On Efficient Trip Planning on Terrain Surfaces

Paper ID: 1139

Abstract

With the advancement of geo-positioning technologies, the terrain surface has become more and more popular and has drawn a lot of attention from the academia and the industry. In this paper, we propose to study a fundamental problem, namely *Trip Planning on Terrain Surfaces (TPTS)*. Given a source point s, a destination point t, and a set P of point-of-interests on the terrain surface, the trip planning problem aims to find a s-t path passing through all points in P on the terrain surface with the minimum length. The trip planning finds a plethora of applications in the map service, military vehicle path planning, the scientific study, etc.

In this paper, we prove that TPTS is an NP-hard problem. Due to its hardness, we develop an approximation algorithm for the TPTS problem. Let N and k denote the number of vertices on the terrain surface and the number of points in P, respectively. The running time and space overhead of our algorithm are respectively $O(\frac{k \cdot N \log^2 N}{\epsilon^{2\beta}})$ and $O(\frac{k}{\epsilon^{2\beta}})$, where ϵ is a real-valued user-specified parameter in the range of $[0, 1]$ and β is a small constant in the range of $[1.5, 2]$. The approximation ratio of our algorithm is $2 \cdot (1 + \epsilon)$. Our theoretical analysis and empirical study both demonstrate that our algorithm significantly outperforms all baselines in terms of running time and space overhead with nearly the same or better approximation ratio.

1 Introduction

Digital terrain surfaces have emerged as an important data object and have attracted much attention from both the academia and the industry [4, 6, 8, 10–12, 18, 25, 28–31, 33, 34] due to the proliferation of the geo-positioning and computer technologies. They are widely used in applications such as geographical information system, simulation and 3D games, etc. Terrain surface data consists of *faces*, *edges* and *vertices*. Figure 1 demonstrates a instance of the terrain surface. As the figure shows, each face (i.e., a triangle) has three adjacent line segments called *edges*. Each edge is connected with two *vertices*. The terrain surface shown in Figure 1 has 22 faces, 35 edges, and 14 vertices.

Given a terrain surface T, the *geodesic distance* between two given points on T is the length of the *shortest* path from one point to the other traveling along the surface. For example, in Figure 1, s and t are two points on the terrain surface. The shortest path from point s to point t, denoted by $\Pi_g(s, t)$, is shown as a sequence of dotted line segments, whose length is called the geodesic distance from s to t. The Euclidean distance from point s to point t is the length of the straight line segment connecting these two points. Due to the undulating nature of terrain surfaces, the geodesic distance often significantly exceeds the Euclidean distance (their relative difference is up to 300% on the real terrain datasets by the result of [4]).

In this paper, we propose to study a fundamental and important problem on terrain surfaces called *Trip Planning on Terrain Surfaces (TPTS)*. TPTS aims to find the shortest path, called s-t path, from a source point s to a destination point t that passes through a given set P of points-of-interest on the terrain surface. Consider the example in Figure 1. In addition to the two points s and t, there is a set $P = \{p_1, p_2\}$ of points-of-interest on the terrain surface. The shortest path from s to t that passes through all points in P (i.e., p_1 and p_2) is denoted by $\Pi_g(s, t|P)$. TPTS is a fundamental problem on terrain surfaces and has wide applications in Geographic Information Systems (GIS), military vehicle routing, etc. Here are some examples.

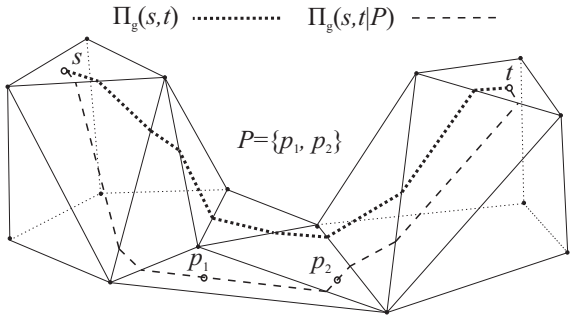

Figure 1: An Example of Trip Planning on Terrain Surface

(1) In map service, hikers would like to find a proper route on the terrain surface to design a proper hiking trail [24]. In this scenario, they provide a starting point s and an ending point t for hiking and places for sightseeing (e.g., a waterfall, a peak, etc.), which must be covered in the route. To achieve this, the best route is the path from s to t on the terrain surface, passing through all the places with the minimum length.

(2) In Geographical Information System (GIS), many vehicles (e.g., Google Map camera cars and military vehicles) require this route planning on the terrain surface [15, 27]. In this case, each vehicle has a starting point s and a destination point t, and data on several pre-defined locations must be collected. The best route is the path on the terrain surface that passes through all the locations with the smallest length to save energy and time.

(3) In military tactical analysis, route planning on the terrain surface is critical for guiding the movement of the military vehicles and equipment [13]. The military vehicle for the delivery or rescue is required to deliver resources, such as food, equipment, medicines, etc., to several locations during their trip from one base to another. A shortest path that passes through all the locations will help them provide timely services.

Although the trip planning problem on terrain surface is significant, to the best of our knowledge, our work is the first one to investigate it systematically. We prove that this problem is NP-hard and highly intractable. In light of this, we propose an efficient approximation algorithm for trip planning on terrain surfaces. In a nutshell, our algorithm first constructs a connected, lightweight, and sparse graph $\mathcal{G}$ called *spanning graph* for all points in $P \cup \{s, t\}$. Denote by N and k the number of vertices on the terrain surface and the number of points in P (i.e., $k = |P|$), respectively. The vertices of the spanning graph are comprised of all points in $P \cup \{s, t\}$, and the spanning graph only has $O(k)$ edges. Then, based upon this spanning graph $\mathcal{G}$, we propose a minimum spanning tree-based al-

gorithm for finding a s-t path $\pi_{\mathcal{G}}(s,t|P)$ on $\mathcal{G}$ that passes through all points in P. Finally, we return the corresponding path of $\pi_{\mathcal{G}}(s,t|P)$ on the original terrain surface. Our theoretical analysis shows that our proposed algorithm has an $O(\frac{k \cdot N \log^2 N}{\epsilon^{2\beta}})$ time complexity (i.e., linear to both k and $N \log^2 N$), an $O(\frac{k}{\epsilon^{2\beta}})$ space complexity (i.e., linear to k) and an approximation ratio $2 \cdot (1+\epsilon)$, where ϵ is a user-specified parameter for the trade-off between efficiency and accuracy and β is a small constant in the range of $[1.5, 2]$.

Contributions. Our contributions are summarized as follows.

- We propose a novel problem on terrain surfaces, namely *Trip Planning on Terrain Surfaces (TPTS)*, which finds a plethora of applications in GIS, map services, military tactical analysis, etc. We also prove the NP-hardness and intractability of the problem.
- We propose an efficient approximation algorithm for TPTS. We theoretically show that the time complexity (resp. space complexity) of our algorithm is $O(\frac{k \cdot N \log N}{\epsilon^{2\beta}})$ (resp. $O(\frac{k}{\epsilon^{2\beta}})$), which is linear to both k and $N \log^2 N$ (resp. linear to k). In particular, the approximation ratio is $2 \cdot (1+\epsilon)$. It beats all competitors in terms of running time, space overhead, and accuracy in theory.
- We conducted extensive experiments on real datasets. The results show that our proposed algorithm significantly outperforms all competitors by orders of magnitudes in terms of running time and space overhead with the same or better accuracy.

The remainder of the paper is organized as follows. Section 2 formally defines the problem of trip planning on terrain surfaces and discusses its NP-hardness. Section 3 reviews the existing studies and presents their adaptations to our problem. Section 4 presents our proposed algorithm. Section 5 demonstrates the experiments. Finally, Section 6 summarizes this work with possible future works.

2 Problem Definition

Consider a terrain surface T. Let V, E and F denote the sets of all vertices, edges and faces on the surface of T, respectively. For example, in Figure 1, each solid point is a vertex, and each solid line segment is an edge. The complexity of the terrain surface T is represented by the total number, denoted by N, of vertices (i.e., $N = |V|$). Each vertex $v \in V$ is a 3D point and we denote its three coordinate values by x_v, y_v, and z_v. We refer to an arbitrary point on the terrain surface simply as 'point', which may or may not be a vertex of the terrain surface. For instance, in Figure 1, there are two points s and t on the terrain surface but they are not vertices. Frequently used notations are summarized in Table 1.

Consider two points s and t on the terrain surface T. A path, denoted by $\pi_g(s,t)$, from s to t on the terrain surface consists of a sequence $\mathcal{S}$ of line segments. Each line segment l in $\mathcal{S}$ lies on a face of the terrain surface and each pair of adjacent line segments share one end-point. The length of a given line segment l is denoted by $|l|$ and the length of the path $\pi_g(s,t)$ is the sum of the lengths of the line segments in $\mathcal{S}$ (i.e., $\sum_{l \in \mathcal{S}} |l|$). Based on the concepts above, we now formally define the *geodesic shortest path*.

Definition 1 (Geodesic Shortest Path). *The* geodesic shortest path *between point s and point t, denoted by $\Pi_g(s,t)$, is the path on T with the minimum length from s to t.*

Notation	Meaning
T, V, E, F	Terrain, Vertices, Edges and Faces.
N	The number of vertices on T.
s, t	Two arbitrary points on the terrain.
p	A point-of-interest (POI) on T (which may or may not be located at a vertex of T).
P	A set of points-of-interest on T.
k	The number of points contained in P (i.e., $k = \|P\|$).
$\pi_g(s,t)$	A path from s to t on the terrain surface.
$\Pi_g(s,t)$	The shortest path from s to t on the terrain surface.
$\Pi_g(s,t\|p)$	The geodesic shortest path from s to t passing through a point p.
$\Pi_g(s,t\|P)$	The geodesic shortest path from s to t passing through each point in P.
$d_g(s,t)$	The geodesic distance from s to t.
$D(p,r)$	A disk on the terrain surface centered at the point p with the radius r.
$\mathcal{G}$	The spanning graph of $P \cup \{s,t\}$ whose vertices are comprised of $P \cup \{s,t\}$.
$d_{\mathcal{G}}(u,v)$	The distance between the vertex u and the vertex v on the spanning graph $\mathcal{G}$.

Table 1: Concepts & Notations

The length of $\Pi_g(s,t)$, denoted by $d_g(s,t)$, is called the *geodesic distance* between s and t. In Figure 1, the geodesic shortest path between s and t is shown in dotted line segments, whose length is equal to the sum of the lengths of all line segments on $\Pi_g(s,t)$. Building on the concept of the geodesic shortest path, we extend it to trips that involve multiple points-of-interest.

Definition 2 (Geodesic Shortest Trip Path). *Given a terrain surface T, two points s and t on the surface of T, and a set P of points on the surface of T, the* geodesic shortest trip path*, denoted by $\Pi_g(s,t|P)$, is the geodesic shortest path from s to t on the terrain surface that passes through all points in P.*

Figure 1 gives an example of $\Pi_g(s,t|P)$, where there are two points s and t and a set $P(=\{p_1,p_2\})$ on the terrain surface. Now we are ready to formally define the problem.

Problem 1 (Trip Planning on Terrain Surface (TPTS)). *Given a terrain surface T, two points s and t on T, and a set P of points-of-interest (POIs) on T, we find the geodesic shortest trip path $\Pi_g(s,t|P)$.*

Intractability. Unfortunately, TPTS is NP-hard as shown below.

Theorem 1 (Hardness). *TPTS is NP-hard.*

Proof. We prove that TPTS is NP-hard by transforming the renown *Euclidean Travelling Salesman Problem (ETSP)* [21] (which has been proved to be NP-hard [20]) into TPTS.

We first present the decision version of the TPTS problem. Given a terrain surface T, two points s and t on the surface of T, a set P of points-of-interest on the surface of T, and a non-negative real number r, the decision version of the TPTS problem is to determine if there is a path $\Pi(s,t|P)$ on the surface of T such that (1) the path starts from s and ends with t, (2) the path passes through all points in P, and (3) the length of the path is at most r.

The Euclidean Travelling Salesman Problem (ETSP) is defined as follows. Given a set Q of points on the 2D plane, it finds a path passing through all points in Q with the minimum length. Note that

in this problem, the distance between any points is their Euclidean distance. Then, we present the decision version of ETSP problem. Given a set Q of points on the 2D plane and a non-negative real number r', the problem is to determine if there is a path on the 2D plane passing through all points in Q such that the length of the path is at most r'. Given this instance of ETSP problem, we can construct a TPTS problem instance as follows. We first create a terrain surface T' which is simply coincides with the 2D plane. Then, we randomly selects a point p' from Q and we denote $Q \setminus \{p'\}$ by P'. After that, the transformed TPTS problem instance is as follows. Given the terrain surface T', two points $s'(= p')$ and $t'(= p')$ on the surface of T', a set P' of points-of-interest on the surface of T', and a non-negative real number r', the TPTS problem instance is to determine if there is a path $\Pi(s', t'|P')$ on the surface of T' such that (1) the path starts from s' and ends with t', (2) the path passes through all points in P', and (3) the length of the path is at most r'. It could be verified that the ETSP problem instance and the transformed TPTS problem instance are equivalent. □

3 Related Work and Baselines

We review existing studies on digital terrain surfaces in Section 3.1 and discuss their adaptations to our problem in Section 3.2. Then, we compare these adaptations with our algorithm in Section 3.3.

3.1 Related Work

Geodesic Shortest Distance and Path Queries. The index-free exact algorithms *MMP* [19], *CH* [3], *ICH* [32], *VS* [28], *DIO* [31], and *KS* [10] compute the exact geodesic distance and path on the fly, where no pre-computed data structures are required. Their time complexities are $O(N^2 \log N)$, $O(N^2 \log N)$, $O(N^2)$, $O(N^2 \log N)$, $O(N^2 \log N)$, and $O(N \log^2 N)$, respectively, where N is the number of vertices on the terrain surface. Later on, the index-free approximation algorithms were proposed to expedite the query processing. All existing on-the-fly approximation algorithms [1, 11, 17] follow the same framework. Intuitively, they introduce some auxiliary points, namely *Steiner points*, and some auxiliary edges, namely *Steiner edges*, on the terrain surface. Based on these points and edges, they construct a *Steiner graph* $\mathbb{G}$. In the query phase, an edge between the source point s and each Steiner point on the face that s lies on is inserted into graph $\mathbb{G}$. Similarly for the destination point t. After the insertion, they perform Dijkstra's algorithm from s to t. These on-the-fly approximation algorithms have the same time complexities $O((N + N') \log(N + N'))$, where N' is the number of Steiner points introduced. They differ in the method for introducing the Steiner points and Steiner edges.

Afterward, index-based algorithms [7–9, 29] for the geodesic shortest path computation were proposed to further accelerate the query processing. The first attempt was a Single-Source All-Destination algorithm [9], where the source point must be known apriori and kept fixed in the query phase. SP-Oracle [7] builds an indexing structure for the shortest path query processing on the Steiner graph. Inspired by [2, 22, 23], SE-Oracle and EAR-Oracle [8, 29] index the geodesic distances and paths by using the techniques called *Well-Separated Pair Decomposition* and *Highway Network*, respectively. Nevertheless, the index-based algorithms have the overhead of preprocessing time and the additional storage consumption for the bulky indexing structure.

(Reverse) Nearest Neighbor Queries on Terrain Surfaces. Also related are (reverse) nearest neighbor quires [5, 6, 25, 33, 34] on the terrain surface. Specifically, [5, 6] studied the k nearest neighbor (kNN) query problem and proposed a multi-resolution representation of the terrain surface for pruning irrelevant regions to boost the efficiency. [25] further studied kNN queries and proposed a Voronoi Diagram method for kNN queries on a set of static points. [30] developed a kNN algorithm based on *SE-Oracle* [29]. [33] studied the problem of dynamic monitoring the kNN for moving objects. [34] studied reverse k nearest neighbors queries.

Other Related Work. [18] studied the problem of finding the shortest geodesic path satisfying a slope constraint and [12] provided a method of lower/upper bound estimations for the geodesic distance between s and t but their bounds are sensitive to the geometric property of the terrain (i.e., they have no quality guarantees).

Algorithm 1: Nearest Neighbor-based Algorithm

Input: A set P of POIs, a source point s, a destination point t, a parameter δ, and a terrain surface T
Output: A path π from s to t passing through all points in P

1: Initialize a set Q of points to be P;
2: Initialize a point o to be s;
3: Initialize a path π to be $\emptyset$;
4: **while** Q is not $\emptyset$ **do**
5: Find the point nn in Q nearest to o by the algorithm [6];
6: Find the shortest geodesic path π' from o to nn through the fastest shortest geodesic path computation algorithm [31];
7: Append π' to the path π;
8: $o \leftarrow nn$;
9: Remove nn from Q;
10: **end while**
11: Find the shortest geodesic path π' from o to t through the fastest shortest geodesic path computation algorithm [31];
12: Append π' to the path π;
13: **return** π;

3.2 Baseline Algorithms

As far as we are concerned, existing studies (as survey in Section 3.1) focus on shortest distance/path queries and (reverse) nearest neighbor queries on terrain surfaces. Trip planning on terrain surfaces (TPTS) has not been studied before. Although the existing works cannot directly be applied to TPTS, we propose several categories of adaptations to solve TPTS. Denote by N, P, and k the total number of vertices on the terrain surface, the set of points-of-interest (POIs), and the number of points in P (i.e., $k = |P|$), respectively. Table 2 summarizes the time complexity, space overhead, and approximation ratio of each adaptation. The detailed analysis of these adaptations can be found in Appendix A.

Nearest Neighbor-based Adaptations (NNA). The pseudocode of NNA is shown in Algorithm 1. It starts from s and iteratively travels to the nearest unvisited POI nn via the shortest geodesic path from the currently visited point to nn (Lines 5-8). This process is repeated until all POIs are visited (Line 4) and the destination

Algorithm	Running Time	Space Overhead	Appro. Ratio
NNA	$O(k \cdot N^2 \log N)$	$O(k+N)$	k
KS-Adapt [10]	$O(k^2 N \log^2(N))$	$O(k^2+N)$	$\frac{3}{2}$
DIO-Adapt [31]	$O(k^2 N^2 \log(N))$	$O(k^2+N)$	$\frac{3}{2}$
FS-Adapt [17]	$O(k^2 N \log(N))$	$O(k^2+N)$	$\frac{1+\delta}{1-\delta} \cdot \frac{3}{2}$
US-Adapt [1]	$O(k^2 \cdot \frac{N}{\epsilon} \log(\frac{N}{\epsilon}) \log \frac{1}{\epsilon})$	$O(k^2+N)$	$\frac{1+\epsilon}{1-\epsilon} \cdot \frac{3}{2}$
K-Algo-Adapt [11]	$O(k^2 \cdot \frac{N}{\epsilon} \log(\frac{N}{\epsilon}))$	$O(k^2+N)$	$\frac{1+\epsilon}{1-\epsilon} \cdot \frac{3}{2}$
SP-Oracle-Adapt [7]	$O(\frac{N \log \frac{1}{\epsilon}}{\epsilon^{2.5}} \log^3(\frac{N \log \frac{1}{\epsilon}}{\epsilon^{1.5}}) \log^2 \frac{1}{\epsilon} + k^2(\frac{1}{\epsilon} \log \frac{1}{\epsilon} + \log\log(\frac{N}{\sqrt{\epsilon}})))$	$O(\frac{N \log \frac{1}{\epsilon}}{\epsilon^2} \log^2 \frac{N \log \frac{1}{\epsilon}}{\epsilon^{1.5}} \log^2 \frac{1}{\epsilon} + k^2)$	$\frac{1+\epsilon}{1-\epsilon} \cdot \frac{3}{2}$
SE-Oracle-Adapt [29]	$O(\frac{N \log^2 N}{\epsilon^{2\beta}} + \frac{Nh \log \frac{1}{\epsilon}}{\sqrt{\epsilon}} \cdot \log \frac{N \log \frac{1}{\epsilon}}{\sqrt{\epsilon}} + \frac{Nh \log \frac{1}{\epsilon}}{\sqrt{\epsilon} \cdot \epsilon^{2\beta}} + k^2 h)$	$O(\frac{Nh \log \frac{1}{\epsilon}}{\sqrt{\epsilon} \cdot \epsilon^{2\beta}} + k^2)$	$\frac{1+\epsilon}{1-\epsilon} \cdot \frac{3}{2}$
EAR-Oracle-Adapt [8]	$O(\zeta N \log(N)) + \frac{N \log(N)}{\epsilon^{2\beta}} + Nh \log(N) + \frac{Nh}{\epsilon^{2\beta}} + k^2 \zeta \log(\zeta)$	$O(\frac{N}{\zeta} + \frac{Nh}{\epsilon^{2\beta}} + k^2)$	$\frac{1+\epsilon}{1-\epsilon} \cdot \frac{3}{2}$
Ours	$O(\frac{k}{\epsilon^{2\beta}} N \log^2 N)$	$O(\frac{k}{\epsilon^{2\beta}} + N)$	$2 \cdot (1+\epsilon)$

Remark: δ is the relative error of the geodesic distance estimated by *FS* which contains many terrain-related factors such as the length of the longest edge and the minimum inner angle of each face on the terrain surface (see [17] for details). β is a small constant in the range of $[1.5, 2]$ and h is around 10-30 by the results of [8, 29, 30]. ζ is a algorithm parameter of *EAR-Oracle* which is normally set to be 16 or 256.

Table 2: Complexities of Our Algorithm and Adaptations of Existing Algorithms

Algorithm 2: Geodesic Distance-based Algorithm

Input: A set P of POIs, a source point s, a destination point t, a parameter δ, and a terrain surface T
Output: A path π from s to t passing through all points in P
1: Initialize the path π to be $\emptyset$;
2: Call an shortest geodesic distance computation algorithm to find the geodesic distance for each pair of points in $P \cup \{s, t\}$;
3: Construct a complete graph G_Q for $P \cup \{s, t\}$, where the weight of each edge (u, v) is $d_g(u, v)$;
4: Find the path, denoted by $\Pi_E(s, t|P)$, from s to t passing through all points in P on G_Q via the method in [16];
5: **for** each pair of adjacent points p and q in $\Pi_E(s, t|P)$ **do**
6: Find the shortest geodesic path π' from p to q through the fastest shortest geodesic path finding algorithm [31];
7: $\pi \leftarrow$ Concatenation of π and π';
8: **end for**
9: **return** π;

Algorithm 3: Distance Oracle-based Algorithm

Input: A set P of POIs, a source point s, a destination point t, a parameter δ, and a terrain surface T
Output: A path π from s to t passing through all points in P
1: Initialize the path π to be $\emptyset$;
2: Build a distance oracle O to index the pairwise geodesic distances among points in $P \cup \{s, t\}$;
3: Perform a distance query on O to find the geodesic distance for each pair of points in $P \cup \{s, t\}$;
4: Same as Lines 3-8 of Algorithm 2;
5: **return** π;

t is reached (Lines 11-12). Despite its simplicity, NNA suffers from a large approximation ratio (i.e., k), esp. when the number of POIs is large. This is because it relies on the nearest neighbor algorithm from [6], which has a running time of $O(N^2 \log N)$). The nearest neighbor algorithms in [25, 30, 33] cannot be applied to NNA since they are designed for finding the nearest neighbor in a fixed set of candidate points. In NNA, however, the set Q of candidate points is dynamic (since we remove one point from Q in each iteration).

Geodesic Distance-based Adaptations (GDBA). The pseudocode is shown in Algorithm 2. We first compute the pairwise geodesic path between each pair of points in $P \cup \{s, t\}$ (Line 2). Due to storage space constraints, only the distances (not the exact paths) are recorded. Then, we construct a complete and undirected query graph G_Q (Line 3). The vertices of G_Q are the points in $P \cup \{s, t\}$, and for each pair of points $u, v \in P \cup \{s, t\}$, there is an edge (u, v) in G_Q whose weight is the geodesic distance between u and v. Next, we adopt the *Travel Salesman Walk Problem (TSWP)* algorithm [16] on graph G_Q to find the shortest path from s to t that passes through each point in P at least once (Line 4). Since TSWP is also NP-hard, we apply the state-of-the-art approximation algorithm [16] (which is based on Integer Linear Programming) for TSWP, whose approximation ratio and running time are $\frac{3}{2}$ and $O(k)$, respectively. After obtaining the path, we compute the shortest geodesic path $\Pi_g(u, v)$ for each pair of adjacent points u and v along the path via an online geodesic shortest path finding algorithm (Lines 5-6). Finally, the concatenation of all paths found is returned (Line 7).

To compute the shortest geodesic path between u and v in the query graph G_Q, we can use either (a) the index-free exact algorithms, namely *KS* and *DIO* (which have the best empirical performance and the best theoretical complexity, respectively) or (b) the index-free approximation algorithms, namely *FS*, *US*, and *K-Algo*. These adaptations are called *KS-Adapt*, *DIO-Adapt*, *FS-Adapt*, *US-Adapt*, and *K-Algo-Adapt*, respectively (summarized in Table 2). Unfortunately, adaptations from this category suffer from a long time for constructing the query graph G_Q (i.e., $O(k^2 \cdot N \log N)$).

Distance Oracle-based Adaptations (DOA). This category only differs from GDBA by computing the pairwise geodesic distances for constructing of the query graph G_Q via the distance oracle. The pseudocode is shown in Algorithm 3. We build a distance oracle (i.e., an indexing structure for answering geodesic distance queries) on the terrain surface (Line 2) and utilize the distance oracle to find the pairwise geodesic distances between points in $P \cup \{s, t\}$ (Line 3).

Algorithm		Running Time ($\leq O(k \cdot N \log^2 N)$?)	Space Overhead ($O(k+N)$?)	Appro. Ratio (Small constant?)
NNA		No	Yes	No
GDBA	*KS-Adapt* [10]	No	No	Yes
	DIO-Adapt [31]	No	No	Yes
	FS-Adapt [17]	No	No	No
	US-Adapt [1]	No	No	Yes
	K-Algo-Adapt [11]	No	No	Yes
DOA	*SP-Oracle-Adapt* [7]	No	No	Yes
	SE-Oracle-Adapt [29]	No	No	Yes
	EAR-Oracle-Adapt [8]	No	No	Yes
Ours		Yes	Yes	Yes

Table 3: Comparison of Our Algorithm and Adaptations of Existing Algorithms

Although DOA mitigates the bottleneck of GDBA in terms of the running time by expediting the query graph construction in practice, its time complexity of building edges on the query graph is still $O(k^2)$. Besides, it also incurs an additional time cost for building the distance oracle and an extra large space overhead for the bulky index. Thus, it is not capable of scaling up to large terrains. It is also worth noting that the query graph in both GDBA and DOA has a space complexity $O(k^2)$, which is not scalable to the number of POIs. As summarized Table 2, we consider all existing distance oracles, namely *SP-Oracle*, *SE-Oracle* and *EAR-Oracle*, in this category of adaptations, and the adaptations are called *SP-Oracle-Adapt*, *SE-Oracle-Adapt*, and *EAR-Oracle-Adapt*, respectively.

3.3 Comparison with Our Algorithm

We compare our proposed algorithm (to be presented in Section 4) with baselines w.r.t. the running time, space overhead, and approximation ratios in Table 3. The following shows our observations.

- *Running time*: The running time of all Distance-Based Adaptations (DOA) is larger than $O(k^2 \cdot N \log N)$ since they involve the expensive building time of a bulky distance oracle on the terrain surface. Worse still, they are quadratic to k and thus, cannot scale well with a large number of POIs. The running time of Nearest Neighbor-Based Adaptation (NNA) is $O(k \cdot N^2 \log N)$, which is quadratic to N. In contrast, the running time of our algorithm is $O(k \cdot N \log^2 N)$ which is linear to both k and $N \log^2 N$.
- *Space overhead*: All Distance-Based Adaptations (GDBA) and all Distance Oracle-Based Adaptations (DOS) have a space overhead larger than $O(k^2)$ due to the space complexity of the query graph G_Q. This prevents their usage from the trip planning with a lot of POIs. Besides, the Distance Oracle-Based Adaptations (DOA) also need to maintain a bulky distance oracle on the terrain surface. The space overhead of NNA and our algorithm is $O(k)$.
- *Appro. Ratio*: The approximation ratios of NNA and FS-Adapt are large (i.e. $\geq k$). Thus, their returned trips can be very long and the users' experience will be degraded when there are a lot of POIs. Other algorithms have a small constant of approximation ratio.

4 Our Proposed Algorithm

In this section, we present our algorithm that beats existing ones in terms of running time, space overhead, and approximation ratio.

Overview. Taken a terrain surface T, a source point s, a destination point t, a set P of POIs, and an error parameter ϵ (a positive real number larger than 1) as input, our algorithm returns the shortest path $\Pi_g(s, t|P)$ from s to t on the terrain surface T that passes through all points in P. Here ϵ is a user-specified parameter for the trade-off between efficiency and accuracy (which will be discussed later). Our algorithm works in the following two steps.

(1) Spanning Graph Construction. In the first step, we construct a sparse spanning graph $\mathcal{G}$. The set of vertices on the spanning graph $\mathcal{G}$ is comprised of all points in $P \cup \{s, t\}$, and $O(k)$ edges between the vertices are introduced so that the spanning graph $\mathcal{G}$ concisely captures the distance relationship between points in $P \cup \{s, t\}$. In particular, given any pair of vertices u and v in $\mathcal{G}$, it holds that (a) there must be a path from u to v on $\mathcal{G}$, (b) the distance $d_{\mathcal{G}}(u, v)$ between u and v on the spanning graph $\mathcal{G}$ is not smaller than their geodesic distance (i.e., $d_{\mathcal{G}}(u, v) \geq d_g(u, v)$), and (c) the distance $d_{\mathcal{G}}(u, v)$ between u and v is at most $(1+\epsilon)$ times larger than their geodesic distance (i.e., $d_{\mathcal{G}}(u, v) \leq (1+\epsilon) \cdot d_g(u, v)$).

(2) Minimum Spanning Tree-based Path Finding. In the second step, we find the *minimum spanning tree (MPT)* of the spanning graph $\mathcal{G}$ rooted at s. Then, we order the vertices based on the tree and make sure that s (reps. t) is at the beginning (resp. end) of the order. Let $\mathcal{I}$ denote the order of vertices. Then, for each adjacent vertices p and p' in $\mathcal{I}$, we find the geodesic shortest path from p to p' and finally, we concatenate all the geodesic shortest paths. We prove that the approximation ratio of our algorithm is $2 \cdot (1+\epsilon)$.

In the following, Section 4.1 and Section 4.2 present the details of the two steps, respectively. Section 4.3 gives the theoretical analysis.

4.1 Spanning Graph Construction

This section presents the first step, i.e., *Spanning Graph Construction*.

Basic Concepts. Given a terrain surface T, a *disk* on T, denoted by $D(p, r)$ (where p is a point on T and r is a positive real number), is defined to be the set of all points on T whose geodesic distance to p (i.e., the center of the disk) is at most r (i.e., the radius of the disk). Mathematically speaking, $D(p, r)$ is $\{p' \mid d_g(p, p') \leq r\}$. Intuitively, $D(p, r)$ encloses all points on the terrain surface that are "close" to p.

Given two points p and q on the terrain surface, we consider the disk $D(q, r)$ centered at q with the radius r. Consider another point q' in $P \cup \{s, t\}$ on the terrain surface. Intuitively, if q' is in $D(q, r)$ and $d_g(p, q)$ is very large compared with r, the geodesic distance between p and q' will not deviate too much from the geodesic distance between p and q (based on which the disk $D(q, r)$ is constructed), since q' is very "close" to q. In this case, we can

Algorithm 4: Spanning Graph Construction(P, s, t, ϵ, T)

Input: A set P of POIs, a source point s, a destination point t, a parameter ϵ, and a terrain surface T
Output: A spanning graph $\mathcal{G}$ based on $P \cup \{s, t\}$

1 Initialize the set of vertices of $\mathcal{G}$ to be $P \cup \{s, t\}$;
2 **for** *each point* p *in* $P \cup \{s, t\}$ **do**
3 Initialize a set $\mathcal{X}$ to be $P \cup \{s, t\} \setminus \{p\}$;
4 **while** $\mathcal{X} \neq \emptyset$ **do**
5 Randomly sample a point q from $\mathcal{X}$;
6 Find the geodesic distance $d_g(p, q)$ online;
7 Add a new edge (p, q) into $\mathcal{G}$;
8 Set the weight of edge (p, q) to be $d_g(p, q)$;
9 $\sigma \leftarrow \frac{2}{\epsilon} + 2, r \leftarrow \frac{d_g(p,q)}{\sigma}$;
10 $O \leftarrow$ the set of points in $\mathcal{X}$ within $D(q, r)$;
11 **for** *each point* q' *in* O **do**
12 Remove q' from $\mathcal{X}$;
13 **return** $\mathcal{G}$;

roughly estimate the geodesic distance between p and q' based on the geodesic distance between p and q.

Algorithm. Based on these concepts, we develop our algorithm for constructing the spanning graph $\mathcal{G}$ that concisely captures the distance relationships between the points in $P \cup \{s, t\}$.

The pseudocode is shown in Algorithm 4. In the beginning, we initialize the set of vertices of $\mathcal{G}$ to be $P \cup \{s, t\}$ (Line 1). Then, we consider each vertex p by a for-loop (Lines 2-12). The goal is to utilize the aforementioned idea to group points in $P \cup \{s, t\} \setminus \{p\}$ based on the disk concept, and then establish incident edges from p to each group so that we can approximately estimate the geodesic distance between p and any points in the group.

To achieve this, we group points and construct incident edges of p in iterations. We initialize a set $\mathcal{X}$ to be $P \cup \{s, t\} \setminus \{p\}$ (Line 3) and execute a while-loop until $\mathcal{X}$ becomes empty (Lines 4-12).

Within the while-loop, we randomly sample a point q from $\mathcal{X}$ (Line 5). We compute the geodesic distance $d_g(p, q)$ between p and q through the one-the-fly geodesic distance computation algorithm (Line 6). The detailed computation will be later shown in *Implementation 1*. Then, we add into $\mathcal{G}$ a new edge (p, q) with weight $d_g(p, q)$ (Lines 7-8) and set the radius r to be $d_g(p, q)/\sigma$, where σ is $2 + 2/\epsilon$ (Line 9). After that, a set O is constructed (Lines 10) by including the points in $\mathcal{X}$ within disks $D(q, r)$ (see *Implementation 2* for details). Next, for each point $q' \in O$, we remove it from $\mathcal{X}$ (Lines 11-12). This is because q' is in the disk $D(q, r)$, and thus, the geodesic distance between p and q' can be estimated based on the geodesic distance between p and q. The while-loop ensures that each point in $P \cup \{s, t\} \setminus \{p\}$ is covered by at least one disk. Once $\mathcal{X}$ is empty, we proceed to the next iteration of the for-loop.

In this way, for each point in $P \cup \{s, t\}$, we established its incident edges. Finally, we return the corresponding graph $\mathcal{G}$ (Line 13).

Example 1 (Spanning Graph Construction). *Consider the example shown in Figure 2. Figure 2 (a) shows a terrain surface with seven points, namely s, t, p_1, p_2, p_3, p_4, and p_5. s and t are the source and destination points, and $\{p_1, p_2, p_3, p_4, p_5\}$ is the set P of POIs. Our goal is to find the (approximate) geodesic shortest trip path from s to t that passes through all points in P. We assume in this example that the error parameter ϵ is 0.5, and thus, σ follows to be $\frac{2}{\epsilon} + 2 = 6$.*

The vertices of the spanning graph $\mathcal{G}$ consist of all the seven points. Initially, there is no edge on $\mathcal{G}$. Figures 2 (a)-(e) demonstrate the first iteration of the for-loop in Lines 2-12 of Algorithm 4. In this iteration, the point p_1 is selected from $P \cup \{s, t\}$ and $\mathcal{X}$ is initialized to be $P \cup \{s, t\} \setminus \{p_1\} = \{p_2, p_3, p_4, p_5, s, t\}$ (Figure 2 (a)). Then, we execute the while-loop in Lines 4-12 of Algorithm 4 as follows.

Figure 2 (b) shows the operations in the first iteration of the while-loop, where the point p_5 is sampled from $\mathcal{X}$. We compute the geodesic shortest path from p_1 to p_5 on the terrain surface and get the geodesic distance $d_g(p_1, p_5)$. Then, we add to the spanning graph $\mathcal{G}$ a new edge (p_1, p_5) whose weight is set to $d_g(p_1, p_5)$. After that, the set O of points within the disk $D(p_5, \frac{d_g(p_1,p_5)}{\sigma})$ is computed to be $\{t, p_5\}$). We remove the points in O from $\mathcal{X}$, resulting in $\mathcal{X} = \{p_2, p_3, p_4, s\}$.

Figure 2 (c) shows the operations in the second iteration in the while-loop, where another point p_3 is sampled from $\mathcal{X}$. We compute the geodesic shortest path from p_1 to p_3 on the terrain surface and find the geodesic distance $d_g(p_1, p_3)$. Similarly, we add a new edge (p_1, p_3) to the spanning graph $\mathcal{G}$, whose weight is set to $d_g(p_1, p_3)$. Next, the set O of points within the disk $D(p_3, \frac{d_g(p_1,p_3)}{\sigma})$ is computed to be $\{p_3, p_4\}$. We remove the points in $\{p_3, p_4\}$ from $\mathcal{X}$, resulting in $\mathcal{X} = \{p_2, s\}$.

Similarly, Figure 2(d) and Figure 2(e) present the operations in the third and fourth iterations of the while-loop, where point p_2 and point s are sampled, respectively (details are omitted).

Finally, Figure 2 (f) gives the final spanning graph $\mathcal{G}$ obtained by Algorithm 4 after all seven points are processed. As shown there, the spanning graph is a weighted and connected graph. For each edge (u, v) in $\mathcal{G}$, the weight of the edge is $d_g(u, v)$ on the terrain surface.

Implementation 1: Online Computation of the Geodesic Distance. In Line 6 of Algorithm 4, we call an existing index-free exact geodesic distance query processing algorithm to find the geodesic distance $d_g(p, q)$ between the two given points p and q online. According to the theoretical analysis of [10, 29, 30], this online computation of the geodesic distance takes $O(N \log^2 N)$ time.

Implementation 2: Find the Set of Points within A Given Disk. In Lines 9-10 of Algorithm 4, we perform an existing index-free exact geodesic distance query processing algorithm to find the set of points in $P \cup \{s, t\}$ within a given disk $D(c, r)$. In particular, this algorithm starts from a given center c and visits each face on the terrain surface in the ascending order of their geodesic distances to c. This algorithm can finally retrieve the list of points in $P \cup \{s, t\}$ that are within the disk $D(c, r)$. According to the theoretical analysis of [10, 29, 30], this online computation of the points within a given disk takes $O(N \log^2 N)$ time.

4.2 Minimum Spanning Tree-based Path Finding

The second step takes the spanning graph $\mathcal{G}$ obtained in the first step, the source vertex s on $\mathcal{G}$, the destination vertex t on $\mathcal{G}$, and the terrain surface T as input and returns the path from source s to destination t on the terrain surface T that passes through all POIs.

Algorithm. Algorithm 5 shows the pseudocode of our Minimum Spanning Tree-based Path Finding algorithm. We begin by finding the minimum spanning tree T of $\mathcal{G}$ (i.e., the spanning tree that

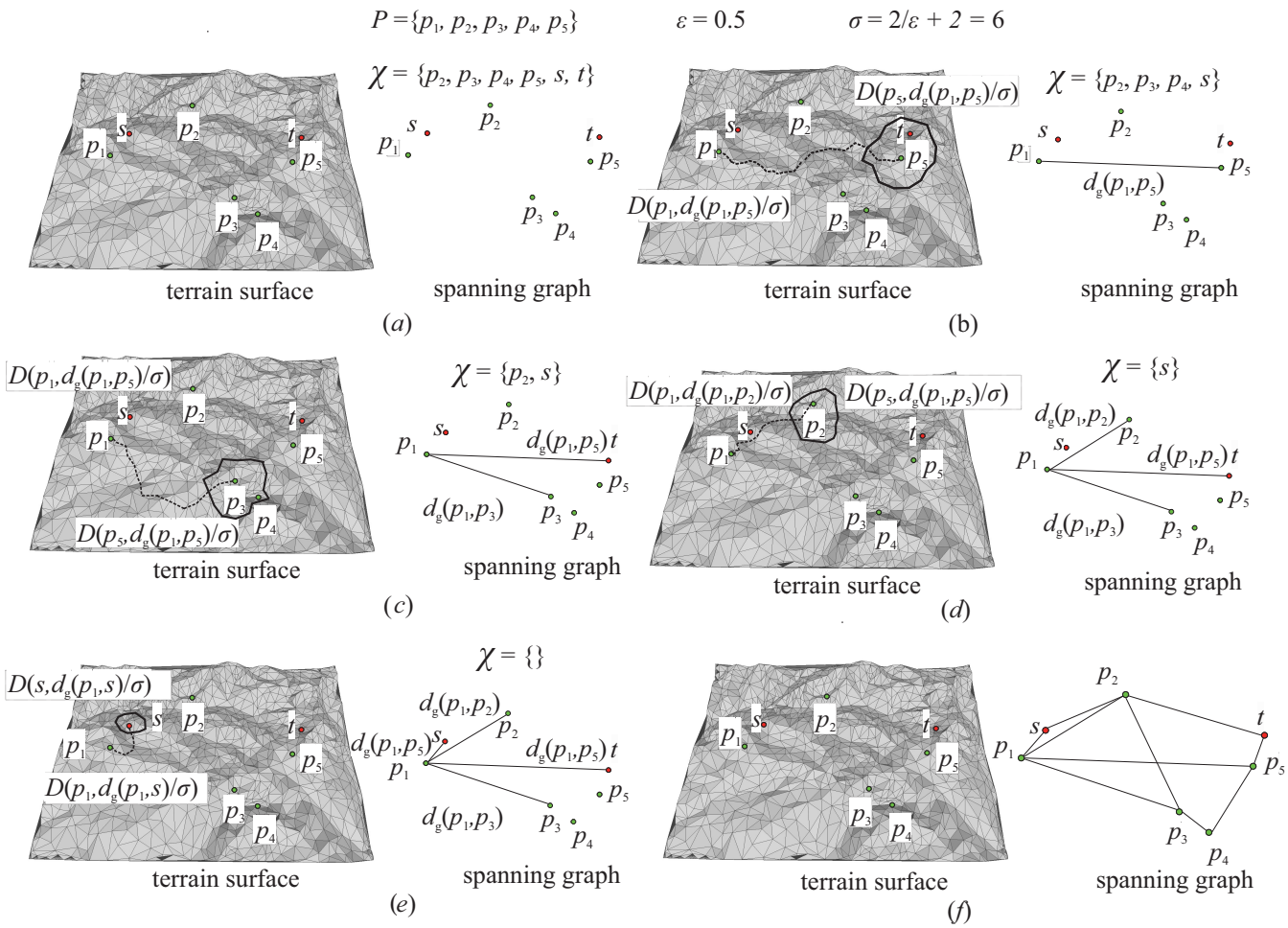

Figure 2: An Example of Spanning Graph Construction

has the minimum weight among all the possible spanning trees) rooted at source s via the renowned *Kruskal's algorithm* [14] (Line 1). Next, we sort all vertices on the tree T in the order of the pre-order traversal (Line 2). Let $\mathcal{I}$ denote this sorted list of vertices on T. We swap destination t with the last vertex in $\mathcal{I}$ (Lines 3-4). Then, we enter a for-loop (Lines 5-8), where for any two adjacent vertices u and v in $\mathcal{I}$, we call an existing index-free geodesic shortest path finding algorithm to find the geodesic shortest path $\Pi_g(u, v)$ from u to v (which takes $O(N \log^2 N)$ time by [10, 29, 30]). Finally, we return the concatenation of all paths found (Line 9).

Example 2 (Minimum Spanning Tree-Based Path Finding). *Figure 3 gives an example of our minimum spanning tree-based path finding algorithm. Figure 3 (a) shows the spanning graph $\mathcal{G}$ obtained by Algorithm 4 in Example 1. Figure 3 (b) presents the minimum spanning tree of the spanning graph. The pre-order traversal of all vertices on the minimum spanning tree gives an order $\mathcal{I}$ is $(s, p_1, p_3, p_4, p_2, t, p_5)$ and we swap t and the last vertex (p_5) in $\mathcal{I}$, resulting in an order $\mathcal{I} = (s, p_1, p_3, p_4, p_2, p_5, t)$ shown in Figure 3 (c). Finally, for each pair of adjacent points u and v in $\mathcal{I}$, we find the geodesic shortest path from u to v as shown in Figure 3 (d). The concatenation of all paths found forms the trip from s to t.*

Algorithm 5: Minimum Spanning Tree-Based Path Finding

Input: A spanning graph $\mathcal{G}$, a source vertex s, a destination vertex t, and a terrain surface T
Output: A path π from s to t passing through all points in P on the terrain surface

1 Find the minimum spanning tree T of $\mathcal{G}$ rooted at s;
2 Sort all vertices in the tree T by the pre-order traversal;
3 Let $\mathcal{I}$ be the sorted list of vertices;
4 Swap t and the last vertex in $\mathcal{I}$;
5 Initialize the path π to be $\emptyset$;
6 **for** *each two adjacent vertices u and v in $\mathcal{I}$* **do**
7 Call an existing index-free geodesic shortest path algorithm to find the geodesic shortest path $\Pi_g(u, v)$ from u to v;
8 $\pi \leftarrow$ Concatenation of π and $\Pi_g(u, v)$;
9 **return** π;

4.3 Theoretical Analysis

This section presents our theoretical analysis on the spanning graph (Theorem 2 and Theorem 3), the running time, space overhead (The-

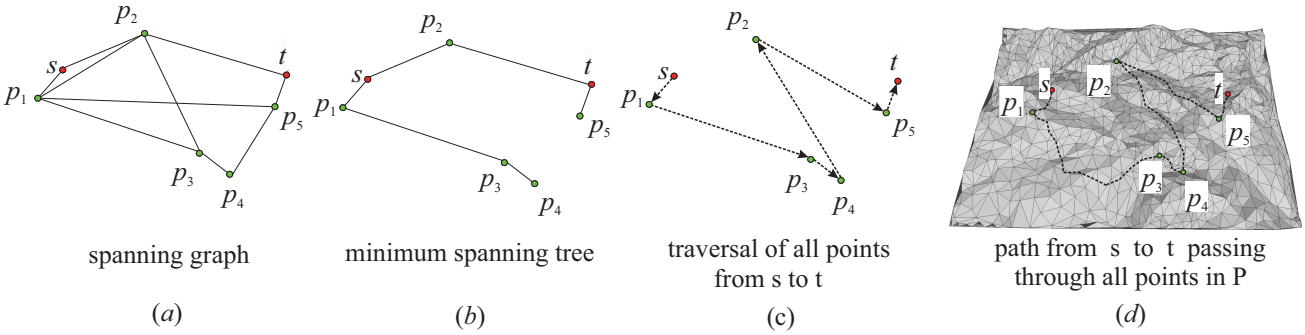

Figure 3: An Example of Minimum Spanning Tree-Based Path Finding

orem 4), and the approximation ratio (Theorem 5) of our algorithm.

Theorem 2 (Distance on the Spanning Graph). *The spanning graph $\mathcal{G}$ is a connected graph. That is, for any pair of distinct vertices u and v on $\mathcal{G}$, there exists a path from u to v on $\mathcal{G}$. In addition, it holds that $d_g(u,v) \le d_{\mathcal{G}}(u,v) \le (1+\epsilon)\cdot d_g(u,v)$, where $d_{\mathcal{G}}(u,v)$ is the shortest distance between u and v on the spanning graph $\mathcal{G}$.*

Proof. We first prove the following lemma.

Lemma 1. *For any pair of distinct vertices u and v on $\mathcal{G}$, there is an edge (u,x) in $\mathcal{G}$ such that $d_g(x,v) \le \frac{d_g(u,x)}{\sigma}$.*

Proof. We prove this lemma by contradiction. Assume that there is no edge (u,x) in $\mathcal{G}$ such that $d_g(x,v) \le \frac{d_g(u,x)}{\sigma}$. That is, for each edge (u,y) in $\mathcal{G}$, $d_g(y,v) > \frac{d_g(u,y)}{\sigma}$ (i.e., v is outside the disk $D(y, \frac{d_g(u,y)}{\sigma})$). Consider the iteration in the for-loop in Lines 2-12 in Algorithm 4, where the point u is selected (i.e., $p = u$ in this iteration). According to Line 3 and Lines 11-12 of Algorithm 4, $\mathcal{X}$ is initialized to be $P \cup \{s,t\} \setminus \{p\}$ and we remove a point q' only if the point q' is within a disk considered before. Since the point v is not in any disk considered (according to our assumption), $\mathcal{X}$ must not be empty. But it contradicts with Line 4 in Algorithm 4 (i.e., when this iteration of the for-loop in Algorithm 4 terminates, $\mathcal{X}$ must be $\emptyset$). □

By Lemma 1, we obtain that for any pair of distinct vertices u and v on $\mathcal{G}$, there is an edge (u,x) in $\mathcal{G}$ such that $d_g(x,v) \le \frac{d_g(u,x)}{\sigma}$. If x is v (in this case, $d_{\mathcal{G}}(u,v(=x)) = d_g(u,v(=x))$), then this theorem must be true. Then, we discuss about the case where x is not v. Then, since v lies in the disk $D(x, \frac{d_g(u,x)}{\sigma})$), we consider the following iterative procedure.

Iterative Procedure: Initially, we assign a list L to be $< (u,v) >$. In iteration $i, i \in [1,+\infty]$, we consider each pair of points in L. Let (o,o') denote the pair of points in L. If (o,o') is not an edge in $\mathcal{G}$, then according to Lemma 1, we can find an edge (o,t') in $\mathcal{G}$ such that o' lies in the disk $D(t', \frac{d_g(o,t')}{\sigma})$. Then, we replace (o,o') with (o,t'), (t',o') in L. If each pair of points in L is an edge in $\mathcal{G}$, then we stop the iterative procedure. Otherwise, we go to the next iteration. Finally, when this iterative procedure terminates, each pair of points in L is an edge in $\mathcal{G}$ and the edges in L forms a path from u to v on $\mathcal{G}$.

Consider list L after the i-th iteration of our iterative procedure. We call a pair (o,o') of points in L *terminal pair* if (o,o') is an edge on $\mathcal{G}$. Otherwise, we call (o,o') *non-terminal pair.*

Then, we introduce a key concept called *maximum weight* of L in the i-th iteration as follows.

Concept 1. *Consider our iterative procedure, the maximum weight w_L^i of L in the i-th iteration is the maximum geodesic distance between each non-terminal pair of points in L created in the i-th iteration.*

The lemma below is for the concept of maximum weight of L.

Lemma 2. *The maximum weight w_L^i of the list L is at most $\frac{d_g(u,v)}{(\sigma-1)^i}$.*

Proof. We prove this lemma by induction. Assume that the maximum weight w_L^x of the list L is at most $\frac{d_g(u,v)}{(\sigma-1)^{x-1}}$ for all positive integer x in the range of $[1, i-1]$. This is trivially true when i is equal to 1. Consider the i-th iteration. Let (o,o') denote one pair of points considered in the iteration. If (o,o') is an edge in $\mathcal{G}$, we simply keep it intact and will not create any new edge. If (o,o') is not an edge in $\mathcal{G}$, (o,o') must be created in the $i-1$-th iteration since otherwise, it must be replaced in the $i-1$-th iteration and can not appear in the i-th iteration. By our assumption, we obtain that $d_g(o,o')$ is at most $\frac{d_g(u,v)}{(\sigma-1)^{i-1}}$. Then, according to Lemma 1, we can find the edge $< o,t' >$ in $\mathcal{G}$ such that o' is in the disk $D(t', \frac{d_g(o,t')}{\sigma})$. And we replace (o,o') with (o,t'), (t',o'). Thus, (o,t') must be an edge on $\mathcal{G}$ and according to Lemma 1, o' is in the disk $D(t', \frac{d_g(o,t')}{\sigma})$. That is, $d_g(t',o'))$ is at most $\frac{d_g(o,t')}{\sigma}$. Together with triangle inequality, we obtain that $d_g(o,o') \ge d_g(o,t') - d_g(o',t') \ge \sigma \cdot d_g(o',t') - d_g(o',t')$. Then, we further obtain that $d_g(o',t') \le \frac{1}{\sigma-1}\cdot d_g(o,o')$. Since $d_g(o,o')$ is at most $\frac{d_g(u,v)}{(\sigma-1)^{i-1}}$, we obtain that

$$d_g(o',t') \le \frac{1}{\sigma-1}\cdot d_g(o,o') \le \frac{d_g(u,v)}{(\sigma-1)^i}$$

□

Lemma 3. *There are at most $\lceil \log_{\sigma-1} \frac{d_g(u,v)}{\min_{o,o' \in P\cup\{s,t\}} d_g(o,o')} \rceil$ iterations in our iterative procedure.*

Proof. We prove this by contradiction. Assume that in $(\lceil \log_{\sigma-1} \frac{d_g(u,v)}{\min_{o,o' \in P\cup\{s,t\}} d_g(o,o')} \rceil + 1)$-th iteration, our algorithm still does not terminate. By Lemma 2, at $\lceil \log_{\sigma-1} \frac{d_g(u,v)}{\min_{o,o' \in P\cup\{s,t\}} d_g(o,o')} \rceil$-

th iteration, the maximum weight of L will be smaller than $\min_{o,o' \in P \cup \{s,t\}} d_g(o, o')$. That is, for each non-terminal pair of points (x, y) in L, the geodesic distance between x and y is smaller than $\min_{o,o' \in P \cup \{s,t\}} d_g(o, o')$. Contradiction! □

By Lemma 3, we obtain that our iterative procedure finally returns a valid path from u to v on the spanning graph $\mathcal{G}$. Thus, we obtain that u and v is connected on $\mathcal{G}$.

Consider the shortest path $\Pi_{\mathcal{G}}(u, v) = (u, o_1, o_2, \ldots, o_i, \ldots, o_m, v)$ from u to v on the spanning graph $\mathcal{G}$. The length of the path $\Pi_{\mathcal{G}}(u, v)$ is equal to $d_g(u, o_1) + d_g(o_1, o_2) + \cdots + d_g(o_i, o_{i+1}) + \cdots + d_g(o_m, v))$. Since the geodesic distance is also a metric where triangle inequality holds, we obtain that the length of the shortest path $\Pi_{\mathcal{G}}(u, v)$ from u to v on the spanning graph $\mathcal{G}$ is larger than or equal to the geodesic distance $d_g(u, v)$ between u and v on the terrain surface (that is, $d_g(u, v) \leq d_{\mathcal{G}}(u, v)$).

Finally, we prove that $d_{\mathcal{G}}(u, v) \leq (1 + \epsilon) \cdot d_g(u, v)$. Consider the list L maintained by our aforementioned iterative procedure.

Lemma 4. *Each terminal pair (o, o') (i.e., an edge on $\mathcal{G}$) inserted into L in the i-th iteration in our iterative procedure has the weight smaller than or equal to $\sigma \cdot d_g(u, v) \cdot (\frac{1}{\sigma - 1})^i$.*

Proof. We prove this lemma by induction. We first consider the base case. Consider the first iteration where (u, v) is considered. We find the edge $< u, x >$ in $\mathcal{G}$ such that v is in the disk $D(x, \frac{d_g(u,x)}{\sigma})$. Next, we replace (u, v) with (u, x), (x, v). Thus, (u, x) must be an edge on $\mathcal{G}$ (i.e., a terminal pair of points). Since v is in the disk $D(x, \frac{d_g(u,x)}{\sigma}))$, $d_g(x, v))$ are at most $\frac{d_g(u,x)}{\sigma}$. According to triangle inequality, we obtain that $d_g(u, v) \geq d_g(u, x) - d_g(x, v) \geq d_g(u, x) - \frac{d_g(u,x)}{\sigma}$. Then, we further obtain that $d_g(u, x) \leq \frac{\sigma}{\sigma - 1} \cdot d_g(u, v)$.

Consider the i-th iteration. Assume that each terminal pair (t, t') (i.e., an edge on $\mathcal{G}$) inserted into L in the x-th iteration in our iterative procedure has the weight smaller than or equal to $\sigma \cdot d_g(u, v)(\frac{1}{\sigma - 1})^x$ for all positive integer x in the range of $[1, i - 1]$. Let (o, o') denote one pair of points considered in the i-th iteration. If (o, o') is an edge in $\mathcal{G}$, it must be inserted in the previous iterations. By our assumption, the lemma is correct. If (o, o') is not an edge in $\mathcal{G}$, (o, o') must be created in the $i - 1$-th iteration since otherwise, it must be replaced in the $i - 1$-th iteration and can not appear in the i-th iteration. By Lemma 2, we obtain that $d_g(o, o')$ is at most $d_g(u, v)(\frac{1}{\sigma - 1})^{i-1}$. Then, according to Lemma 1, we can find the edge $< o, t' >$ in $\mathcal{G}$ such that o' is in the disk $D(t', \frac{d_g(o,t')}{\sigma})$. And we replace (o, o') with (o, t'), (t', o'). Since o' is in the disk $D(t', \frac{d_g(o,t')}{\sigma})$, $d_g(t', o')$ are at most $\frac{d_g(o,t')}{\sigma}$. According to triangle inequality, we obtain that $d_g(o, o') \geq d_g(o, t') - d_g(o', t') \geq d_g(o, t') - \frac{d_g(o',t')}{\sigma}$. Then, we further obtain that

$$d_g(o, t') \leq \frac{\sigma}{\sigma - 1} \cdot d_g(o, o') \leq d_g(u, v) \cdot \sigma \frac{1}{(\sigma - 1)^i}$$

□

Thus, the length of the path found by our iterative procedure is

$$\sum_{i=1}^{k} \frac{\sigma \cdot d_g(u, v)}{(\sigma - 1)^i} = \sum_{i=1}^{k} \frac{(\frac{2}{\epsilon} + 2) \cdot d_g(u, v)}{(\frac{2}{\epsilon} + 1)^i}$$

$$= \sum_{i=1}^{k} ((\frac{\epsilon}{2 + \epsilon})^i) \cdot (\frac{2}{\epsilon} + 2) \cdot d_g(u, v)$$

$$= \frac{\epsilon}{2 + \epsilon} \cdot \frac{1 - (\frac{\epsilon}{2+\epsilon})^k}{1 - \frac{\epsilon}{2+\epsilon}} \cdot (\frac{2}{\epsilon} + 2) \cdot d_g(u, v)$$

$$\leq \frac{\epsilon}{2 + \epsilon} \cdot \frac{1}{1 - \frac{\epsilon}{2+\epsilon}} \cdot (\frac{2}{\epsilon} + 2) \cdot d_g(u, v) = (1 + \epsilon) \cdot d_g(u, v)$$

□

Theorem 3 (The Number of Edges on the Spanning Graph). *The number of edges on the spanning graph $\mathcal{G}$ is $O(\frac{k}{\epsilon^{2\beta}})$.*

Proof. Consider a vertex u on $\mathcal{G}$ and the iteration of the for-loop in Lines 2-12 in Algorithm 4, where u is considered (i.e., in this iteration, $p = u$.). Let N_u denote the set of all vertices on $\mathcal{G}$ such that for each vertex $v \in N_u$, there is an edge (u, v) created in this iteration (i.e., $N_u = \{v | (u, v)$ is created in this iteration$\}$) Let l_{min} and l_{max} denote $\min_{v \in N_u} d_g(u, v)$ and $\max_{v \in N_u} d_g(u, v)$, respectively. Then, we partition all vertices in N_u in several groups as follows. The i-th group, denoted by N_u^i, contains all points in N_u such that their geodesic distance to u is in the range of $[l_{min} \cdot 2^{i-1}, l_{min} \cdot 2^i]$, where $i \in [1, \lceil \log \frac{l_{max}}{l_{min}} \rceil]$ (mathematically speaking, $N_u^i = \{v | v \in N_u, l_{min} \cdot 2^{i-1} \leq d_g(u, v) \leq l_{min} \cdot 2^i\}$). Consider the i-th group N_u^i.

Lemma 5. *The pairwise geodesic distance between any two vertices v and w in the i-th group N_u^i is at least $l_{min} \cdot \frac{2^{i-1}}{\sigma}$.*

Proof. We prove this lemma by contradiction. Assume that there exists two vertices v and w in the i-th group N_u^i such that their geodesic distance $d_g(v, w)$ is smaller than $l_{min} \cdot \frac{2^{i-1}}{\sigma}$. Consider the two edges (u, v) and (u, w). Without loss of generality, we assume that the edge (u, v) is created before (u, w). Since $d_g(u, v) \geq l_{min} \cdot 2^{i-1}$ (by the definition of N_u^i) and $d_g(v, w) < l_{min} \cdot \frac{2^{i-1}}{\sigma}$ (by our assumption), we obtain that $d_g(v, w) \leq \frac{d_g(u,v)}{\sigma}$. Note that by Line 10 of Algorithm 4, O contains all points in the disk $D(u, \frac{d_g(u,v)}{\sigma})$. Thus, $w \in O$ and w is removed from $\mathcal{X}$ right after (u, v) is created. But according to our assumption, (u, w) is created later which implies that w has not been removed from $\mathcal{X}$ after (u, v) is created. Contradiction! □

Then, we recite the following lemma from [29].

Lemma 6 (Lemma 8 of [29]). *A disk $D(p, r)$ centered at p with the radius r can hold at most $(2^{2\beta})^j$ points on the terrain surface, the minimum pairwise geodesic distance is at least $\frac{r}{2^j}$, where β is a terrain-related parameter called* capacity dimension *and it is a positive real number in the range of* $[1.5, 2]$.

By the definition of N_u^i, all points in N_u^i must be located within the disk $D(u, l_{min} \cdot 2^i)$. By Lemma 5, the pairwise geodesic distance between any two vertices v and w in the i-th group N_u^i is at least $l_{min} \cdot \frac{2^{i-1}}{\sigma}$. Thus, by Lemma 8 [29], we obtain that the number

of points in N_u^i is at most $(2\sigma)^{2\beta}$. Then, we further obtain the following lemma.

Lemma 7. *The number of edges created in each iteration in the for-loop in Lines 2-12 of Algorithm 4 is $O(\log\alpha \cdot \sigma^{2\beta})$, where α is the aspect ratio of $P \cup \{s,t\}$ and it is defined to be $\frac{\max_{u,v\in P\cup\{s,t\}} d_g(u,v)}{\min_{u,v\in P\cup\{s,t\}} d_g(u,v)}$.*

Proof. Since the the number of points in N_u^i is at most $(2\sigma)^{2\beta}$, we obtain that the size of N_u is $\sum_{i=1}^{\lceil \log \frac{l_{max}}{l_{min}} \rceil} |N_u^i| \leq \sum_{i=1}^{\lceil \log \frac{l_{max}}{l_{min}} \rceil} (2\sigma)^{2\beta} \leq \sum_{i=1}^{\lceil \log \alpha \rceil} (2\sigma)^{2\beta} = O(\log\alpha\sigma^{2\beta})$. □

Then, we obtain that the number of edges created in each iteration of the for-loop in $\mathcal{G}$ is $O(\sum_{u\in P\cup\{s,t\}} \log\alpha\sigma^{2\beta}) = O(\sigma^{2\beta}\cdot k) = O(\frac{k}{\epsilon^{2\beta}})$. Note that $\log\alpha$ (i.e., $\log \frac{\max_{u,v\in P\cup\{s,t\}} d_g(u,v)}{\min_{u,v\in P\cup\{s,t\}} d_g(u,v)}$) is very small on terrain surfaces as shown in Lemma 2 and its subsequent discussion of [29]. $\log\alpha$ is at most 30 according to the experiments of [29]. Even in an extreme case (where $\min_{u,v\in P\cup\{s,t\}} d_g(u,v)$ is one nanometer (10^{-9} m)) and $\max_{u,v\in P\cup\{s,t\}} d_g(u,v)$ is the length of the Earth Equator (which is around 4×10^7 m), it is at most 56. Thus, we simply treat $\log\alpha$ as a small constant. □

Theorem 4 (Time and Space). *The running time of our algorithm is $O(k\cdot\frac{N\cdot\log^2 N}{\epsilon^{2\beta}})$ and the space overhead of our algorithm is $O(\frac{k}{\epsilon^{2\beta}})+N)$.*

Proof. Consider Algorithm 4. In Algorithm 4, Line 1 takes $O(k)$ time. There are totally k iterations in the for-loop in Lines 2-12. Line 3 in the loop takes $O(k)$ time. Then, consider the while-loop in Lines 4-12. Line 5 takes constant time and the operations in Lines 6-12 create an edge in the spanning graph $\mathcal{G}$. According to Lemma 7, we obtain that the operations in Lines 7-12 are performed $O(\log\alpha\cdot\sigma^{2\beta})$ times. Line 6 takes $O(N\log^2 N)$ time by Implementation Detail 1. Lines 7-9 take constant time. Line 10 take $O(N\log^2 N)$ time for finding the disk by Implementation Detail 2. Since each point in $P\cup\{s,t\}$ is removed from $\mathcal{X}$ only once in one iteration of the for-loop in Lines 2-12, the accumulated running time of the operations in Lines 11-12 within one iteration of the for-loop shown in Lines 2-12 is $O(k^2)$. Thus, we obtain that the running time of Algorithm 4 is $O(\frac{k}{\epsilon^{2\beta}}\cdot N\log^2 N)$. Since the number of edges in $\mathcal{G}$ is $O(\frac{k}{\epsilon^{2\beta}})$ by Theorem 3 and the cardinality of $\mathcal{X}$ is at most k, we obtain that the space consumption of Algorithm 4 is $O(\frac{k}{\epsilon^{2\beta}}+N)$.

Consider Algorithm 5. We adopt the renowned *Kruskal's Algorithm* [14] to find the minimum spanning tree. According to its results, the running time of Line 1 in Algorithm 5 is $O(k\log k)$. Line 2 takes $O(k\log k)$ time for sorting, and Lines 3-5 take constant time. In the for-loop in Lines 6-8, we invoke a geodesic shortest path finding algorithm (which takes $O(N\log^2 N)$ time by the results of [29, 30]) by $k-1$ times. Thus, Lines 6-8 takes $O(k\cdot N\log^2 N)$ time. Since the MST takes $O(k)$ space, the space consumption of Algorithm 5 is $O(k+N)$. □

Theorem 5 (The Approximation Ratio). *The approximation ratio of our algorithm is $2\cdot(1+\epsilon)$.*

Proof. Consider the minimum spanning tree T (of the spanning graph $\mathcal{G}$) found by Algorithm 5. Let $\Pi'_{\mathcal{G}}(s,t|P)$ denote the shortest path from s to t passing through all vertices on the spanning graph $\mathcal{G}$. By the definition of minimum spanning tree, the weight $W(T)$ of T (recall that the weight of a tree is the sum of the weights of all edges in the tree) must be smaller than or equal to the length of $\Pi'_{\mathcal{G}}(s,t|P)$ (since $\Pi'_{\mathcal{G}}(s,t|P)$ is also a spanning tree). By the definition of the pre-order traversal of a tree, the length of the path, denoted by $\pi(s,t|P)$, found by Algorithm 5 is at most twice of $W(T)$. Thus, we obtain that the length of the path $\pi(s,t|P)$ found by Algorithm 5 is at most twice of the length of the path $\Pi'_{\mathcal{G}}(s,t|P)$. That is,

$$|\pi(s,t|P)| \leq 2\cdot w(T) \leq 2\cdot|\Pi'_{\mathcal{G}}(s,t|P)| \quad (1)$$

Recall that $\Pi_g(s,t|P)$ denotes the shortest geodesic path from s to t passing through all points in P on the terrain surface. Let $\mathcal{L} = (o_1 = s, o_2, \ldots, o_i, \ldots, o_{k+2} = t)$ denote the list of all points in $P\cup\{s,t\}$ sorted in the their order in $\Pi_g(s,t|P)$. Then, the length $|\Pi_g(s,t|P)|$ is equal to $\sum_{i=1}^{k+1} d_g(o_i,o_{i+1})$. By Theorem 2, we obtain

$$|\Pi_g(s,t|P)| = \sum_{i=1}^{k+1} d_g(o_i,o_{i+1}) \geq \sum_{i=1}^{k+1}\frac{d_{\mathcal{G}}(o_i,o_{i+1})}{1+\epsilon} = \frac{1}{1+\epsilon}\sum_{i=1}^{k+1} d_{\mathcal{G}}(o_i,o_{i+1}) \quad (2)$$

In the inequality above, $d_{\mathcal{G}}(o_i,o_{i+1})$ is the shortest distance between o_i and o_{i+1} on $\mathcal{G}$. By the definition of $\Pi'_{\mathcal{G}}(s,t|P)$ and Inequality (2), we obtain that

$$|\Pi'_{\mathcal{G}}(s,t|P)| \leq \sum_{i=1}^{k+1} d_{\mathcal{G}}(o_i,o_{i+1}) \leq (1+\epsilon)\cdot|\Pi_g(s,t|P)| \quad (3)$$

By Inequality (1) and Inequality (3), we obtain that

$$|\pi(s,t|P)| \leq 2\cdot|\Pi'_{\mathcal{G}}(s,t|P)| \leq 2\cdot(1+\epsilon)\cdot|\Pi_g(s,t|P)| \quad (4)$$

That is, the approximation ratio $\frac{|\pi(s,t|P)|}{|\Pi_g(s,t|P)|}$ is at most $2\cdot(1+\epsilon)$. □

Dataset	No. of Vertices	Region Covered	Reference
HM	793	15 km^2	[26]
HL	2,470	49 km^2	[26]
RM	3,696	71 km^2	[26]
BH (L)	146,547	$14km\times10km$	[11, 29]
EP (L)	164,238	$10.7km\times14km$	[11, 29]
SF (L)	172,186	$14km\times11.1km$	[11, 29]
BH (H)	1,318,844	$14km\times10km$	[11, 29]
EP (H)	1,392,236	$10.7km\times14km$	[11, 29]
SF (H)	1,539,082	$14km\times11.1km$	[11, 29]

Table 4: Dataset Statistics

5 Experiment

Using real-life terrain datasets, we experimentally evaluated the effectiveness and efficiency of our proposed algorithm for TPTS.

5.1 Experimental Setting

The experiment was conducted on a Linux machine with 3.60GHz Intel Corel CPU and 32 GB memory. All algorithms were implemented in C++. The experiment setting is detailed as follows.

Datasets. We adopted nine real terrain datasets with various sizes, and the statistics of them are shown in Table 4. All datasets were collected from the United States Geological Survey (USGS) system [26]. Following the existing study [8], we first considered three

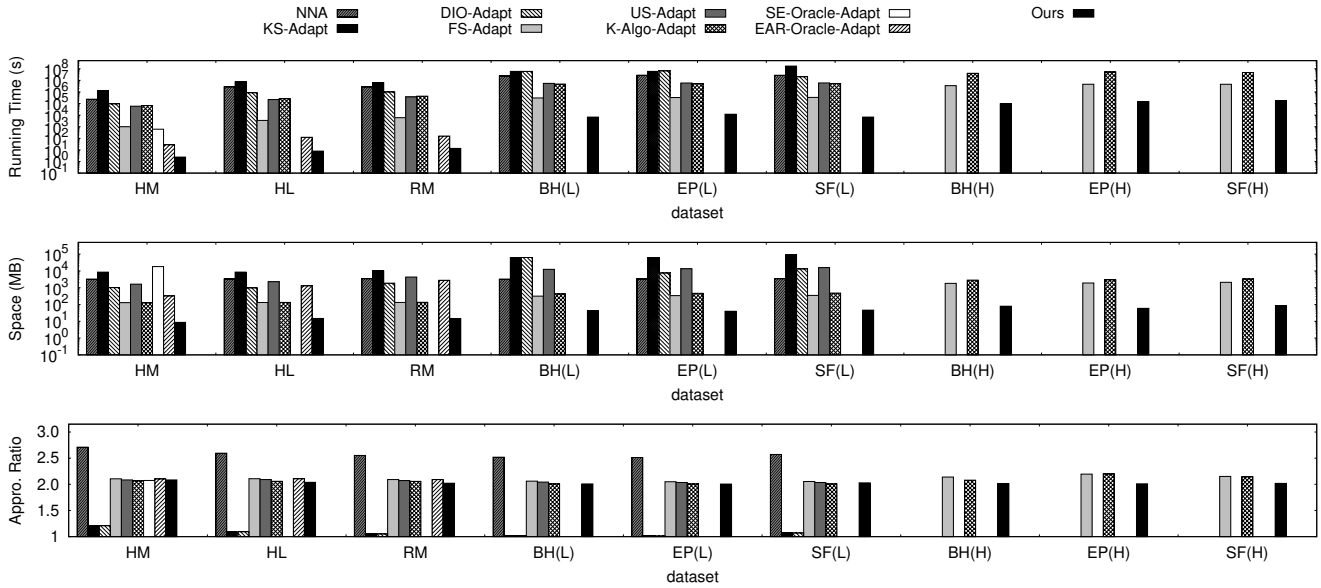

Figure 4: Results on All Datasets

small datasets (each of which has fewer than 4,000 vertices), namely *Horst Mountain (in short, HM)*, *HeadLightMountain (in short, HL)*, and *RobinsonMountain (in short, RM)*. Then, we considered three large datasets, namely *Bearhead (BH)*, *Eaglepeak (EP)*, and *San Francisco South (SF)*. Each large dataset has two versions with different resolutions and correspondingly different numbers of vertices. We call the datasets with low resolution *BH (L)*, *EP (L)*, and *SF (L)*, and call the ones with high resolution *BH (H)*, *EP (H)*, and *SF (H)*. To construct a query in the experiment, we randomly generate two points s and t and a set P of points on the terrain surface and each point in $P \cup \{s, t\}$ is generated by an existing method in the experiment of [8]. Each result reported in our experiment is averaged over 50 queries.

Algorithms. We tested our proposed algorithm and all adaptations of the existing algorithms presented in Section 3.2 except *SP-Oracle-Adapt* (which cannot be built even in the smallest terrain datasets due to its very large space consumption). Besides, according to the results of [8, 29], *SE-Oracle* and *EAR-Oracle* are shown to be superior to *SP-Oracle* in terms of all measurements. Since *SE-Oracle-Adapt* and *EAR-Oracle-Adapt* are both considered in our experiments, we safely ignore *SP-Oracle-Adapt*.

Factors & Measurements. We studied the effects of the following parameters: (1) the number N of vertices on the terrain surface, which reflects the complexity of the terrain datasets, (2) the number k of POIs in the trip planning query, and (3) the error parameter ϵ. By default, ϵ is set to be 0.2 and k is set to be 1000.

We consider the following measurements in the experiment. (1) The running time for processing a trip planning query, (2) the space consumption in the query processing, and (3) the *actual* approximation ratio, which is computed by $\frac{COST}{COST^*}$, where $COST$ (resp. $COST^*$) is the length of the path returned by the algorithm (resp. the length of the optimal solution). To compute the optimal solution, we first compute the query graph G_Q and the weight of each edge in G_Q is computed by *DIO* algorithm [31]. To boost the computation of G_Q, we employ the parallel computation and invoke multiple threads to compute the weights of multiple edges in parallel. Then, we perform a standard technique for *Travel Salesman Walk Problem (TSWP)* algorithm [16] to find the exact solution on G_Q.

5.2 Experimental Results

In this section, we first present the results on all real datasets and then report the sensitivity tests, by varying different parameters.

Results on All Real Datasets. The running time, space consumption, and actual approximation ratio of each algorithm are shown in Figure 4. Note that some baselines are not shown in certain datasets when they either run out of memory or cannot be finished in a reasonable time (for reasons that will be elaborated in detail shortly). As shown in Figure 4, we have the following observations.

(1) Our algorithm enjoyed the smallest running time. The running time of each adaptation was significantly larger than that of ours by up to more than two orders of magnitude. In particular, both *NNA* and *DIO-Adapt* could not scale to the three high resolution datasets due to their quadratic cost w.r.t. the number of vertices on the terrains. *KS-Adapt* had to compute the exact geodesic distance for each pair of points in $P \cup \{s, t\}$, without effective early termination, and thus, it could not scale to the three largest datasets either.

(2) Our algorithm had the best space efficiency and could scale up to the largest terrain datasets with millions of vertices. In contrast, the space consumption of all adaptations was significantly larger than that of ours. This is because these adaptations need to build a query graph G_Q with $O(k^2)$ space cost. Besides, each DOA adaptation (i.e., *SE-Oracle-Adapt* and *EAR-Oracle-Adapt*) needed to maintain a bulky indexing structure. Therefore, their performance was only reported in the smallest *HM* dataset (resp. the three smallest datasets). Moreover, *US-Adapt* introduced too many auxiliary points on the terrains, leading to a large space overhead. Consequently, it could not scale to the three high resolution datasets.

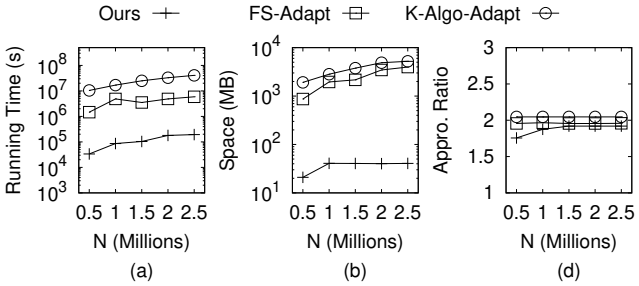

Figure 5: Effect of N (No. of Vertices on the Terrain Surface)

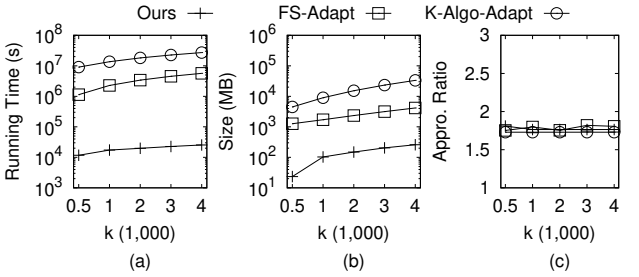

Figure 6: Effect of k

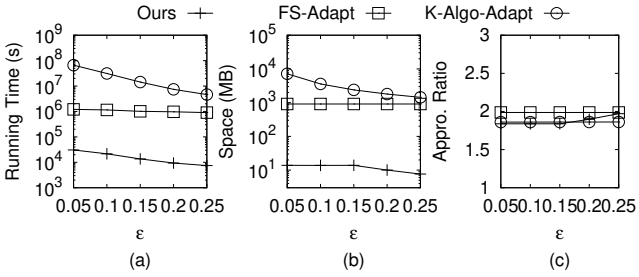

Figure 7: Effect of ϵ

(3) The actual approximation ratio of our algorithm was comparable to most of the other algorithms (which is around 2). In comparison, the actual approximation ratio of *NNA* was the worst. Note that *KS-Adapt* and *DIO-Adapt* had the smallest actual approximation ratio since they compute the exact pairwise geodesic distance for constructing the query graph G_Q at the expense of high time and space costs, as previously mentioned.

According to the observations above, we simplify the presentation in the remainder of this section for simplicity and clarity as follows: (1) Since the space consumption of DOA (including *SE-Oracle-Adapt* and *EAR-Oracle-Adapt*) and *US-Adapt* is too large to fit our limited memory budget, especially in the three largest datasets, we do not show their results in the subsequent experiments. (2) Since the running time of NNA, *DIO-Adapt*, and *KS-Adapt* is large and they cannot be finished in a reasonable time (i.e., a month) in the three largest datasets, we omit their results. In the following, we tested the sensitivity of our algorithm to the parameters.

Effect of N. To test the scalability of each algorithm considered in our experiment, we adopted five larger terrain datasets generated from *EP (H)*. Specifically, we first enlarged the size of *EP (H)* by an up-sampling method. The up-sampling method inserted a new vertex at the center of each face of *EP (H)* and introduced a new edge between the newly inserted vertex and each adjacent vertex of the face. Then, the face was divided into three faces. After that, we cropped five sub-regions of this enlarged *EP (H)* datasets, where the five sub-regions contained 0.5M, 1M, 1.5M, 2M, and 2.5M vertices, respectively. The results on the five datasets are shown in Figure 5.

As shown there, our algorithm scaled well to millions of vertices. Its running time was up to two orders of magnitude faster than that of the baseline methods, and its space consumption was significantly smaller than the baseline methods. Its actual approximation ratio also had a slight improvement. These findings were consistent with the theoretical counterpart in Table 2.

Effect of k. We studied the effect of the number k of POIs in P on the *SF (H)* dataset, by setting the values of k to be 500, 1000, 2000, 3000 and 4000, respectively. Figure 6 shows the results. As expected, the running time and space overhead of each algorithm monotonically increased with the increasing k. In contrast, the actual approximation ratio of each algorithm was not sensitive to the values of k. The observations above were also consistent with the theoretical analysis in Table 2. As the figure reveals, our algorithm outperformed the two baselines by 1-2 orders of magnitudes regarding running time and space overhead with nearly the same actual appro. ratio.

Effect of ϵ. We studied the effect of the error parameter ϵ on the *BH (H)* dataset. The values of ϵ considered in the experiment were 0.05, 0.1, 0.15, 0.2, 0.25. As shown, the performance of *FS-Adapt* was indifferent to the values of ϵ since it was independent of ϵ (see Table 2). In contrast, the running time and space overhead of *K-Algo-Adapt* and our algorithm monotonically decreased with the increasing ϵ since the more stringent accuracy requirement leads to heavier computational effort. Accordingly, the approximation ratios of *K-Algo-Adapt* and our algorithm monotonically increased with the increasing ϵ. Consistent with previous results, our algorithm beat the two baselines regarding running time and space overhead by up to 1-2 orders of magnitude with nearly the same appro. ratio.

Summary. In the experiment, we compared our algorithm with the adaptations of the existing algorithms for terrain surfaces. We studied the effect of N, the effect of ϵ, and the effect of k. We observe that (1) DOA (including *SE-Oracle-Adapt* and *SE-Oracle-Adapt*) all have a forbiddingly large space consumption; and (2) *NNA*, *DIO-Adapt*, *KS-Adapt*, *US-Adapt* all have a large running time, and each of them is not capable of scaling up to sizable datasets with millions of vertices. In comparison, our algorithm scales up to the million-scale datasets, demonstrating the best execution efficiency and the smallest space overhead. It outperforms all competitors by up to 1-2 orders of magnitude in terms of running time and space overhead with nearly the same or better actual approximation ratio.

6 Conclusion

In this paper, we study a fundamental and important problem called *Trip Planning on Terrain Surfaces (TPTS)*, which finds a plethora of applications in GIS, military tactical analysis, map services, etc. We prove that TPTS is NP-hard and propose an efficient approximation algorithm for TPTS. The running time, space consumption and approximation ratio of our algorithm are $O(\frac{k}{\epsilon^{2\beta}} N \log^2 N)$, $O(\frac{k}{\epsilon^{2\beta}})$ and $2 \cdot (1+\epsilon)$, respectively, where N is the number of vertices on the terrain surface, ϵ is a user-specified parameter, and β is a small constant within $[1.5, 2]$. Our empirical study demonstrates that our algorithm significantly outperforms all baselines regarding running

time and space consumption with the same or better accuracy.

One topic for future work is to study trip planning on a more general version of terrain surfaces called *weighted terrain surfaces*. On a weighted terrain surface, each face will be assigned a real-valued positive weight that encodes the property (i.e., slope, etc.) and captures the travel cost (i.e., travel time, energy consumption) of the face. The geodesic distance on a weighted terrain surface will get the weight information involved and is more challenging.

References

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

[2] 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. In *Journal of ACM*.

[3] Jindong Chen and Yijie Han. 1990. Shortest paths on a polyhedron. In *SoCG*.

[4] Ke Deng, Heng Tao Shen, Kai Xu, and Xuemin Lin. 2006. Surface k-NN query processing. In *ICDE*.

[5] 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)*.

[6] 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. In *VLDBJ*.

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

[8] Bo Huang, Victor Junqiu Wei, Raymond Chi-Wing Wong, and Bo Tang. 2023. EAR-Oracle: On Efficient Indexing for Distance Queries between Arbitrary Points on Terrain Surface. *SIGMOD* (2023).

[9] 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)*.

[10] Sanjiv Kapoor. 1999. Efficient computation of geodesic shortest paths. In *STOC*.

[11] Manohar Kaul, Raymond Chi-Wing Wong, and Christian S Jensen. 2015. New lower and upper bounds for shortest distance queries on terrains. In *VLDB*.

[12] 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 *VLDB*.

[13] Baki Koyuncu and Erkan Bostancı. 2009. 3D battlefield modeling and simulation of war games. In *Proceedings of the 3rd International Conference on Communications and information technology*.

[14] Joseph B Kruskal. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society* 7, 1 (1956), 48–50.

[15] Jean-François Lalonde, Nicolas Vandapel, Daniel F Huber, and Martial Hebert. 2006. Natural terrain classification using three-dimensional ladar data for ground robot mobility. In *Journal of field robotics*.

[16] Fumei Lam and Alantha Newman. 2008. Traveling salesman path problems. *Mathematical Programming* 113, 1 (2008), 39–59.

[17] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. In *Algorithmica*.

[18] Lian Liu and Raymond Chi-Wing Wong. 2011. Finding shortest path on land surface. In *SIGMOD*.

[19] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. In *SIAM Journal on Computing*.

[20] Christos H Papadimitriou. 1977. The Euclidean travelling salesman problem is NP-complete. *Theoretical computer science* 4, 3 (1977), 237–244.

[21] Louis V Quintas and Fred Supnick. 1965. On some properties of shortest Hamiltonian circuits. *The American Mathematical Monthly* 72, 9 (1965), 977–980.

[22] Jagan Sankaranarayanan and Hanan Samet. 2009. Distance oracles for spatial networks. In *Proceedings of the 25th IEEE International Conference on Data Engineering*.

[23] Jagan Sankaranarayanan and Hanan Samet. 2010. Query processing using distance oracles for spatial networks. *IEEE Transactions on Knowledge and Data Engineering*, (Best Papers of ICDE 2009 Special Issue.).

[24] 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. In *Personal and Ubiquitous Computing*.

[25] Cyrus Shahabi, Lu-An Tang, and Songhua Xing. 2008. Indexing land surface for efficient knn query. In *VLDB*.

[26] 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

[27] Nicolas Vandapel, Raghavendra Rao Donamukkala, and Martial Hebert. 2006. Unmanned ground vehicle navigation using aerial ladar data. In *The International Journal of Robotics Research*.

[28] Vishal Verma and Jack Snoeyink. 2009. Reducing the memory required to find a geodesic shortest path on a large mesh. In *SIGSPATIAL*.

[29] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, and David M. Mount. 2017. Distance Oracle on Terrain Surface. In *SIGMOD*.

[30] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, David M Mount, and Hanan Samet. 2022. Proximity Queries on Terrain Surface. *TODS* (2022).

[31] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, David M Mount, and Hanan Samet. 2024. On Efficient Shortest Path Computation on Terrain Surface: A Direction-Oriented Approach. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[32] Shi-Qing Xin and Guo-Jin Wang. 2009. Improving Chen and Han's Algorithm on the Discrete Geodesic Problem. In *TOG*.

[33] S. Xing, C. Shahabi, and B. Pan. 2009. Continuous monitoring of nearest neighbors on land surface. In *VLDB*.

[34] D. Yan, Z. Zhao, and W. Ng. 2012. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *CIKM*.

A Analysis of Baselines

A.1 Running Time and Space Overhead

Theorem 6. *The running time of* NNA *is* $O(k \cdot N^2 \log N)$ *and the space overhead of* NNA *is* $O(k+N)$.

Proof. *NNA* performs the nearest neighbor search algorithm [6] (each of which takes $O(N^2 \log N)$ time by [6]) for k times and performs the geodesic shortest path (each of which takes $O(N \log^2 N)$ by [8, 29, 30]). Thus, we obtain that the running time of NNA is $O(k \cdot N^2 \log N)$.

The space consumption of NNA includes the storage of the $P \cup \{s,t\}$ ($k+2$ points) and the terrain surface ($O(N)$ space complexity) and the storage of the final path ($O(N)$). Thus, we obtain that the space overhead of NNA is $O(k+N)$. □

Theorem 7. *The running time of* KS-Adapt, DIO-Adapt, FS-Adapt, US-Adapt *and* K-Algo-Adapt *are* $O(k^2 N \log^2(N))$, $O(k^2 N^2 \log(N))$, $k^2 N \log(N)$, $O(k^2 \cdot \frac{N}{\epsilon} \log(\frac{N}{\epsilon}) \log \frac{1}{\epsilon})$, *and* $O(k^2 \frac{N}{\epsilon} \log(\frac{N}{\epsilon}))$, *respectively. The space overhead of* KS-Adapt, DIO-Adapt, FS-Adapt, US-Adapt *and* K-Algo-Adapt *are all* $O(k^2+N)$.

Proof. The running time of each of these algorithms (*KS-Adapt*, *DIO-Adapt*, *FS-Adapt*, *US-Adapt* and *K-Algo-Adapt*) consists of (1) the building time of the query graph G_Q and (2) the TSWP path finding on the query graph G_Q. For the construction of the query graph G_Q (which is a complete graph and the vertices of G_Q is comprised of $P \cup \{s,t\}$), *KS-Adapt*, *DIO-Adapt*, *FS-Adapt*, *US-Adapt* and *K-Algo-Adapt* invoke *KS* ($O(N^2 \log N)$ time complexity), *DIO* ($O(N^2 \log N)$ time complexity), *FS* ($O(N \log N)$ time complexity), *US* ($O(\frac{N}{\epsilon} \log(\frac{N}{\epsilon}) \log \frac{1}{\epsilon})$ time complexity) and *K-Algo* ($O(\frac{N}{\epsilon} \log(\frac{N}{\epsilon})$ time complexity) to get the weight of each edge on G_Q, respectively. Thus, we obtain that the running time of the query graph building in *KS-Adapt*, *DIO-Adapt*, *FS-Adapt*, *US-Adapt* and *K-Algo-Adapt* are $O(k^2 N \log^2(N))$, $O(k^2 N^2 \log(N))$, $k^2 N \log(N)$, $O(k^2 \cdot \frac{N}{\epsilon} \log(\frac{N}{\epsilon}) \log \frac{1}{\epsilon})$, and $O(k^2 \frac{N}{\epsilon} \log(\frac{N}{\epsilon}))$, respectively. For the TSWP path finding, each of them applies the state-of-the-art approximation algorithm [16] (which is based on Integer Linear Programming) and the running time is $O(k)$. Thus, we obtain that the running time of *KS-Adapt*, *DIO-Adapt*, *FS-Adapt*, *US-Adapt* and *K-Algo-Adapt* are $O(k^2 N \log^2(N))$, $O(k^2 N^2 \log(N))$, $k^2 N \log(N)$, $O(k^2 \cdot \frac{N}{\epsilon} \log(\frac{N}{\epsilon}) \log \frac{1}{\epsilon})$, and $O(k^2 \frac{N}{\epsilon} \log(\frac{N}{\epsilon}))$, respectively.

The space overhead of each of these algorithms includes (1) the space consumption of the $P \cup \{s,t\}$ ($O(k)$ space) and the terrain surface ($O(N)$ space) and (2) the space consumption of the query graph ($O(k^2)$). Thus, we obtain that each of these algorithms has an $O(k^2+N)$ space overhead. □

Theorem 8. *The running time of* SP-Oracle-Adapt, SE-Oracle-Adapt *and* EAR-Oracle-Adapt *are* $O(\frac{N\log\frac{1}{\epsilon}}{\epsilon^{2.5}}\log^3(\frac{N\log\frac{1}{\epsilon}}{\epsilon^{1.5}})\log^2\frac{1}{\epsilon}+k^2(\frac{1}{\epsilon}\log\frac{1}{\epsilon}+\log\log(\frac{N}{\sqrt{\epsilon}})))$ $O(\frac{N\log^2 N}{\epsilon^{2\beta}}+\frac{Nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}}\cdot\log\frac{N\log\frac{1}{\epsilon}}{\sqrt{\epsilon}}+\frac{Nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}\cdot\epsilon^{2\beta}}+k^2h)$ *and* $O(\zeta N\log(N))+\frac{N\log(N)}{\epsilon^{2\beta}}+Nh\log(N)+\frac{Nh}{\epsilon^{2\beta}}+k^2\zeta\log(\zeta)$, *respectively.*

Proof. The running time of each of the three algorithms *SP-Oracle-Adapt*, *SE-Oracle-Adapt* and *EAR-Oracle-Adapt* consists of (1) the building time of the indexing structure (*SP-Oracle-Adapt*, *SE-Oracle-Adapt*, and *EAR-Oracle-Adapt*, respectively), (2) the building time of the query graph G_Q, and (3) the TSWP path finding on the query graph G_Q. By the results of [7, 8, 29], the building time of indexing structures *SP-Oracle-Adapt*, *SE-Oracle-Adapt* and *EAR-Oracle-Adapt* are $O(\frac{N\log\frac{1}{\epsilon}}{\epsilon^{2.5}}\log^3(\frac{N\log\frac{1}{\epsilon}}{\epsilon^{1.5}})\log^2\frac{1}{\epsilon})$ $O(\frac{N\log^2 N}{\epsilon^{2\beta}}+\frac{Nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}}\cdot\log\frac{N\log\frac{1}{\epsilon}}{\sqrt{\epsilon}}+\frac{Nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}\cdot\epsilon^{2\beta}})$ and $O(\zeta N\log(N))+\frac{N\log(N)}{\epsilon^{2\beta}}+Nh\log(N)+\frac{Nh}{\epsilon^{2\beta}}$, respectively. For the construction of the query graph G_Q (which is a complete graph and the vertices of G_Q is comprised of $P\cup\{s,t\}$), *SP-Oracle-Adapt*, *SE-Oracle-Adapt* and *EAR-Oracle-Adapt* invoke the distance query processing algorithm of *SP-Oracle*, *SE-Oracle* and *EAR-Oracle* to get the weight of each edge on G_Q, respectively. The query time of *SP-Oracle*, *SE-Oracle* and *SP-Oracle* is $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon}+\log\log(\frac{N}{\sqrt{\epsilon}}))$, $O(h)$ and $O(\zeta\log(\zeta))$. Thus, we obtain that the building time of the query graph in *SP-Oracle-Adapt*, *SE-Oracle-Adapt* and *EAR-Oracle-Adapt* are $O(k^2(\frac{1}{\epsilon}\log\frac{1}{\epsilon}+\log\log(\frac{N}{\sqrt{\epsilon}})))$ $O(k^2h)$ and $O(k^2\zeta\log(\zeta))$, respectively. For the TSWP path finding, each of them applies the state-of-the-art approximation algorithm [16] (which is based on Integer Linear Programming) and the running time is $O(k)$.

Thus, we finally obtain that the running time of *SP-Oracle-Adapt*, *SE-Oracle-Adapt* and *EAR-Oracle-Adapt* are $O(\frac{N\log\frac{1}{\epsilon}}{\epsilon^{2.5}}\log^3(\frac{N\log\frac{1}{\epsilon}}{\epsilon^{1.5}})\log^2\frac{1}{\epsilon}+k^2(\frac{1}{\epsilon}\log\frac{1}{\epsilon}+\log\log(\frac{N}{\sqrt{\epsilon}})))$ $O(\frac{N\log^2 N}{\epsilon^{2\beta}}+\frac{Nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}}\cdot\log\frac{N\log\frac{1}{\epsilon}}{\sqrt{\epsilon}}+\frac{Nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}\cdot\epsilon^{2\beta}}+k^2h)$ and $O(\zeta N\log(N))+\frac{N\log(N)}{\epsilon^{2\beta}}+Nh\log(N)+\frac{Nh}{\epsilon^{2\beta}}+k^2\zeta\log(\zeta)$, respectively. □

Theorem 9. *The space overhead of* SP-Oracle-Adapt, SE-Oracle-Adapt *and* EAR-Oracle-Adapt *are* $O(\frac{N\log\frac{1}{\epsilon}}{\epsilon^2}\log^2\frac{N\log\frac{1}{\epsilon}}{\epsilon^{1.5}}\log^2\frac{1}{\epsilon}+k^2)$, $O(\frac{Nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}\cdot\epsilon^{2\beta}}+k^2)$ *and* $O(\frac{N}{\zeta}+\frac{Nh}{\epsilon^{2\beta}}+k^2)$, *respectively.*

Proof. The space overhead of each of the algorithms (*SP-Oracle-Adapt*, *SE-Oracle-Adapt* and *EAR-Oracle-Adapt*) consists of (1) the space overhead of the indexing structures (*SP-Oracle*, *SE-Oracle* and *EAR-Oracle*, respectively) and (2) the space consumption of query graph G_Q. According to the results of [7, 8, 29], the space overhead of *SP-Oracle*, *SE-Oracle* and *EAR-Oracle* are $O(\frac{N\log\frac{1}{\epsilon}}{\epsilon^2}\log^2\frac{N\log\frac{1}{\epsilon}}{\epsilon^{1.5}}\log^2\frac{1}{\epsilon})$, $O(\frac{Nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}\cdot\epsilon^{2\beta}})$ and $O(\frac{N}{\zeta}+\frac{Nh}{\epsilon^{2\beta}})$, respectively. The query graph G_Q takes $O(k^2)$ space. Thus, we finally obtain that the space overhead of *SP-Oracle-Adapt*, *SE-Oracle-Adapt* and *EAR-Oracle-Adapt* are $O(\frac{N\log\frac{1}{\epsilon}}{\epsilon^2}\log^2\frac{N\log\frac{1}{\epsilon}}{\epsilon^{1.5}}\log^2\frac{1}{\epsilon}+k^2)$, $O(\frac{Nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}\cdot\epsilon^{2\beta}}+k^2)$ and $O(\frac{N}{\zeta}+\frac{Nh}{\epsilon^{2\beta}}+k^2)$, respectively. □

A.2 Approximation Ratio

Theorem 10. *The approximation ratio of* KS-Adapt *and* DIO-Adapt *is* $\frac{3}{2}$.

Proof. In *KS-Adapt* and *DIO-Adapt*, all pairwise distances between each pair of points in $P\cup\{s,t\}$ are computed through the construction of the query graph G_Q. The length l^* of the shortest path from s to t passing through all points in P on the query graph G_Q is equal to the length l of the shortest geodesic path $\Pi_{G_Q}(s,t)$ from s to t passing through all points in P on the terrain surface. Then, each of them invokes the approximation TSWP algorithm to find the approximate shortest path $\pi_{G_Q}(s,t)$ from s to t passing through all points in P on the graph G_Q. Since for each edge (u,v) on G_Q, the weight of the edge (u,v) is the geodesic distance $d_g(u,v)$ between u and v. We obtain that the length l' of the path $\pi_{G_Q}(s,t)$ is equal to the length l_{ALG} of the path found on the terrain surface by *KS-Adapt* and *DIO-Adapt*. Since the approximation TSWP algorithm has an approximation ratio $\frac{3}{2}$, we obtain that $\frac{l'}{l}\le\frac{3}{2}$ (that is $\frac{l_{ALG}}{l^*}\le\frac{3}{2}$). Thus, the approximation ratio of *KS-Adapt* and *DIO-Adapt* is $\frac{3}{2}$. □

Theorem 11. *The approximation ratio of* FS-Adapt *is* $\frac{1+\delta}{1-\delta}\cdot\frac{3}{2}$.

Proof. In *FS-Adapt*, all pairwise ϵ-approximate distances between each pair of points in $P\cup\{s,t\}$ are computed through the construction of the query graph G_Q. The length l^* of the shortest path from s to t passing through all points in P on the query graph G_Q is at most $1+\delta$ times larger than the length l of the shortest geodesic path $\Pi_{G_Q}(s,t)$ from s to t passing through all points in P on the terrain surface. Then, each of them invokes the approximation TSWP algorithm to find the approximate shortest path $\pi_{G_Q}(s,t)$ from s to t passing through all points in P on the graph G_Q. Since for each edge (u,v) on G_Q, the weight of the edge (u,v) is at most $1+\delta$ times larger than the geodesic distance $d_g(u,v)$ between u and v. We obtain that the length l' of the path $\pi_{G_Q}(s,t)$ is equal to the length l_{ALG} of the path found on the terrain surface by *FS-Adapt*. Since the approximation TSWP algorithm has an approximation ratio $\frac{3}{2}$, we obtain that $\frac{l'}{l}\le\frac{3}{2}$. Since $l_{ALG}\le(1+\delta)\cdot l'$) and $l^*\ge(1-\delta)\cdot l$, we obtain that $\frac{l_{ALG}}{l^*}\le\frac{1+\delta}{1-\delta}\cdot\frac{3}{2}$. Thus, the approximation ratio of *FS-Adapt* is $\frac{1+\delta}{1-\delta}\cdot\frac{3}{2}$. □

Theorem 12. *The approximation ratio of* US-Adapt, K-Algo-Adapt, SP-Oracle-Adapt, SE-Oracle-Adapt, EAR-Oracle-Adapt *is* $\frac{1+\epsilon}{1-\epsilon}\cdot\frac{3}{2}$.

Proof. In *US-Adapt*, *K-Algo-Adapt*, *SP-Oracle-Adapt*, *SE-Oracle-Adapt*, *EAR-Oracle-Adapt*, all pairwise ϵ-approximate distances between each pair of points in $P\cup\{s,t\}$ are computed through the construction of the query graph G_Q. The length l^* of the shortest path from s to t passing through all points in P on the query graph G_Q is at most $1+\epsilon$ times larger than the length l of the shortest ge-

odesic path $\Pi_{G_Q}(s,t)$ from s to t passing through all points in P on the terrain surface. Then, each of them invokes the approximation TSWP algorithm to find the approximate shortest path $\pi_{G_Q}(s,t)$ from s to t passing through all points in P on the graph G_Q. Since for each edge (u,v) on G_Q, the weight of the edge (u,v) is at most $1+\epsilon$ times larger than the geodesic distance $d_g(u,v)$ between u and v. We obtain that the length l' of the path $\pi_{G_Q}(s,t)$ is equal to the length l_{ALG} of the path found on the terrain surface by *US-Adapt*, *K-Algo-Adapt*, *SP-Oracle-Adapt*, *SE-Oracle-Adapt*, *EAR-Oracle-Adapt*. Since the approximation TSWP algorithm has an approximation ratio $\frac{3}{2}$, we obtain that $\frac{l'}{l} \leq \frac{3}{2}$. Since $l_{ALG} \leq (1+\epsilon)\cdot l'$ and $l^* \geq (1-\epsilon)\cdot l$, we obtain that $\frac{l_{ALG}}{l^*} \leq \frac{1+\epsilon}{1-\epsilon}\cdot\frac{3}{2}$. Thus, the approximation ratio of *US-Adapt*, *K-Algo-Adapt*, *SP-Oracle-Adapt*, *SE-Oracle-Adapt*, *EAR-Oracle-Adapt* is $\frac{1+\epsilon}{1-\epsilon}\cdot\frac{3}{2}$. □