.1 Notations. We highlight the important notations here for reference. Let m denote the number of Steiner points placed on each bisector and N denote the number of vertices on the terrain surface. The error control parameter is denoted by ϵ. Recall that the number of boxes is denoted by ζ^2 (i.e., ζ is the square root of the number of boxes). α is the maximum ratio of faces inside a box. Let d_M^+ (resp. d_M^-) denote the largest (resp. smallest) Minkowski distance when p reaches infinity (resp.



(a) Segment on Single Face (b) Edges between Faces

Figure 6: Steiner Points and Edges in Base Graph

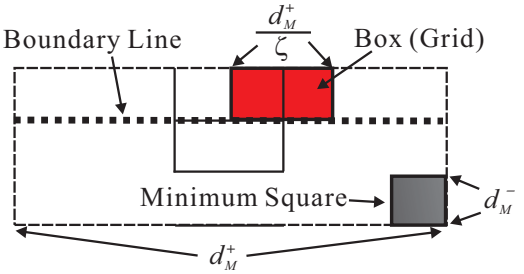


Figure 7: Grid-Based Partition Example

negative infinity) among all vertex pairs. γ is the ratio of d_M^+ and d_M^-. The maximum number of highway nodes covered by a square with side length d_M^- is denoted by λ. The highway node set is denoted by $\mathcal{V}$ and the base graph in *EAR-Oracle* is denoted by G_B.

4.6.2 Building Time and Space Consumption.

Lemma 1. *In the base graph constructed by bisector-fixed scheme, the number of additional Steiner points introduced is $O(mN)$ and the construction time of the base graph is $O(m^2N)$.*

Proof. The *bisector-fixed scheme* places m Steiner points on each bisector of each face on the terrain. Each bisector is defined by two edges sharing a common point. Since each face is a triangle, it has exactly three bisectors. Thus, the total number of introduced Steiner points is $m \cdot 3|F| = 3m|F| = O(m|F|)$. Since the terrain surface is a planar graph, we obtain that $|F| = O(N)$ and the number of Steiner points is $O(mN)$. Figure 6(a) shows an example of Steiner points for face f when $m = 3$. In this example, the bisector l is defined by edges AB and AC who share a common point A. Recall that two bisectors share the same edge if and only if there exist one edge defines both of them. As described in Implementation Detail 1, the Steiner edges of the base graph will be only established for the bisectors who share a common edge. As Figure 6(b) shows, there are two adjacent faces f_1 and f_2. The two bisectors l_1 and l_2 share a common edge since they are both defined by edge AC. For the Steiner point p on l_1, the Steiner edges will be established between p and all Steiner points on l_2 (i.e., q_1, q_2 and q_3). By geometry property, for each bisector l, there are at most seven bisectors who share a common edge with l. Thus, each Steiner point will only connect to the Steiner points from at most seven bisectors. So the base graph construction requires $O(mN \times 7m) = O(m^2N)$ time. □

Lemma 2. *The size of the highway node set $|\mathcal{V}| = O(\lambda\gamma\zeta)$ and each box contains $O(\frac{\lambda\gamma}{\zeta})$ highway nodes.*

Proof. For two points u and v, let $d_m^+(u,v) = max(x_u - x_v, y_u - y_v)$, $d_m^-(u,v) = min(x_u - x_v, y_u - y_v)$ be the Minkowski distance of p reaching infinity and negative infinity, respectively. Recall that $d_M^+ = \max_{u,v \in V} d_m^+(u,v)$ and $d_M^- = \min_{u,v \in V} d_m^-(u,v)$. From the definition of γ (in Section 4.6.1), we have $\gamma = \frac{d_M^+}{d_M^-}$.

Figure 7 shows an example of a terrain surface on the x-y plane. d_M^+ is maximum side length of the entire terrain surface. d_M^- is the side length of a minimum square. The red box is a grid which is generated according to Implementation Detail 1. The largest side length of a box is $\frac{d_M^+}{\zeta}$.

We call a line segment a *boundary line* if (1) the line segment is axis-parallel, (2) it overlaps with the boundary of at least one box and (3) the two end points lies on the boundary of the terrain surface. Consider the example as shown in Figure 7. It shows a boundary line passing a box (i.e., the red box).

By the geometric property, a line with the length d_M^+ has overlap with at most $2 \cdot \frac{d_M^+}{d_M^-} = O(\gamma)$ minimum squares (see Figure 7 for example).

By the definition of λ (see Section 4.6.1 for its definition), each minimum square has overlap with at most λ highway nodes. Thus, a line with the length d_M^+ has overlap with $O(\lambda\gamma)$ highway nodes (See Figure 7 for example). Since we have exactly ζ boxes in a single row or column, the number of boundary lines is $2 \cdot (\zeta + 1)$. Thus, the total number of highway nodes is $O(\lambda\gamma 2 \cdot (\zeta + 1)) = O(\lambda\gamma\zeta)$.

For each box, it has four sides. Consider the side with largest side length. The length of this side is $\frac{d_M^+}{\zeta}$ and it has overlap with at most $\frac{d_M^+}{d_M^-\zeta}$ minimum squares. Since each minimum square contains at most λ highway nodes, the number of highway nodes overlapped with a side is with at most $\frac{\lambda\gamma}{\zeta}$. Thus, the number of highway nodes overlapped with a box is $4 \cdot \frac{\lambda\gamma}{\zeta} = O(\frac{\lambda\gamma}{\zeta})$. □

Lemma 3. *The building time and space consumption of the distance map are $O(\lambda\gamma\zeta mN \log(mN)))$ and $O(\frac{m\lambda\gamma N}{\zeta})$, respectively.*

Proof. The distance map contains the distance between highway nodes and Steiner points in their corresponding boxes. For each highway node, we invoke the A-SSAD-Box algorithm to compute the distance map. Let $\mathcal{V}_{box}$ be the maximum number of highway nodes of a box. As such, the building time and space consumption of the distance map are $O(|\mathcal{V}| \cdot T_A)$ and $O(|\mathcal{V}_{box}| \cdot n_S)$, where T_A denotes the running time of A-SSAD-Box algorithm and n_S denotes the number of Steiner points. By Lemma 2, we obtain that $|\mathcal{V}|$ is $O(\lambda\gamma\zeta)$ and $|\mathcal{V}_{box}|$ is $O(\frac{\lambda\gamma}{\zeta})$. Then, we analyze T_A and n_S as follows. Since A-SSAD-Box is a Dijkstra's algorithm on the sub-graph of our base graph, we obtain that T_A is $O(mN \log(mN))$ and n_s is $O(mN)$ by Lemma 1. Thus, we obtain that the building time and space consumption

of distance map are $O(\lambda\gamma\zeta mN\log(mN))$ and $O(\frac{m\lambda\gamma N}{\zeta})$, respectively. □

Theorem 1. *Let h be a constant determined by the terrain and β be the intrinsic dimension of the terrain. The building time of EAR-Oracle is $O(\lambda\gamma\zeta mN\log(mN)) + \frac{N\log(N)}{\epsilon^{2\beta}} + Nh\log N + \frac{Nh}{\epsilon^{2\beta}})$.*

Proof. The building time of *EAR-Oracle* consists of three parts: (1) Base graph (2) Highway nodes (including highway nodes extraction and distance map construction) (3) Highway edges.

For the first part, by Lemma 1, the base graph could be constructed in $O(m^2N)$ time .

For the second part, to extract all highway nodes, we adopt a quadtree based algorithm (introduced in Section 4.2). Since there are ζ^2 boxes, the depth of the quadtree is $\log(\zeta)$. Thus, the time complexity of this part is $O(\log(\zeta)N)$.

For the distance map construction, by Lemma 3, this procedure could be accomplished in $O(\lambda\gamma mN\log(mN))$ time.

For the last part, highway edges construction, we combine (a) Theorem 3 in [39] (which shows the building time and space consumption of *SE-Oracle*), (b) the fact that the POIs that *SE-Oracle*s indexes in our case are the highway nodes, (c) the technique that we substitute the original surface disk (resp. SSAD algorithm) in the original *SE-Oracle* with our shrunk surface disk (resp. A-SSAD-Disk) and finally, we obtain that the time complexity of our highway network generation is $O(\frac{N\log(N)}{\epsilon^{2\beta}} + Nh\log(N) + \frac{Nh}{\epsilon^{2\beta}})$.

By combining the results of the above components we obtain that the build time of *EAR-Oracle* is: $O((m^2 + \log(\zeta))N + \lambda\gamma\zeta mN\log(mN)) + \frac{N\log(N)}{\epsilon^{2\beta}} + Nh\log(N) + \frac{Nh}{\epsilon^{2\beta}})$. For simplicity we omitted the first term in the big O notation since m, $\log(\zeta)$ are relative small compared with other terms. □

Theorem 2. *Let h be a constant determined by the terrain and β be the intrinsic dimension of the terrain. The space consumption of EAR-Oracle is $O(\frac{m\lambda\gamma N}{\zeta} + \frac{Nh}{\epsilon^{2\beta}})$.*

Proof. The space consumption of *EAR-Oracle* contains: (1) Base graph. (2) Highway nodes (including the space of distance map) and (3) Highway edges.

For the first part, by Lemma 1, we obtain that the space consumption of the base graph is $O(m^2N)$.

For the second part, by Lemma 3, we obtain the space consumption of the distance map is $O(\frac{m\lambda\gamma N}{\zeta})$.

For the last part, highway edges, we combining (a) Theorem 3 in [39] (which shows the building time and space consumption of *SE-Oracle*), (b) the fact that the POIs that *SE-Oracle* indexes in our case are the highway nodes and finally, we obtain that the space complexity of the highway network is $O(\frac{Nh}{\epsilon^{2\beta}})$.

By combining the space consumption of the above components, we obtain that the space complexity of *EAR-Oracle* is $O(m^2N + \frac{m\lambda\gamma N}{\zeta} + \frac{Nh}{\epsilon^{2\beta}}) = O(\frac{m\lambda\gamma N}{\zeta} + \frac{Nh}{\epsilon^{2\beta}})$ since m is a small constant. □

4.6.3 Query Time and Accuracy.

Lemma 4. *The highway network $\mathcal{G}$ in EAR-Oracle is a connected graph. Let $d_{\mathcal{G}}(u,v)$ denote the shortest distance between any two vertices u and v in $\mathcal{G}$. For any pair of highway nodes v_1 and v_2, $d_{\mathcal{G}}(v_1,v_2) \le (1+\epsilon)\tilde{d}_g(v_1,v_2)$, where ϵ is a user-defined error bound.*

Proof. We recall the **Lemma 1.6** in [34] as follows:

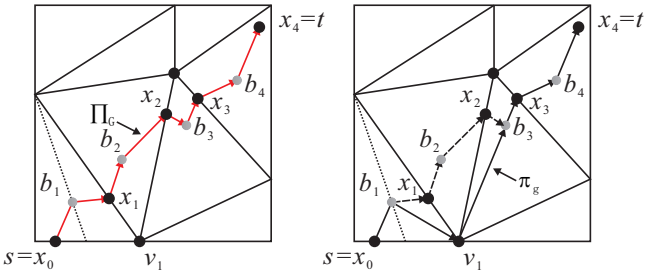
Lemma 5. *Given a set V of points and a function $d(\cdot,\cdot)$ for measuring the pairwise distances of the points in V. Let $\mathcal{P}$ denote the node pair set of the SE-Oracle constructed on V by using $d(\cdot,\cdot)$ as the distance metric with the error parameter equals to ϵ. The graph $\mathcal{G}$ (with V as its vertices and its edges $E = \{(c_x, c_y)|(x,y) \in \mathcal{P}\}$, where c_x and c_y denote the centers of the nodes x and y) is connected and $d_{\mathcal{G}}(v_1,v_2) \le (1+\epsilon)d(v_1,v_2)$ for any $v_1, v_2 \in V$.*

By applying Lemma 5 to our highway network $\mathcal{G}$ (in our case, V is the set containing all highway nodes and the function $d(\cdot,\cdot)$ is $\tilde{d}_g(\cdot,\cdot)$), $\mathcal{G}$ is a connected graph and $d_{\mathcal{G}}(v_1,v_2) \le (1+\epsilon)\tilde{d}_g(v_1,v_2)$ for any two vertices v_1 and v_2 in $\mathcal{G}$. □

Lemma 6. *Given two arbitrary surface points s and t. Let k be the number of faces passed by the geodesic path $\Pi_g(s,t)$ and L_{max} be the maximum edge length of the terrain surface. The approximate distance returned by the base graph $\tilde{d}_g(s,t) \le d_g(s,t) + \frac{\sqrt{3}}{2(m+1)}kL_{max}$.*

Proof. According to Stewart's theorem, the maximum length among all bisectors is $\frac{\sqrt{3}}{2}L_{max}$. Figure 6(a) shows a bisector on face f. According to the *bisector-fixed scheme* in Implementation Detail 1, each bisector is evenly partitioned by m Steiner points. Thus, the length of each partition is at most $\frac{\sqrt{3}}{2(m+1)}L_{max}$. The geodesic path $\Pi_g(s,t)$ along the terrain surface consisting of several connected line segments. Each of them locate in a unique face. We prove the error bound of a single line segment first and the total error bound could be calculated by summation the error bound of all faces passed by $\Pi_g(s,t)$. W.l.o.g, consider the line segment on face f with end points x and y shown in Figure 6(a). O is the intersection point of xy and AD. We select the Steiner point which is closest to O on AD for analysis (p_2 in the example). From the triangle inequality, we have $|xp_2| \le |xO| + |p_2O|$ and $|p_2y| \le |Oy| + |p_2O|$. By adding these two inequalities we obtain that $|xp_2| + |p_2y| \le |xy| + 2|p_2O|$. Since $|p_2O| \le \frac{\sqrt{3}}{4(m+1)}L_{max}$, there exists a line segment on f which has an additive error and it equals to $\frac{\sqrt{3}}{2(m+1)}L_{max}$. $\Pi_g(s,t)$ passes k faces thus there exits a surface path $\pi'_g(s,t)$ with length $\tilde{d}_g{}'(s,t)$ (by replacing each line segment in $\Pi_g(s,t)$ in the way described above) whose distance error is $\frac{\sqrt{3}}{2(m+1)}kL_{max}$ in the base graph G_B. Since $\tilde{d}_g(s,t)$ is the shortest path from s to t in the base graph, we obtain that $\tilde{d}_g(s,t) \le \tilde{d}_g{}'(s,t) \le d_g(s,t) + \frac{\sqrt{3}}{2(m+1)}kL_{max}$. □

Consider a path from s to t in the query graph G_Q (for inter-box query processing in Section 4.4) and we order the vertices on the path in the ascending order of their distance to s. We define the first (resp. last) vertex on the path which is also in $\mathcal{G}$



(a) (b)

Figure 8: Example of $\Pi_G(s,t)$ and $\pi_g(s,t)$

to be the escape point (i.e., the point escape from s), denoted by v_{es} (resp. the entrance point (i.e., the point entrance into t), denoted by v_{en}).

Lemma 7. *There exists a path $\pi_g(s,t)$ in G_Q and its length $d'_g(s,t) \le d_g(s,t) + \delta + 4L_{max}$, where L_{max} is the maximum length of edges in terrain edges.*

Proof. As shown in Figure 8, the shortest path $\Pi_G(s,t)$ on the base graph is shown in Figure 8(a). The path in G_Q is shown in Figure 8(b). We notice that the path in G_Q $\pi_g(s,t) = \{s, b_{i-1}, v_1, b_{i+1}, ..., b_{j-1}, v_2, b_{j+1}, ..., t\}$, where v_1 and v_2 are the escape point and entrance point, respectively. The escape point and entrance point will 'skip' the Steiner points b_i and b_j (see Figure 8 for example when $i = 2$ for the escape point v_1). We prove the distance error for v_1 here (the distance error for v_2 could be proved in the same way as v_1). By the triangle inequality, we observe that $|b_{i-1}v_1| \le |b_{i-1}b_i| + |b_iv_1|$ and $|v_1b_{i+1}| \le |b_ib_{i+1}| + |b_iv_1|$. Combine these two inequalities we obtain that $|b_{i-1}v_1| + |v_1b_{i+1}| \le |b_{i-1}b_i| + |b_ib_{i+1}| + 2|b_iv_1|$. From the Snell law, we obtain that $|b_{i-1}b_i| = |b_{i-1}x_{i-1}| + |x_{i-1}b_i|$ and $|b_ib_{i+1}| = |b_ix_i| + |x_i + b_{i+1}|$. Since $|b_iv_1| \le L_{max}$, we obtain that $|b_{i-1}v_1| + |v_1b_{i+1}| \le |b_{i-1}x_{i-1}| + |x_{i-1}b_i| + |b_ix_i| + |x_i + b_{i+1}| + 2L_{max}$. The other parts of $\Pi_G(s,t)$ and $\pi_g(s,t)$ are the same so the distance error introduced by v_1 is $2L_{max}$. Thus we have $d'_g(s,t) \le d_G(s,t) + 4L_{max}$ and by the results in [6, 10, 24], $d_G(s,t) \le d_g(s,t) + \delta$, we obtain $d'_g(s,t) \le d_g(s,t) + \delta + 4L_{max}$. □

Theorem 3. *The approximate distance returned by EAR-Oracle $d(s,t) \le (1+\epsilon)(d_g(s,t) + 2\delta)$). Where δ is an additive error less than Fixed Scheme [24]. According to existing studies [6, 10, 24], δ is a very small value compared with $d_g(s,t)$.*

Proof. The distance query could be divided into two categories. We prove the error bound for each of them as follows:

(1) For inner-box queries, the approximate distance will be returned by performing A-SSAD-Query algorithm on base graph G_B. According to [24], we obtain that $d(s,t) \le d_g(s,t) + \delta$.

(2) For inter-box queries, the distance will be calculated by querying the query graph G_Q. Let v_1, v_2 be the highway nodes corresponding to the escape and entrance points, respectively. Recall that $\kappa = \lfloor (\frac{4(m+1)}{\sqrt{3}} \rfloor \approx \lfloor 2.309(m+1)) \rfloor$ is a m related constant. According to the construction of query graph G_Q, G_s (resp. G_t) contains the Steiner points which is at most κ hops away from s (resp. t). Let k be the number of faces passed by the geodesic path from s to t.

If $k \le 2\kappa$, t will be covered by the union of G_s and G_t, so the approximate distance query is equivalent to query the base graph. Thus, we obtain that $d(s,t) \le d_g(s,t) + \delta$.

For the case of $k > 2\kappa$, $d(s,t)$ could be divided into three subpaths: s to v_1 (subpath in G_s), v_1 to v_2 (subpath in highway network $\mathcal{G}$) and v_2 to t (subpath in G_t). By Lemma 4, $d_{\mathcal{G}}(v_1,v_2) \le (1+\epsilon)\tilde{d}_g(v_1,v_2)$ and thus, $d(s,t) \le \tilde{d}_g(s,v_1) + (1+\epsilon)\tilde{d}_g(v_1,v_2) + \tilde{d}_g(v_2,t) \le (1+\epsilon)(\tilde{d}_g(s,v_1) + \tilde{d}_g(v_1,v_2) + \tilde{d}_g(v_2,t))$. These three subpaths form a surface path $\pi_g(s,t) = \{s, ..., v1, ..., v2, ..., t\}$ on base graph G_B. Let $d'_g(s,t)$ be the length of $\pi_g(s,t)$. Let L_{max} be the largest edge length among all edges of the terrain surface, we obtain that $d'_g(s,t) \le d_g(s,t) + \delta + 4L_{max}$ by Lemma 7. Thus, we have that $d(s,t) \le (1+\epsilon)d'_g(s,t) \le (1+\epsilon)(d_g(s,t) + \delta + 4L_{max})$. According to Lemma 6 and the definition of κ, we further obtain that $\delta = \frac{\sqrt{3}}{2(m+1)}kL_{max} > \frac{\sqrt{3}}{2(m+1)} \cdot \frac{8(m+1)}{\sqrt{3}} = 4L_{max}$. So we finally obtain that $d(s,t) \le (1+\epsilon)(d_g(s,t) + 2\delta)$.

By combining (1) and (2), we obtain that the returned distance $d(s,t) \le (1+\epsilon)(d_g(s,t) + 2\delta)$. □

Theorem 4. *The approximate distance of EAR-Oracle $d(s,t)$ could be returned in $O(\lambda\gamma\zeta \log(\lambda\gamma\zeta))$ time.*

Proof. We prove the query time for inner-box and inter-box queries as follows:

(1) The inner-box queries are processed by the A-SSAD-Query algorithm. It performs a Dijkstra's algorithm inside a box. A-SSAD-Query starts from s and terminates until t is reached. Since s and t in the same box, according to the definition of α (see Section 4.6.1), the number of Steiner points within a box is $O(\alpha mN)$. Thus, the time complexity of each inner-box query is $O(\alpha mN \log(\alpha mN))$.

(2) For inter-box queries, with the help of the distance map, the distance between each pair of Steiner point and highway node in the same box could be retrieved in constant time. There are $3m$ Steiner points on each face and each box has $O(\frac{\lambda\gamma}{\zeta})$ highway nodes by Lemma 2. Thus, the connection between G_s, G_t and highway network $\mathcal{G}$ could be established in $O(\frac{m\lambda\gamma}{\zeta})$ time. In addition, we need to add all Steiner points which is at most κ hops away from s and t. Since each face has at most three neighbors, the number of faces within κ hops to s or t is $O(3 \cdot 2^\kappa - 2)$. Thus, the number of Steiner points in G_s (resp. G_t) is $O((3 \cdot 2^\kappa - 2)3m) = O(2^\kappa m)$. By the construction of G_Q in Section 4.4, we obtain the size of G_Q is $|G_Q| = |G_s| + |\mathcal{G}| + |G_t| = 2^\kappa m + 1 + |\mathcal{V}| + 2^\kappa m + 1 = |\mathcal{V}| + 2^{\kappa+1}m + 2$. Thus, the processing time of an inter-box query is $O(|G_Q| \log |G_Q|) = O((|\mathcal{V}| + 2^{\kappa+1}m + 2) \log(|\mathcal{V}| + 2^{\kappa+1}m + 2))$. By Lemma 2, we obtain that $|\mathcal{V}| = O(\lambda\gamma\zeta)$ and the overall query time (including G_Q construction time) is $O(\frac{m\lambda\gamma}{\zeta} + (\lambda\gamma\zeta + 2^{\kappa+1}m + 2) \log(\lambda\gamma\zeta + 2^{\kappa+1}m + 2))$. Since m and κ are small constants, we omit the terms of m and κ in the big O notation for simplicity and we obtain that the inter-box query time is $O(\lambda\gamma\zeta \log(\lambda\gamma\zeta))$.

Finally, we obtain that the query time of *EAR-Oracle* is $\max\{O(\alpha mN \log(\alpha mN)), O(\lambda\gamma\zeta \log(\lambda\gamma\zeta))\}$. The value of α is related to the skewness of the terrain surface and is a small real value. Since inner-box queries has spatial locality and they are only a minority of all arbitrary point-to-arbitrary point queries (i.e., $\frac{1}{\zeta^2}$), the inter-box query time dominates the inner-box query time. Thus, we obtain the query time of *EAR-Oracle* is $O(\lambda\gamma\zeta \log(\lambda\gamma\zeta))$.

Table 2 shows the average query time of inner-box and inter-box queries of *EAR-Oracle* under the default setting. We randomly generate 1000 queries for both query types. From this table we could see that the query time of inter-box queries dominates the one of inner-box queries. In addition, the number of inner-box queries is much smaller than the number of inter-box queries. The ratio of inner-box queries is only about 5% with $\zeta = 4$ (i.e., the number of partitions is 16). This ration will be further smaller as the number of partitions increases. □

We summarize our theoretical results as follows.

Theorem 5. *The time complexity, space consumption, query time of EAR-Oracle are* $O(\lambda\gamma\zeta mN \log(mN)) + \frac{N \log(N)}{\epsilon^{2\beta}} + Nh \log(N) + \frac{Nh}{\epsilon^{2\beta}})$, $O(\frac{m\lambda\gamma N}{\zeta} + \frac{Nh}{\epsilon^{2\beta}})$, $O(\lambda\gamma\zeta \log(\lambda\gamma\zeta))$ *and the multiplicative error bound and the additive error bound of EAR-Oracle are* ϵ *and* 2δ*, respectively.*

Proof. By Theorem 1, Theorem 2, Theorem 3 and Theorem 4, we obtain the theorem. □

5 EXPERIMENT

This section presents the experiments of this work. Section 5.1 presents the setting of our experiment including the datasets, algorithms and measures, etc. Section 5.2 presents the experimental results.

5.1 Experimental Setting

We conduct our experiments on a Linux machine with 3.60GHz Intel Xeon Gold 5122 CPU and 256 GB memory. All algorithms were implemented in C++. Some geometric operators (i.e., surface mesh processing, plane rotation, and etc.) are implemented by using the Computational Geometry Algorithms Library (CGAL v5.3) [36]. The geodesic shortest distance (exact shortest distance on unweighted terrain surfaces) is calculated by the triangulated surface mesh shortest path package [22].

Datasets. Following the existing studies, we adopted several real datasets from studies [20, 39]. Besides, we also collected some more real datasets from United States Geological Survey (USGS) system [35]. These datasets are digital elevation models and are processed with meshlab [9]. They correspond to real terrain surfaces with different sizes and different regions. The statistics of each dataset could be found in Table 3. Note that these datasets only provide the information of vertices, edges and faces and do not contain the weight information of each face. For the experiment on the weighted terrain surface, we adopted the method in [37] to generate the weight for each face.

Table 2: Inner/Inter-Box Average Query Time Comparison

Dataset	Inner Box (ms)	Inter Box (ms)
GunnisonForest (GF)	11.88	286.35
LaramieMountain (LM)	10.98	245.33
BearHead (BH)	23.18	219.32
EaglePeak (EP)	29.21	326.12

Table 3: Dataset Statistics

Dataset	No. of Faces	Region Covered	Reference
HorseMountain (HM)	1488	15 km^2	[35]
BigMountain (BM)	2772	29 km^2	[35]
HeadLightMountain (HL)	4771	49 km^2	[35]
RobinsonMountain (RM)	7,200	71 km^2	[35]
GunnisonForest (GF)	199,998	10,038 km^2	[35]
LaramieMountain (LM)	199,996	12,400 km^2	[35]
BearHead (BH)	292,914	140 km^2	[20, 39]
EaglePeak (EP)	325,713	150 km^2	[20, 39]

Algorithms. In our experiment, we consider both on-the-fly algorithms and index-based algorithms. For the on-the-fly algorithms, we consider *Fixed Scheme* [24], *Unfixed Scheme* [4] and *K-Algo* [20] which are the state-of-the-art approximate on-the-fly algorithms with an accuracy guarantee. For the index-based algorithms, we consider *SE-oracle* [39] which is the state-of-the-art index-based algorithm and it dominates any other index-based algorithm in terms of building time, space consumption, query time and accuracy according to [39].

Factors and Measures. We considered four factors in this paper, namely the data size (the number of faces of the terrain surface), ϵ (the error parameter given by the user), ζ (the number of boxes on the terrain surface) and m (the number of Steiner points placed on each bisector). We set $\epsilon = 0.2$ and $m = 5$ as the default values for ϵ and m. The default value of ζ for each dataset is determined as follows. We set ζ to be the integer which makes the size of highway node set $|\mathcal{V}|$ closest to 20% of N. Note that $|\mathcal{V}|$ grows up when ζ increases since the number of boundary line increases. Four measures are considered in the paper, namely *building time* (i.e., the time of constructing the indexing structure), *space consumption* (i.e., the memory consumption of the indexing structure), *query time* (the running time of answering a distance query) and the *relative error* (the empirical error of the distance returned by the algorithm). Note that each query time was reported as the average among 100 queries tested.

Query Generation. In this paper, we studied the arbitrary point-to-arbitrary point query. For each distance query, we randomly selected two arbitrary points, one of which as the source and the other as the destination. We first randomly generate two points $p_1(x_1, y_1, 0)$, $p_2(x_2, y_2, 0)$ bounded by the projection of the terrain surface on the 2D x-y plane. Then, we find the corresponding surface point s and t with the same x and y coordinate values with p_1 and p_2, respectively.

5.2 Experimental Results

In this section, we report our experiment results. Section 5.2.1 and Section 5.2.2 present the results on the unweighted and weighted terrain surfaces, respectively. Finally, Section 5.2.3 presents the benefit of using *EAR-Oracle*.

5.2.1 Results on Unweighted Terrain. Figure 9 shows the experimental results on the unweighted terrain datasets under the default parameter setting. As the figure shows, *SE-Oracle* could not scale up to large datasets and we do not show its results when it ran out of memory. We observed the following results: (1) *EAR-Oracle* outperforms *SE-Oracle* by more than 1 order of magnitudes in terms of building time and space consumption. *EAR-Oracle* could also scale up to large datasets but *SE-Oracle* could not handle. (2) *EAR-Oracle* has the smallest query time for each tested dataset and significantly outperforms all on-the-fly algorithms. (3) Each tested algorithm has a very small relative error which is much smaller than the default theoretical error bound. These results match the theoretical analysis in Section 4.6. *EAR-Oracle* only indexes the highway nodes rather than Steiner points and thus, its building time and memory usage is much smaller than *SE-Oracle*. The indexing structure of *EAR-Oracle* is lightweight and thus, its query time is much smaller than on-the-fly algorithms which perform the distance query on a large Steiner graph containing all Steiner points.

Scalability Test. We tested the selected algorithms with five different terrain datasets with various number of faces (i.e., 0.2M, 0.4M, 0.6M, 0.8M, 1.0M) under the default setting. They correspond to 5 sub-regions of a high-resolution EaglePeak (EP) dataset from existing study [39] (which has about 2.8 millions of faces). Figure 10 shows the results. Note that we do not show the result of an algorithm if its memory usage exceeds our memory budget (i.e., *SE-Oracle* and *Unfixed Scheme* for large datasets). We have the following observations from the experiments: (1) *SE-Oracle* ran out of memory space for a small dataset with only 0.2M faces while *EAR-Oracle* could scale up to a large dataset with 1 million faces. The building time, memory consumption and query time of *EAR-Oracle* increase when the number of faces increases. (2) The query time of *EAR-Oracle* is smaller than all on-the-fly algorithms by more than 1 order of magnitudes. Since the number of faces increases, the query times of *EAR-Oracle* and on-the-fly algorithms increase. (3) The relative error of each tested algorithm is very small compared with the default theoretical error bound. When the number of faces increases, more Steiner points will be placed to the terrain surface. Thus, *EAR-Oracle* needs more building time and memory space to finish the indexing structure construction. In addition, the sizes of query graph and Steiner graph increase and thus, more time is needed for distance query processing. These results match the theoretical analysis in Section 4.6.

Effect of ϵ. We tested the selected algorithms with five different values of ϵ from $\{0.05, 0.1, 0.15, 0.2, 0.25\}$ on the BigMountain dataset. The results are shown in Figure 11. According to these results, we observed that: (1) The building time and space consumption of *EAR-Oracle* are smaller than those of *SE-Oracle* by more than 1 order of magnitudes, respectively. (2) Since the value of ϵ increases, the building time, space consumption of *SE-Oracle* decrease. For *EAR-Oracle*, the two measures slightly decrease when ϵ increases. (3) *EAR-Oracle* has the smallest query time among all tested algorithms. The query times of *Unfixed Scheme* and *EAR-Oracle* decrease when ϵ increases. (4) The relative error of each tested algorithm is smaller than the corresponding parameter setting of ϵ (i.e., less than the 0.05, the minimum value of tested ϵ). The value of ϵ influences the number of placed Steiner points (for *Unfixed Scheme* and *K-Algo*), the size and building time of WSDPD (for *SE-Oracle* and *EAR-Oracle*) and the size of the highway network (for *EAR-Oracle*). When ϵ increases, the number of placed Steiner points, the size and processing time of WSDPD and the size of highway network $|\mathcal{G}|$ will decrease. These results match the theoretical analysis in Section 4.6.

Effect of ζ. We tested selected algorithms with different values of ζ from $\{4, 8, 16, 32, 64\}$ (i.e., the correspond quadtree depths are $\{2, 3, 4, 5, 6\}$) on the GunnisonForest (simplified) dataset. *SE-Oracle* ran out of memory and thus, we do not show its result. Figure 12 shows all results obtained. We have the following observations: (1) When ζ increases, the building time of *EAR-Oracle* increases but the space consumption decreases first and then slightly increases. (2) The query time of *EAR-Oracle* increases when ζ increases. The query time of other tested algorithms remains since ζ only affects *EAR-Oracle*. (3) All tested algorithms have a very small relative error compared with the default value of ϵ. The building time and space consumption of *EAR-Oracle* is influenced by the highway network and WSDPD. When ζ increases, the number of highway nodes increases but the number of the Steiner points in a box decreases. Besides, more highway nodes lead to larger processing time and space consumption of WSDPD. Note that the memory consumption of *EAR-Oracle* is determined by the distance map and WSDPD. When ζ increases, the distance map will rapidly decreases but the size of WSDPD will increase. The results of building time and memory consumption are comprehensive of the above components. For query time, more highway nodes imply larger query graph size and longer query time. When $|\mathcal{V}|$ is close to $|V|$, the query time of *EAR-Oracle* becomes $O(\lambda\gamma|V|\log(\lambda\gamma|V|))$ which is comparable to the complexity of *Fixed Scheme*.

Effect of m. We tested the selected algorithms with different values of m from $\{3, 4, 5, 6, 7\}$ on the BigMountain dataset. The experiment results are shown in Figure 13. We observe the following: (1) The building time and space consumption of *EAR-Oracle* outperforms *SE-Oracle* by more than 1 order of magnitudes. Besides, the two measures increase when m grows up. (2) *EAR-Oracle* has the smallest query time among all tested algorithms. The query times of *Fixed Scheme* and *SE-Oracle* increase when m grows up while the query time of other tested algorithms remains. (3) All tested algorithms have a very small relative error which is much smaller than the default error bound. When m increases, more Steiner points are introduced, and thus the size and processing time of WSDPD (for *SE-Oracle* and *EAR-Oracle*), base graph (for *SE-Oracle* and *EAR-Oracle*) and distance map (for *EAR-Oracle*) will increase. For query time, the size of Steiner graph in *Fixed Scheme* and the number of pairs for enumeration in *SE-Oracle* will increase, and thus more time is required for their query processing. For *Unfixed Scheme* and K-Algo, their Steiner graph size remain unchanged thus their query time do not change. For *EAR-Oracle*, according to our theoretical analysis in Section 4.6.3, the query time is dominated by the highway network size (i.e.,

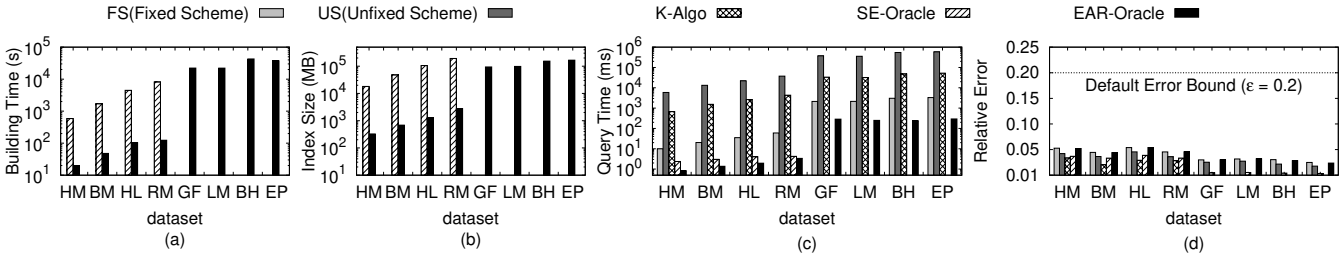


Figure 9: Experimental Results on Unweighted Terrain Datasets

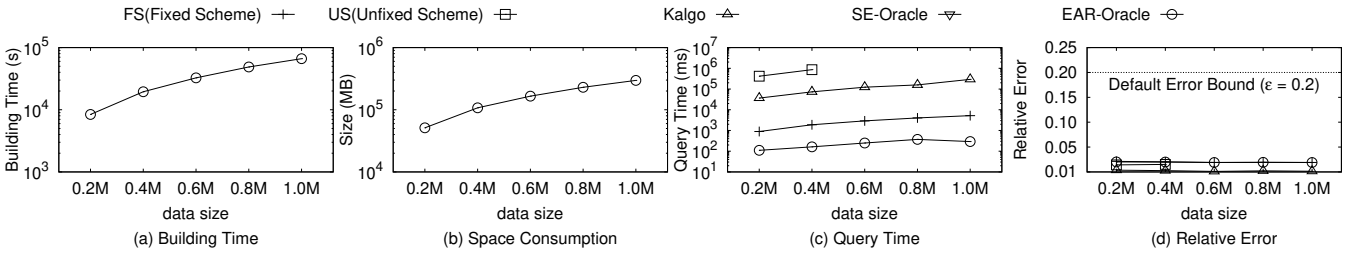


Figure 10: Scalability Test on EP Dataset

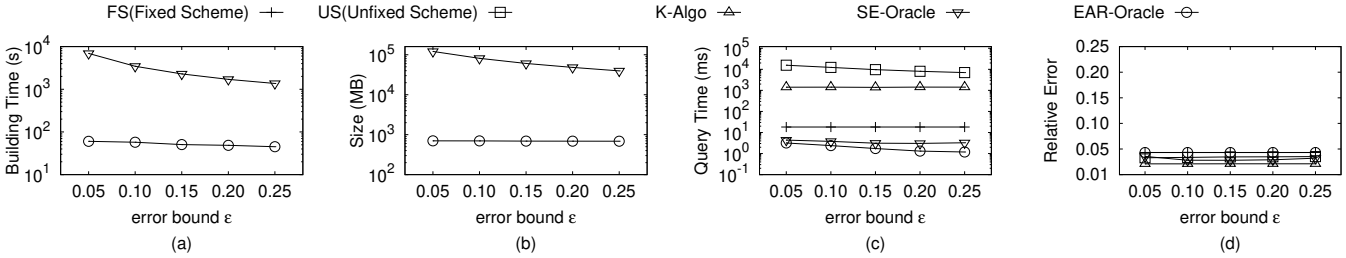


Figure 11: Effect of ϵ on BigMountain Dataset

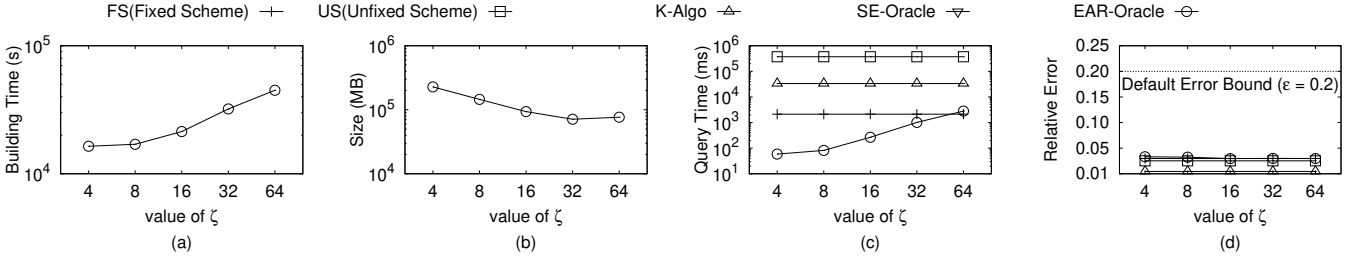


Figure 12: Effect of ζ on GunnisonForest (simplified) Dataset

inter-box queries) which does not change a lot when m changes, so the query time of *EAR-Oracle* remains. These results match our theoretical analysis since the query graph is lightweight and only contains Steiner points within κ hops away from s and t.

5.2.2 Results on Weighted Terrain. We also tested the selected algorithms on weighted terrain surface under the default setting. Since there are no algorithm could find the exact shortest distance on weighted terrain surfaces, we select the results from *Fixed Scheme* as the pivot for relative error calculation.

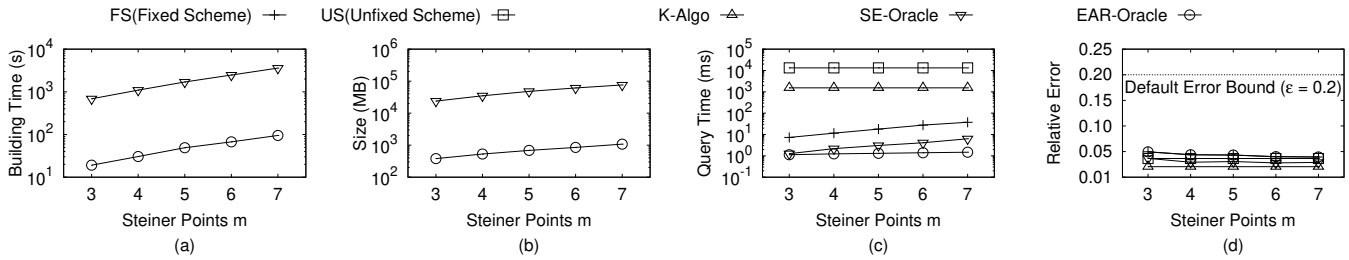


Figure 13: Effect of m on BigMountain Dataset

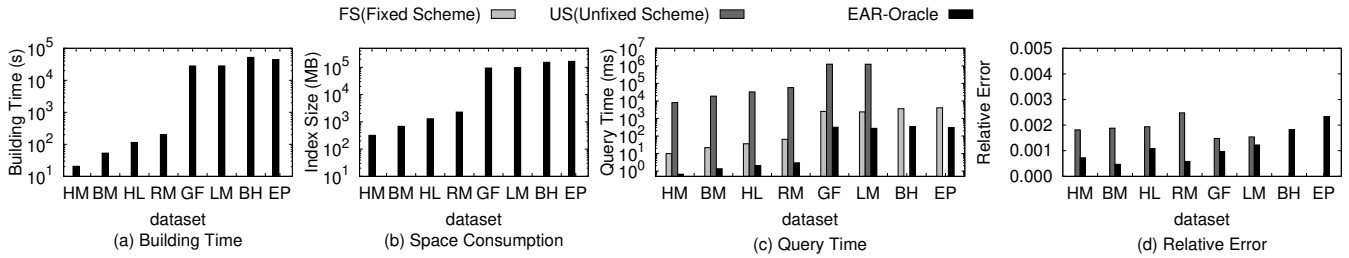


Figure 14: Experimental Results on Weighted Terrain Datasets

The results are shown in Figure 14. Please note that we do not show the results which exceed our memory budget (i.e., 256GB). We observed the following results: (1) *EAR-Oracle* and *Fixed Scheme* has the similar performance as on unweighted surfaces in building time, space consumption and query time. (2) *EAR-Oracle* has the smallest query time which is more than 1 order of magnitudes less than the on-the-fly algorithms. (3) *Unfixed Scheme* has longer query time compared with unweighted terrain surfaces. It also ran out of memory on BH and EP datasets. (4) The tested algorithms have very close result quality. For *EAR-Oracle* and *Fixed Scheme*, the only difference between the weighted terrain surfaces and unweighted terrain surfaces is that the face weights involve the edge weight calculation. However, for *Unfixed Scheme*, more Steiner points must be introduced according to its geometric based scheme. These results are consistent with our theoretical analysis.

5.2.3 Discussion on Benefits of Using EAR-Oracle. This section discusses the advantages of using EAR-Oracle over other algorithms. In this discussion, we keep feeding queries to each algorithm. For each on-the-fly algorithm, its total time cost is calculated as the total query processing time. For each index-based algorithm, its the total time cost is calculated as the sum of its building time and the total query processing time. For each index-based algorithm, we also considered the comparison with the fastest on-the-fly algorithm *Fixed Scheme* and reported its *warm-up query number* which is the minimum number of queries such that the index-based algorithm achieves smaller total time cost than Fixed Scheme. By our results on RobinsonMountain dataset, the warm-up query number of EAR-Oracle is 2,187 which is significantly smaller than that of SE-Oracle (which is 147,382). This implies that the cost of building EAR-Oracle could be amortized by a moderate number of queries and it is worthwhile to build EAR-Oracle. We also obtained similar observations on other datasets and they are omitted here for the sake of limited space.

6 CONCLUSION

In this paper, we proposed a highway network-based indexing structure for answering the arbitrary point-to-arbitrary point distance query on the terrain surface. It achieved state-of-the-art performances in terms of the building time, space consumption, query time and distance error. Remarkably, it could scale up to million-scale terrain surfaces where all existing indexing-based algorithms could not due to their prohibitively large building time and space consumption. It significantly outperforms all on-the-fly algorithms in terms of query time.

REFERENCES

[1] A. Al-Badarneh, H. Najadat, and A. Alraziqi. 2012. A classifier to detect tumor disease in MRI brain images. In *The international conference series on Advances in Social Network Analysis and Mining (ASONAM)*. 784–787.

[2] Lyudmil Aleksandrov, Hristo N Djidjev, Hua Guo, Anil Maheshwari, Doron Nussbaum, and Jörg-Rüdiger Sack. 2010. Algorithms for approximate shortest path queries on weighted polyhedral surfaces. In *Discrete & Computational Geometry*. 762–801.

[3] Lyudmil Aleksandrov, Hristo N Djidjev, Hua Guo, Anil Maheshwari, Doron Nussbaum, and Jörg-Rüdiger Sack. 2010. Algorithms for approximate shortest path queries on weighted polyhedral surfaces. In *Discrete & Computational Geometry*. 762–801.

[4] Lyudmil Aleksandrov, Anil Maheshwari, and J-R Sack. 2005. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of ACM*, 25–53.

[5] Paul B Callahan and S Rao Kosaraju. 1995. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of ACM*, 67–90.

[6] Marcel Campen, Martin Heistermann, and Leif Kobbelt. 2013. Practical Anisotropic Geodesy. *Comput. Graph. Forum*, 63–71.

[7] Nathaniel Chaney, Laura Torres-Rojas, and Jason Simon. 2021. Leveraging clustering and geostatistics to improve the modeling of sub-grid land-atmosphere interactions in Earth system models. In *EGU General Assembly Conference Abstracts*. EGU21–14039.

[8] Jindong Chen and Yijie Han. 1990. Shortest paths on a polyhedron. In *International Symposium on Computational Geometry* (*SoCG*). 360–369.

[9] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*, Vittorio Scarano, Rosario De Chiara, and Ugo Erra (Eds.). The Eurographics Association. https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136

[10] Keenan Crane, Marco Livesu, Enrico Puppo, and Yipeng Qin. 2020. A Survey of Algorithms for Geodesic Paths and Distances. *arXiv preprint arXiv:2007.10430* (2020).

[11] Ke Deng, Heng Tao Shen, Kai Xu, and Xuemin Lin. 2006. Surface k-NN query processing. In *IEEE International Conference on Data Engineering* (*ICDE*). 78.

[12] Ke Deng and Xiaofang Zhou. 2004. Expansion-based algorithms for finding single pair shortest path on surface. In *International Conference on Web and Wireless Geographical Information Systems* (*WWGIS*). 151–166.

[13] Ke Deng, Xiaofang Zhou, Heng Tao Shen, Qing Liu, Kai Xu, and Xuemin Lin. 2008. A multi-resolution surface distance model for k-NN query processing. *ACM International Journal on Very Large Data Bases* (*VLDBJ*), 1101–1119.

[14] Brett G Dickson and Paul Beier. 2007. Quantifying the influence of topographic position on cougar (Puma concolor) movement in southern California, USA. *Journal of Zoology*, 270–277.

[15] Hristo N Djidjev and Christian Sommer. 2011. Approximate distance queries for weighted polyhedral surfaces. In *The European Symposium on Algorithms* (*ESA*). 579–590.

[16] Raphael A. Finkel and Jon Louis Bentley. 1974. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica* 4 (1974), 1–9. https://doi.org/10.1007/BF00288933

[17] Wei Hu. 2008. *Co-location Pattern Discovery*. Springer US, 98–103.

[18] S. A. Huettel, A. W. Song, and G. McCarthy. 2004. Functional magnetic resonance imaging. In *Sinauer Associates*.

[19] Takashi Kanai and Hiromasa Suzuki. 2000. Approximate shortest path on a polyhedral surface based on selective refinement of the discrete graph and its applications. In *Proceedings Geometric Modeling and Processing 2000. Theory and Applications* (*GMPTA*). 241–250.

[20] Manohar Kaul, Raymond Chi-Wing Wong, and Christian S Jensen. 2015. New lower and upper bounds for shortest distance queries on terrains. *ACM International Conference on Very Large Data Bases* (*VLDB*), 168–179.

[21] Manohar Kaul, Raymond Chi-Wing Wong, Bin Yang, and Christian S Jensen. 2013. Finding shortest paths on terrains by killing two birds with one stone. In *ACM International Conference on Very Large Data Bases* (*VLDB*). 73–84.

[22] Stephen Kiazyk, Sébastien Loriot, and Éric Colin de Verdière. 2021. Triangulated Surface Mesh Shortest Paths. In *CGAL User and Reference Manual*. CGAL Editorial Board. https://doc.cgal.org/5.3/Manual/packages.html#PkgSurfaceMeshShortestPath

[23] M. Kortgen, G. J. Park, M. Novotni, and R. Klei. 2003. 3D shape matching with 3D shape contexts. In *Central European Seminar on Computer Graphics* (*CESCG*). 5–17.

[24] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica*, 527–562.

[25] Lik-Hang Lee, Zijun Lin, Rui Hu, Zhengya Gong, Abhishek Kumar, Tangyao Li, Sijia Li, and Pan Hui. 2021. When Creators Meet the Metaverse: A Survey on Computational Arts. *ArXiv* abs/2111.13486 (2021). https://arxiv.org/abs/2111.13486

[26] Lik-Hang Lee, Tristan Braud, Pengyuan Zhou, Lin Wang, Dianlei Xu, Zijun Lin, Abhishek Kumar, Carlos Bermejo, and Pan Hui. 2021. All One Needs to Know about Metaverse: A Complete Survey on Technological Singularity, Virtual Ecosystem, and Research Agenda. *ArXiv* abs/2110.05352 (2021).

[27] Lian Liu and Raymond Chi-Wing Wong. 2011. Finding shortest path on land surface. In *ACM Conference on Management of Data* (*SIGMOD*). 433–444.

[28] Anders Mårell, John P Ball, and Annika Hofgaard. 2002. Foraging and movement paths of female reindeer: insights from fractal analysis, correlated random walks, and Lévy flights. *Canadian Journal of Zoology*, 854–865.

[29] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. *SIAM J. Comput.*, 647–668.

[30] Zaw Lin Oo and Mya Sandar Kyin. 2020. Discovery and comparative study on spatial co-location and association rule mining of spatial data mining. *International Journal for Advance Research and Development* (2020), 16–19.

[31] L Tiina Sarjakoski, Pyry Kettunen, Hanna-Marika Flink, Mari Laakso, Mikko Rönneberg, and Tapani Sarjakoski. 2012. Analysis of verbal route descriptions and landmarks for hiking. *Personal and Ubiquitous Computing*, 1001–1011.

[32] Cyrus Shahabi, Lu-An Tang, and Songhua Xing. 2008. Indexing land surface for efficient knn query. In *ACM International Conference on Very Large Data Bases* (*VLDB*). 1020–1031.

[33] J. Shotton, J. Winn, C. Rother, and A. Criminisi. 2006. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European Conference on Computer Vision* (*ECCV*). 1–15.

[34] Michiel H. M. Smid. 2018. The Well-Separated Pair Decomposition and Its Applications. In *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 2: Contemporary and Emerging Applications, Chapter 4*. Chapman and Hall/CRC.

[35] United States Geological Survey. 2021. 3D Elevation Program 1 arc-second Digital Elevation Model. In *United States Geological Survey*. Distributed by OpenTopography. https://doi.org/10.5069/G98K778D

[36] The CGAL Project. 2021. *CGAL User and Reference Manual*. CGAL Editorial Board. https://doc.cgal.org/5.3/Manual/packages.html

[37] Nguyet Tran, Michael J Dinneen, and Simone Linz. 2020. Close Weighted Shortest Paths on 3D Terrain Surfaces. In *The ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems 2021* (*SIGSPATIAL*). 597–607.

[38] Chuyuan Wang, Yubin Li, Soe W Myint, Qunshan Zhao, and Elizabeth A Wentz. 2019. Impacts of spatial clustering of urban land cover on land surface temperature across Köppen climate zones in the contiguous United States. *Landscape and urban planning* 192 (2019), 103668.

[39] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, and David M. Mount. 2017. Distance Oracle on Terrain Surface. In *ACM Conference on Management of Data* (*SIGMOD*). 1211–1226.

[40] Shi-Qing Xin and Guo-Jin Wang. 2009. Improving Chen and Han's Algorithm on the Discrete Geodesic Problem. *ACM Transactions on Graphics* (*TOG*), 104:1–104:8.

[41] S. Xing, C. Shahabi, and B. Pan. 2009. Continuous monitoring of nearest neighbors on land surface. In *ACM International Conference on Very Large Data Bases* (*VLDB*). 1114–1125.

[42] D. Yan, Z. Zhao, and W. Ng. 2012. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *The Conference on Information and Knowledge Management* (*CIKM*). 942–951.