Constrained Shortest Path Finding on Terrain Surfaces

VICTOR JUNQIU WEI, Macau University of Science and Technology, Macau SAR, China
MIN XIE, Shenzhen Institute of Computing Sciences, Shenzhen, China
WEICHENG WANG*, Beijing Institute of Technology, Zhuhai, China

With the advancement of geo-positioning technologies, the terrain surface has become more and more popular and has drawn a lot of attention from academia and industry. In this paper, we propose a fundamental problem, namely *Constrained Shortest Path Finding on Terrain Surfaces (CPTS)*. Given a source point s, a destination point t, and a set P of point-of-interests on the terrain surface, the goal is to find the shortest s-t path passing through all points in P on the terrain surface. This problem finds a plethora of applications in map services, military vehicle path planning, scientific studies, etc.

We prove that CPTS is an NP-hard problem. Due to its hardness, we then develop an approximation algorithm for the 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 $O(\frac{k \cdot N \log^2 N}{\epsilon^{2\beta}})$ 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 a nearly identical or better approximation ratio.

CCS Concepts: • **Information systems → Proximity search**; **Geographic information systems**.

Additional Key Words and Phrases: Shortest Path, Terrain Surfaces, Geographical Information System

ACM Reference Format:
Victor Junqiu Wei, Min Xie, and Weicheng Wang. 2026. Constrained Shortest Path Finding on Terrain Surfaces. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 154 (June 2026), 24 pages. https://doi.org/10.1145/3802031

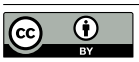
1 Introduction

Digital terrain surfaces have emerged as an important data object and attracted much attention from both academia and industry [5, 7, 9, 11–13, 20, 27, 30–33, 36, 37] 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 an instance of the terrain surface. As shown therein, 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

*Weicheng Wang is the corresponding author.

Authors' Contact Information: Victor Junqiu Wei, wjqjsnj@gmail.com, Macau University of Science and Technology, Macau SAR, China; Min Xie, xiemin@sics.ac.cn, Shenzhen Institute of Computing Sciences, Shenzhen, China; Weicheng Wang, wangweic@bit.edu.cn, Beijing Institute of Technology, Zhuhai, China.



ACM 2836-6573/2026/6-ART154
https://doi.org/10.1145/3802031

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 [5]).

In this paper, we propose a fundamental and important problem on terrain surfaces called *Constrained Shortest Path Finding on Terrain Surfaces (CPTS)*. CPTS 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 $\Pi_g(s, t|P)$. CPTS is a fundamental problem on terrain surfaces and has wide applications in Geographic Information Systems (GIS), military routing, etc, where the visiting of plenty of points-of-interest has to be carefully planned.

(1) In Geographical Information Systems (GIS), unmanned vehicles (e.g., Google Map camera cars) are often deployed in regions with complex topography, including steep slopes, rugged ridges, and urban overpasses, etc., to collect data. Each vehicle has a starting point s and a destination point t, and data must be collected at predefined locations. In the study of [16, 29], there are roughly 300-2000 pre-defined locations to be visited. Vehicles need the route planning to find an optimal path that is drivable (i.e., constrained by the terrain surface) and passes through all locations with the smallest total length to save energy and time.
(2) In military operations, troops need to navigate through regions with challenging terrain surfaces, such as steep slopes, exposed sides, narrow mountain passes, etc. For the delivery or rescue missions, they typically require moving from a base location s to a target location t, while visiting several locations to deliver supplies or perform tactical tasks [14]. The route planning in this context must strictly conform to the terrain surface to pass through all the locations and minimize the traveling cost.
(3) In geography research, scientists normally send an autonomous unmanned vehicle (AUV) [1, 22] to a set of (around 500-1000) points on terrain surfaces. The terrain surface may include rugged slopes, ravines, or undulating geological structures, and the points are uniformly sampled or specified by the scientists on the fly. At each specified point, the AUV collects data or samples regarding the physical and chemical properties (e.g., geomagnetic strength, temperature, oxygen concentration, the ingredients of the soil/minerals, etc.) of the ground and soil there. The data collected will be utilized to obtain the distribution of the properties studied, which are normally visualized as a heat map or contour lines. The source point of the trip is the current location of the AUV and the destination point is the place where the data or samples should be stored. The path from the source point to the destination point passing through all specified points is the one with the best energy efficiency.

Note that in many mountainous wilderness areas mentioned in the above applications, there are typically no road networks. As a result, the geodesic distance becomes the best metric for measuring distances. This naturally leads to the constrained shortest path problem on the terrain surface, which is significant in these scenarios. However, to the best of our knowledge, our work is the first one to investigate this problem systematically. In the subsequent sections, we prove that this problem is NP-hard and highly intractable.

Despite that constrained shortest path problem has been investigated in Euclidean space [17, 19], CPTS is fundamentally different and encounters unique challenges in the context of terrain surfaces.

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 most fundamental operation *distance computation* in Euclidean space only involves two points, and thus, takes constant time (which is cheap). However, for terrain

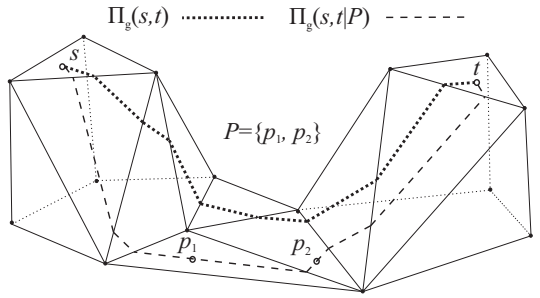


Fig. 1. An Example of Constrained Shortest Path on Terrain Surface

surfaces, the shortest path must conform to the surface topology, which may have a complex irregular structure due to the presence of ridges, cliffs, abrupt elevation, etc. As a result, the shortest path between two points deviates significantly from a straight line, and it needs $O(N \log^2 N)$ to compute. This increased complexity makes it infeasible to simply adapt existing Euclidean-based algorithms to the terrain scenario. Existing algorithms rely heavily on precomputing all pairwise distances among the points in $P \cup \{s, t\}$, which incurs $O(k^2)$ space consumption and a running time quadratic to k. When it comes to the terrain surface, distance computation is expensive, and thus, precomputing all pairwise distances becomes prohibitive. In Section 4.2, we will show how naive adaptations of existing algorithms suffer from poor performance, being either inefficient, space-consuming or inaccurate.

In light of this, we propose an efficient approximation algorithm for CPTS. We first construct a connected, lightweight, and sparse graph $\mathcal{G}$ called *spanning graph* for all points in $P \cup \{s, t\}$, whose vertices are comprised of all points in $P \cup \{s, t\}$ with only $O(k)$ edges. This spanning graph $\mathcal{G}$ provides the ϵ-approximate geodesic distance between any pair of points in $P \cup \{s, t\}$ with only $O(k)$ space, which effectively reduces the space overhead as required by existing algorithms. Then, based upon this, we propose a strategy 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. As can be noticed, our algorithm avoids the pre-computation of all pairwise geodesic distances, which is required by existing algorithms and leads to a significant cost saving. The theoretical analysis shows that our 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. Differing from existing adaptions, we aim to strike a balance among the running time, space overhead, and approximation ratio, providing both theoretical ground and an empirically effective solution. Our contributions are summarized as follows.

- We study a novel problem on terrain surfaces, *Constrained Shortest Path Finding on Terrain Surfaces (CPTS)*, which finds a plethora of applications in GIS, map services, military tactical analysis, etc. We also prove the NP-hardness of the problem.
- We propose an efficient approximation algorithm for CPTS. 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 $2 \cdot (1+\epsilon)$. In theory, it beats all competitors in terms of running time, space overhead, and accuracy.
- We conducted extensive experiments on several real datasets. The results show that our proposed algorithm significantly beats all competitors by orders of magnitude in terms of running time and space overhead with the same or better accuracy.

Notation	Meaning
T, V, E, F	The terrain surface, Vertices, Edges and Faces.
N	The number of vertices on T.
s, t	Two arbitrary points on T.
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 T.
$\Pi_g(s,t)$	The shortest path from s to t on T.
$\Pi_g(s,t\|p)$	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 T 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 remainder of the paper is organized as follows. Section 2 formally defines the problem of constrained shortest path finding on terrain surfaces (CPTS) and discusses its NP-hardness. Section 3 presents the algorithm as well as its theoretical analysis. Section 4 reviews existing studies and baselines, and compares them against our algorithm. Section 5 reports the experiments. Finally, Section 6 summarizes this work.

2 Problem Definition

Consider a terrain surface T. Let V, E, and F denote the sets of all vertices, edges, and faces on T, respectively. The complexity of the terrain surface T is represented by the total number of vertices, denoted by N (i.e., $N = |V|$). Each vertex $v \in V$ is a 3D point with coordinates x_v, y_v, and z_v. We also refer to any arbitrary location on T, simply as *point*, which may or may not be a vertex. For instance, in Figure 1, each solid point is a vertex, and each solid line segment is an edge. There are two points s and t on the terrain surface, but neither is a vertex. Frequently used notations are shown in Table 1.

Consider two points s and t on T. A path, denoted by $\pi_g(s,t)$, from s to t on T 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 shares 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 these concepts, we formally define the *geodesic shortest path*.

Definition 2.1 (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.

In Figure 1, the geodesic shortest path between s and t is shown in dotted line segments. The length of $\Pi_g(s,t)$, denoted by $d_g(s,t)$, is called the *geodesic distance* between s and t. Building upon the definition of the geodesic shortest path for two given points, we extend it to *trips* that involve multiple points-of-interest.

Definition 2.2 (Geodesic Shortest Trip Path). Given a terrain surface T, two points s and t on T, and a set P of points on T, the *geodesic shortest trip path*, denoted by $\Pi_g(s,t|P)$, is the geodesic shortest path from s to t on T 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 (Constrained Shortest Path Finding on Terrain Surfaces (CPTS)). *Given a terrain surface T, two points s and t on T, and a set P of points-of-interest (POIs) on T, the CPTS problem is to find the geodesic shortest trip path $\Pi_g(s, t|P)$.*

Intractability. Unfortunately, the CPTS problem is NP-hard.

Theorem 2.3 (Hardness). *Problem CPTS is NP-hard.*

Proof. We prove that CPTS is NP-hard by transforming the renowned *Euclidean Travelling Salesman Problem (ETSP)* [24] (which has been proved to be NP-hard [23]) into CPTS.

We first present the decision version of the CPTS 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 CPTS 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 CPTS 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 CPTS 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 CPTS 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 CPTS problem instance are equivalent. □

3 Our Proposed Algorithm

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

Overview. Taken a terrain surface T, a source point s, a destination point t, a set P of points-of-interest (POIs), and an error parameter ϵ as input, our algorithm returns the shortest path $\Pi_g(s, t|P)$. Here ϵ is a user-specified parameter in the range of $[0, 1]$ for the trade-off between the efficiency and the accuracy (to be discussed later).

Our algorithm works in the following two steps.

(1) Spanning Graph Construction. The core component of our algorithm is a sparse and light-weight spanning graph denoted by $\mathcal{G}$, whose vertices are comprised of all points in $P \cup \{s, t\}$ with only $O(k)$ edges, so that $\mathcal{G}$ concisely captures the distance relationship between points in $P \cup \{s, t\}$. In particular, given any two 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) Path Finding. Once we have a spanning graph $\mathcal{G}$ containing s, t and all points in P, it will be very natural to find the visiting order of the points in $P \cup \{s, t\}$ through a path finding on $\mathcal{G}$. Note that if we adopt other algorithms, e.g., well-separated pair decomposition, it will only provide a

decomposition. It is still an open question how to find the visiting order of the points in $P \cup \{s, t\}$. Worse still, to generate the well-separated pair decomposition, it is necessary to build a tree-like structure (e.g., partition tree [31]), which may incur more time and space overhead. In light of this, in the second step, we order the vertices based on the spanning graph and make sure that s (resp. 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 $2 \cdot (1 + \epsilon)$.

Section 3.1 and Section 3.2 present the details of the two steps, respectively. Section 3.3 gives the theoretical analysis.

3.1 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, where the geodesic distance $d_g(p, q)$ is already known, we consider another point $q' \in P \cup \{s, t\}$ on the disk $D(q, r)$. Intuitively, if q' is in $D(q, r)$ and $d_g(p, q)$ is large enough compared with r, the geodesic distance between p and q' will not deviate too much from the known geodesic distance between p and q (i.e., the center based on which the disk $D(q, r)$ is constructed), since q' is very "close" to q. In this case, we can roughly estimate the geodesic distance between p and q' based on the geodesic distance between p and q.

Algorithm. We develop our algorithm for constructing the spanning graph $\mathcal{G}$ in Algorithm 1. In a high level, instead of computing geodesic distances between all point pairs, based on the intuition above, it leverages a small number of exact geodesic distances and local disks on the terrain surface to concisely capture the distance relationships between the points in $P \cup \{s, t\}$.

We first 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 2-13). The goal is to utilize the aforementioned idea to group points in $P \cup \{s, t\} \setminus \{p\}$ based on the disk concept, and 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 do this, we initialize a set $\mathcal{X}$ to be $P \cup \{s, t\} \setminus \{p\}$ (Line 3) and execute a while-loop until $\mathcal{X}$ is empty (Lines 4-13).

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 $d_g(p, q)$ if it does not already exist (Lines 7-9) and set the radius r to be $d_g(p, q)/\sigma$, where σ is $2+2/\epsilon$ (Line 10). The reason for this setting of r will be discussed in Section 3.3. After that, a set O is constructed (Lines 11) 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 12-13). 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.

For each point in $P \cup \{s, t\}$, we establish its incident edges. Finally, we return the graph $\mathcal{G}$ constructed on $P \cup \{s, t\}$ (Line 14).

Example 3.1 (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

Algorithm 1: Spanning Graph Construction SG(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 **if** *there does not exist an edge* (p, q) *in* $\mathcal{G}$ **then**
8 Add a new edge (p, q) into $\mathcal{G}$;
9 Set the weight of edge (p, q) to be $d_g(p, q)$;
10 $\sigma \leftarrow \frac{2}{\epsilon} + 2, r \leftarrow \frac{d_g(p,q)}{\sigma}$;
11 $O \leftarrow$ the set of points in $\mathcal{X}$ within $D(q, r)$;
12 **for** *each point* q' *in* O **do**
13 Remove q' from $\mathcal{X}$;
14 **return** $\mathcal{G}$;

Algorithm 2: CPTS Algorithm

Input: A set P of POIs, a source point s, a destination point t, a parameter ϵ, and a terrain surface T

Output: The geodesic shortest trip path $\Pi_g(s, t|P)$

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

(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 2-13 of Algorithm 1. 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 4-13 of Algorithm 1 as follows.

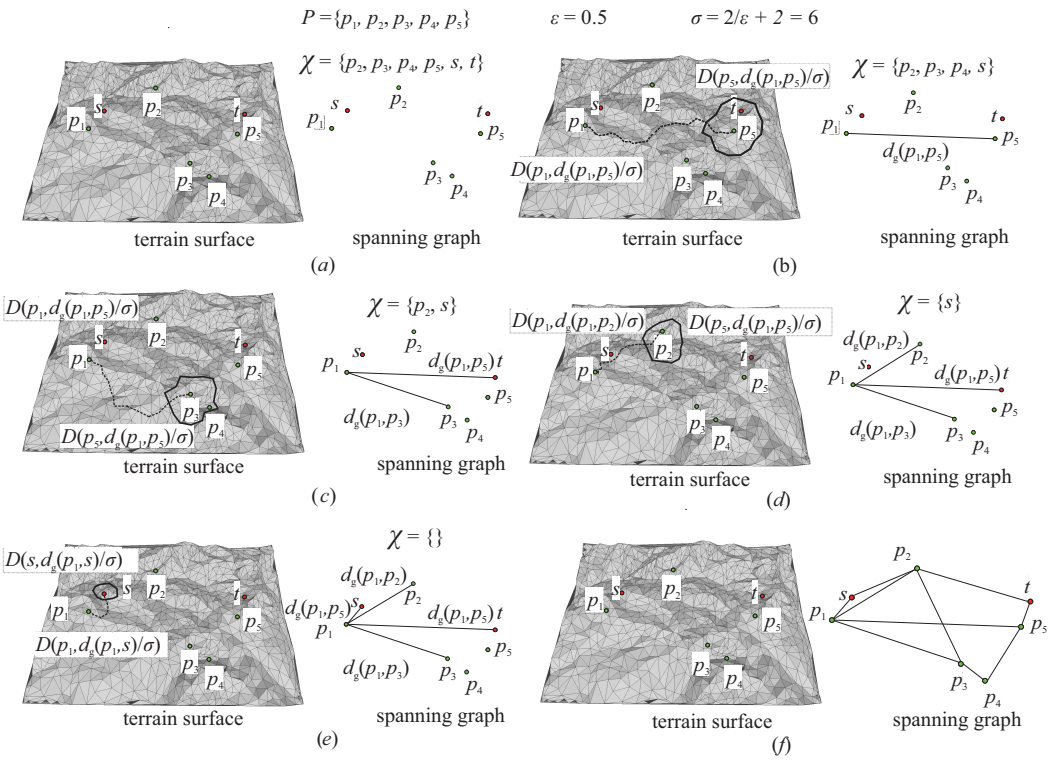


Fig. 2. An Example of Spanning Graph Construction

Figure 2 (b) shows the operations in the first iteration, 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, 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 1 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 1, 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 analysis of [11, 31, 32], 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 10-11 of Algorithm 1, we perform an existing index-free exact geodesic distance query processing algorithm to find the

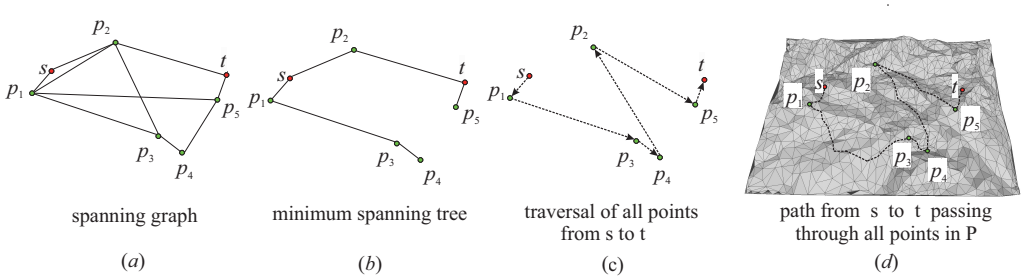


Fig. 3. An Example of Minimum Spanning Tree-Based Path Finding

points in $P \cup \{s, t\}$ within a given disk $D(c, r)$. 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. According to the theoretical analysis of [11, 31, 32], this online computation of the points within a given disk takes $O(N \log^2 N)$ time.

3.2 Constrained Shortest Path Finding on Terrain Surfaces (CPTS) Algorithm

Based on the spanning graph $\mathcal{G}$, the second step takes a source s on $\mathcal{G}$, a destination t on $\mathcal{G}$, a set P of POIs and the terrain surface T as input and returns the geodesic shortest trip path $\Pi_g(s, t|P)$.

Algorithm. As outlined in Algorithm 2, we first obtain the spanning graph $\mathcal{G}$ via Algorithm 1 and find the minimum spanning tree $\mathcal{T}$ of $\mathcal{G}$ (i.e., the spanning tree that has the minimum weight among all possible spanning trees) rooted at source s via the renowned *Kruskal's algorithm* [15] (Line 1-2). Next, we sort all vertices on the minimum spanning tree $\mathcal{T}$ by the pre-order traversal (Line 3). Let $\mathcal{I}$ denote this sorted list of vertices. We swap destination t with the last vertex in $\mathcal{I}$ (Lines 4-5) to ensure t is the last to be visited. Then, we enter a for-loop (Lines 6-9), 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 according to [11, 31, 32]). Finally, we return as the output the concatenation of all these geodesic shortest paths found (Line 10).

Example 3.2 (Minimum Spanning Tree-Based Path Finding). Figure 3 gives an example of our minimum spanning tree-based path finding algorithm. Based on the spanning graph $\mathcal{G}$ obtained by Algorithm 1 in Example 3.1 (Figure 3 (a)), we compute the minimum spanning tree of $\mathcal{G}$, as shown in Figure 3 (b). 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.

3.3 Theoretical Analysis

This section presents our theoretical analysis on the spanning graph (Theorems 3.3-3.8), the running time, space overhead (Theorem 3.11), and our approximation ratio (Theorem 3.12).

Theorem 3.3 (**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 $\mathcal{G}$.*

Proof. We first prove the following lemma.

Lemma 3.4. *For any pair of distinct vertices u and v on $\mathcal{G}$, there is an edge (u,x) in $\mathcal{G}$ such that $d_g(x,v) \leq \frac{d_g(u,x)}{\sigma}$.*

Proof. We prove it by contradiction. Assume that there is no edge (u,x) in $\mathcal{G}$ such that $d_g(x,v) \leq \frac{d_g(u,x)}{\sigma}$. 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 disk $D(y, \frac{d_g(u,y)}{\sigma})$). Consider the iteration in the for-loop in Lines 2-12 in Algorithm 1, where the point u is selected (i.e., $p = u$ in this iteration). By Line 3 and Lines 11-12 of Algorithm 1, $\mathcal{X}$ 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 1 (i.e., when this iteration of the for-loop in Algorithm 1 terminates, $\mathcal{X}$ must be $\emptyset$). □

By Lemma 3.4, 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 $d_g(x,v) \leq \frac{d_g(u,x)}{\sigma}$. 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 3.4, 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. *In 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.*

Lemma 3.5 is for the concept of maximum weight of L.

Lemma 3.5. *The maximum weight w_L^i of 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 3.4, 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 3.4, 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 $d_g(o,o') \geq d_g(o,t') - d_g(o',t') \geq \sigma \cdot d_g(o',t') - d_g(o',t')$. Then, we further obtain that $d_g(o',t') \leq \frac{1}{\sigma-1} \cdot d_g(o,o')$. Since $d_g(o,o')$ is at most $\frac{d_g(u,v)}{(\sigma-1)^{i-1}}$, we obtain that $d_g(o',t') \leq \frac{1}{\sigma-1} \cdot d_g(o,o') \leq \frac{d_g(u,v)}{(\sigma-1)^i}$. □

Lemma 3.6. *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 3.5, 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 3.6, 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 $d_{\mathcal{G}}(u,v) \leq (1+\epsilon) \cdot d_g(u,v)$. Consider the list L maintained by our aforementioned iterative procedure.

Lemma 3.7. *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 3.5, we obtain that $d_g(o,o')$ is at most $d_g(u,v)(\frac{1}{\sigma-1})^{i-1}$. Then, by Lemma 3.4, we can find the edge $< o, t' >$ in $\mathcal{G}$ such that o' is in the disk $D(t', \frac{d_g(o,t')}{\sigma})$. 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 $\frac{d_g(o,t')}{\sigma}$. By 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 $d_g(o,t') \leq \frac{\sigma}{\sigma-1} \cdot d_g(o,o') \leq d_g(u,v) \cdot \sigma \frac{1}{(\sigma-1)^i}$. □

Thus, the length of the path found by our iterative procedure is $\sum_{i=1}^{k} \frac{\sigma \cdot d_g(u,v)}{(\sigma-1)^i} = \sum_{i=1}^{k} \frac{(\frac{2}{\epsilon}+2) \cdot d_g(u,v)}{(\frac{2}{\epsilon}+1)^i} = \sum_{i=1}^{k} ((\frac{\epsilon}{2+\epsilon})^i) \cdot (\frac{2}{\epsilon}+2) \cdot d_g(u,v) = \frac{\epsilon}{2+\epsilon} \cdot \frac{1-(\frac{\epsilon}{2+\epsilon})^k}{1-\frac{\epsilon}{2+\epsilon}} \cdot (\frac{2}{\epsilon}+2) \cdot d_g(u,v) \leq \frac{\epsilon}{2+\epsilon} \cdot \frac{1}{1-\frac{\epsilon}{2+\epsilon}} \cdot (\frac{2}{\epsilon}+2) \cdot d_g(u,v) = (1+\epsilon) \cdot d_g(u,v)$. □

We proceed to introduce a parameter β called the *largest capacity dimension* of the terrain surface, a concept originally proposed in [31]. Due to limited space, we simply provide a high level intuition of β and refer the interested readers to [31]. The *largest capacity dimension* β of a set P of points on a terrain surface is defined as the maximum positive real number such that for any point p in P and any non-negative radius r, the disk $D(p,r)$ intersects with at most $2 \cdot 2^{2\beta}$ disjoint

disks each having radius at least $\frac{r}{4}$; in 2D Euclidean space, a disk of radius r can overlap with at most 12 disjoint disks of radius $\frac{r}{4}$, which leads to the conclusion that when the terrain surface is a perfectly flat 2D plane, β is at most 1.3, while with terrain surfaces exhibiting height fluctuations, β is typically larger than 1.3 due to the additional geometric complexity introduced by surface variations. Their experimental results [31] also confirmed that the *largest capacity dimension* β of the terrain surfaces considered in their experiment is between 1.3 and 1.5. In general cases, β can be a little bit larger, since the terrain surface could be regarded as a 2D surface with some fluctuations in terms of height. Thus, we round this range up and assume β is a small constant in [1.5, 2]. Next, we analyze the number of edges on the spanning graph.

Theorem 3.8 (**The Number of Edges on 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 1, 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, $N_u^i = \{v|v \in N_u, l_{min} \cdot 2^{i-1} \leq d_g(u, v) \leq l_{min} \cdot 2^i\}$). Consider the i-th group N_u^i.

Lemma 3.9. *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 by contradiction. Assume there is 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 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 1, 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! □

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 3.9, 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 of [31], 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 3.10. *The number of edges created in each iteration in the for-loop in Lines 2-12 of Algorithm 1 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 [31]. $\log \alpha$ is at

most 30 according to the experiments of [31]. 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. □

Theorem 3.11 (**Running Time and Space Overhead**). *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 1. In Algorithm 1, 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 3.10, 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 1 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 3.8 and the cardinality of $\mathcal{X}$ is at most k, we obtain that the space consumption of Algorithm 1 is $O(\frac{k}{\epsilon^{2\beta}}+N)$.

Consider Algorithm 2. We adopt the renowned *Kruskal's Algorithm* [15] to find the minimum spanning tree. According to its results, the running time of Line 1 in Algorithm 2 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 [31, 32]) by $k-1$ times. Thus, Lines 6-8 takes $O(k\cdot N\log^2 N)$ time. Since MST takes $O(k)$ space, the space overhead of Algorithm 2 is $O(k+N)$. □

Theorem 3.12 (**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 2. 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 2 is at most twice of $W(T)$. Thus, we obtain that the length of the path $\pi(s,t|P)$ found by Algorithm 2 is at most twice of the length of the path $\Pi'_{\mathcal{G}}(s,t|P)$. That is,

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

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 3.3, we obtain $\Pi_g(s,t|P)|=\sum_{i=1}^{k+1} d_g(o_i,o_{i+1})\ge\sum_{i=1}^{k+1}\frac{d_{\mathcal{G}}(o_i,o_{i+1})}{1+\epsilon}=\frac{1}{1+\epsilon}\sum_{i=1}^{k+1} d_{\mathcal{G}}(o_i,o_{i+1})$, where $d_{\mathcal{G}}(o_i,o_{i+1})$ is the shortest distance between o_i and o_{i+1} on $\mathcal{G}$. Together with the definition of $\Pi'_{\mathcal{G}}(s,t|P)$, we obtain that $|\Pi'_{\mathcal{G}}(s,t|P)|\le\sum_{i=1}^{k+1} d_{\mathcal{G}}(o_i,o_{i+1})\le(1+\epsilon)\cdot|\Pi_g(s,t|P)|$. Together with Inequality (1), we obtain that $|\pi(s,t|P)|\le 2\cdot|\Pi'_{\mathcal{G}}(s,t|P)|\le 2\cdot(1+\epsilon)\cdot|\Pi_g(s,t|P)|$. That is, the approximation ratio $\frac{|\pi(s,t|P)|}{|\Pi_g(s,t|P)|}$ is at most $2\cdot(1+\epsilon)$. □

Algorithm	Running Time	Space Overhead	Appro. Ratio
TPES [19]	$O(k^2+N)$	$O(k^2+N)$	$\frac{3}{2}\cdot\gamma$
NNA	$O(k\cdot N^2\log N)$	$O(k+N)$	k
KS-Adapt [11]	$O(k^2N\log^2(N))$	$O(k^2+N)$	$\frac{3}{2}$
DIO-Adapt [33]	$O(k^2N^2\log(N))$	$O(k^2+N)$	$\frac{3}{2}$
FS-Adapt [18]	$O(k^2N\log(N))$	$O(k^2+N)$	$\frac{1+\delta}{1-\delta}\cdot\frac{3}{2}$
US-Adapt [2]	$O(k^2\cdot\frac{N}{\epsilon}\log(\frac{N}{\epsilon})\log\frac{1}{\epsilon})$	$O(k^2+N)$	$\frac{1+\epsilon}{1-\epsilon}\cdot\frac{3}{2}$
K-Algo-Adapt [12]	$O(k^2\cdot\frac{N}{\epsilon}\log(\frac{N}{\epsilon}))$	$O(k^2+N)$	$\frac{1+\epsilon}{1-\epsilon}\cdot\frac{3}{2}$
SP-Oracle-Adapt [8]	$O(\frac{N\log\frac{1}{\epsilon}}{\zeta\cdot\epsilon^{2.5}}\log^3(\frac{N\log\frac{1}{\epsilon}}{\zeta\cdot\epsilon^{1.5}})\log^2\frac{1}{\epsilon}$ $+k^2(\frac{1}{\epsilon}\log\frac{1}{\epsilon}+\log\log(\frac{N}{\zeta\cdot\sqrt{\epsilon}})))$	$O(\frac{N\log\frac{1}{\epsilon}}{\zeta\cdot\epsilon^2}\log^2\frac{N\log\frac{1}{\epsilon}}{\zeta\cdot\epsilon^{1.5}}\log^2\frac{1}{\epsilon}+k^2)$	$\frac{1+\epsilon}{1-\epsilon}\cdot\frac{3}{2}$
SE-Oracle-Adapt [31]	$O(\frac{N\log^2 N}{\epsilon^{2\beta}}+\frac{nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}}\cdot\log\frac{n\log\frac{1}{\epsilon}}{\sqrt{\epsilon}}$ $+\frac{nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}\cdot\epsilon^{2\beta}}+k^2h)$	$O(\frac{nh\log\frac{1}{\epsilon}}{\sqrt{\epsilon}\cdot\epsilon^{2\beta}}+k^2)$	$\frac{1+\epsilon}{1-\epsilon}\cdot\frac{3}{2}$
EAR-Oracle-Adapt [9]	$O(N\log(N)+\frac{N\log(N)}{\zeta\cdot\epsilon^{2\beta}}+\frac{Nh}{\zeta}\log(N)$ $+\frac{Nh}{\zeta\cdot\epsilon^{2\beta}}+k^2\zeta\log(\zeta)$	$O(\frac{N}{\zeta^2}+\frac{Nh}{\zeta\cdot\epsilon^{2\beta}}+k^2)$	$\frac{1+\epsilon}{1-\epsilon}\cdot\frac{3}{2}$
Ours	$O(\frac{k}{\epsilon^{2\beta}}N\log^2 N)$	$O(\frac{k}{\epsilon^{2\beta}}+N)$	$2\cdot(1+\epsilon)$

Remark: γ is the maximum relative difference between geodesic distance and Euclidean distance between any two points s and t on T (i.e., $\gamma=\max_{s,t\in T}\frac{d_g(s,t)}{d(s,t)}$). 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 T (see [18] for details). β is a small constant in the range of $[1.5,2]$ and h is around 10-30 by results of [9, 31, 32]. ζ is the number of regions in *SP-Oracle* and *EAR-Oracle* which is normally set to be 16 or 256.

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

4 Related Work and Baselines

In this section, we discuss related works and show how we adapt existing algorithms for CPTS, which are compared experimentally.

4.1 Related Work

Geodesic Shortest Distance and Path Queries. The index-free exact algorithms *MMP* [21], *CH* [4], *ICH* [35], *VS* [30], *DIO* [33], and *KS* [11] 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. Later on, the index-free approximation algorithms were proposed to expedite the query processing. All existing on-the-fly approximation algorithms [2, 12, 18] 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 which 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 only differ in the method for introducing the Steiner points and Steiner edges.

Afterward, index-based algorithms [8–10, 31] for the geodesic shortest path computation were proposed to further accelerate the query processing. The first attempt was a Single-Source All-Destination algorithm [10], where the source point must be known apriori and kept fixed in the query phase. SP-Oracle [8] builds an indexing structure for the shortest path query processing on the Steiner graph. Inspired by [3, 25, 26], SE-Oracle and EAR-Oracle [9, 31] 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 [6, 7, 27, 36, 37] on the terrain surface. Specifically, [6, 7] 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. [27] further studied kNN queries and proposed a Voronoi Diagram method for kNN queries on a set of static points. [32] developed a kNN algorithm based on *SE-Oracle* [31]. [36] studied the problem of dynamic monitoring the kNN for moving objects. [37] studied reverse k nearest neighbors queries.

Other Related Work. [20] studied the problem of finding the geodesic shortest path satisfying a slope constraint and [13] 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). [39, 40] studied the path queries on the dynamic terrain or moving objects, but their algorithms are inferior to the aforementioned shortest path query algorithms in the static setting. Although constrained shortest path has been extensively studied in Euclidean space [17, 19], in the context of terrain surfaces, this problem becomes highly challenging. The most fundamental problems (e.g., the shortest distance/path query processing) take constant time in Euclidean space but take more than $O(N \log N)$ time on terrain surfaces.

4.2 Baseline Algorithms

As far as we are concerned, existing studies (as surveyed in Section 4.1) focus on shortest distance/path queries and (reverse) nearest neighbor queries on terrain surfaces. Constrained shortest path on terrain surfaces (CPTS) has not been studied before. Although the existing works cannot directly be applied to CPTS, we propose several categories of adaptations to solve CPTS. Denote by N, P, and k the total number of vertices on the terrain surface T, 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 (detailed analysis in the technical report [34]). The adaptations are detailed as follows and they are either *inefficient*, *space-consuming* or *inaccurate*.

Adaptation of Constrained Shortest Path on Euclidean Space (TPES). TPES finds the shortest path Π_E from s to t passing through each point in P on the 2D x-y plane via a constrained shortest path algorithm in Euclidean space [17, 19]. In particular, it first constructs a complete and undirected query graph G_E. The vertices of G_E are all 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_E whose weight is the Euclidean distance between u and v. Next, it applies the state-of-the-art approximation algorithm [17] (which is based on Integer Linear Programming), whose approximation ratio and running time are $\frac{3}{2}$ and $O(k)$, respectively. After that, it finds the path on the terrain surface whose projection on the 2D x-y plane coincides with Π_E. This baseline suffers from a significantly large approximation ratio due to the difference between geodesic distances and Euclidean distances. Besides, since it requires a complete graph G_E, it incurs an $O(k^2)$ space overhead and processing time, imposing limitations on scaling to large k.

Nearest Neighbor-based Adaptations (NNA). The pseudocode of NNA is shown in Algorithm 3. It starts from s and iteratively travels to the nearest unvisited POI nn via the geodesic shortest path from the currently visited point (Lines 5-8). This process is repeated until all POIs are visited (Line 4) and the destination 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 [7], which has a running time of $O(N^2 \log N))$ and as a result, the overall running time of NNA is $O(k \cdot N^2 \log N)$. The nearest neighbor algorithms in [27, 32, 36] cannot be applied to NNA since they are designed for finding the nearest neighbor

Algorithm		Running Time ($\leq O(k \cdot N \log^2 N)$?)	Space Overhead ($O(k+N)$?)	Appro. Ratio (Small constant?)
TPES [19]		Yes	No	No
NNA		No	Yes	No
GDBA	*KS-Adapt* [11]	No	No	Yes
	DIO-Adapt [33]	No	No	Yes
	FS-Adapt [18]	No	No	No
	US-Adapt [2]	No	No	Yes
	K-Algo-Adapt [12]	No	No	Yes
DOA	*SP-Oracle-Adapt* [8]	No	No	Yes
	SE-Oracle-Adapt [31]	No	No	Yes
	EAR-Oracle-Adapt [9]	No	No	Yes
Ours		Yes	Yes	Yes

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

Algorithm 3: 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: The geodesic shortest trip path $\Pi_g(s, t|P)$

```
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 ∅;
4  while Q is not ∅ do
5      Find the point nn in Q nearest to o by the algorithm [7];
6      Find the geodesic shortest path π' from o to nn by the fastest geodesic shortest path
         finding algorithm [33];
7      Append π' to the path π;
8      o ← nn;
9      Remove nn from Q;
10 Find the geodesic shortest path π' from o to t through the fastest geodesic shortest path
     computation algorithm [33];
11 Append π' to the path π;
12 return π;
```

in a fixed set of candidate points. In NNA, however, the set $\mathcal{X}$ of candidate points is dynamic (since we remove points from $\mathcal{X}$ in each iteration).

Geodesic Distance-based Adaptations (GDBA). The pseudocode is in Algorithm 4. We compute the pairwise geodesic path between each pair of points in $P \cup \{s, t\}$ (Line 2). Due to storage limit, only the distances (not the exact paths) are stored. 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, it applies the state-of-the-art approximation algorithm [17] (which is based on Integer Linear Programming and the running time is $O(k)$) to find the shortest path from s to t that passes through all points in P on G_Q (Line 4). After obtaining the path, we compute the geodesic shortest 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 geodesic shortest path between u and v, 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

Algorithm 4: 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: The geodesic shortest trip path $\Pi_g(s, t|P)$

1 Initialize the path π to be $\emptyset$;
2 Call a geodesic shortest 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 [17];
5 **for** *each pair of adjacent points p and q in* $\Pi_E(s, t|P)$ **do**
6 Find the geodesic shortest path π' from p to q by the fastest geodesic shortest path finding algorithm [33];
7 $\pi \leftarrow$ Concatenation of π and π';
8 **return** π;

FS, *US*, and *K-Algo*. These adaptations are called *KS-Adapt*, *DIO-Adapt*, *FS-Adapt*, *US-Adapt*, and *K-Algo-Adapt*, respectively. Unfortunately, adaptations from this category suffer from a long time for constructing 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. We build a distance oracle (that is, an indexing structure to answer geodesic distance queries) on the terrain surface and utilize the distance oracle to find the pairwise geodesic distances. Then, the remaining algorithm is the same as Lines 3-8 of Algorithm 4. We consider all existing distance oracles, namely *SP-Oracle*, *SE-Oracle* and *EAR-Oracle*, in this category, and the adaptations are denoted as *SP-Oracle-Adapt*, *SE-Oracle-Adapt*, and *EAR-Oracle-Adapt*, respectively.

We further enhance the DOA algorithms as follows for our problem. *SE-Oracle-Adapt* can be directly improved if the underlying distance oracle *SE-Oracle* is build upon the points in $P \cup \{s, t\}$ only instead of the entire terrain, since *SE-Oracle* was originally designed for the geodesic distance queries between a set of pre-defined points (e.g., $P \cup \{s, t\}$ in our problem). For *SP-Oracle-Adapt* and *EAR-Oracle-Adapt*, their underlying distance oracles *SP-Oracle* and *EAR-Oracle* are both partition-based. That is, their indexing structures uniformly partition the terrain surface into ζ regions and connect the regions with a highway network. To compute the geodesic distance between any pair of points s and t, it suffices to keep only the highway network and the regions containing at least one point in $P \cup \{s, t\}$ (and their associated components in the indexing structures). Therefore, we tailor the two adaptations *SP-Oracle-Adapt* and *EAR-Oracle-Adapt* as described above. Let ζ' denote the number of kept regions, where $1 \leq \zeta' \leq \zeta$. Let N' denote the total number of vertices in the highway network and the ζ' kept regions. Thus, we obtain that $N' = \frac{\zeta' \cdot N}{\zeta} = O(\frac{N}{\zeta})$.

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 the query graph is still quadratic to k. 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 k.

Dataset	No. of Vertices	Region Covered	Reference
HM	793	15 km^2	[28]
HL	2,470	49 km^2	[28]
RM	3,696	71 km^2	[28]
BH (L)	146,547	$14km \times 10km$	[12, 31]
EP (L)	164,238	$10.7km \times 14km$	[12, 31]
SF (L)	172,186	$14km \times 11.1km$	[12, 31]
BH (H)	1,318,844	$14km \times 10km$	[12, 31]
EP (H)	1,392,236	$10.7km \times 14km$	[12, 31]
SF (H)	1,539,082	$14km \times 11.1km$	[12, 31]

Table 4. Dataset Statistics

4.3 Comparison with Our Algorithm

We compare our proposed algorithm with baselines w.r.t. the running time, space overhead, and approximation ratios in Table 3.

- *Running time*: The running time of all GDBA algorithms is larger than $O(k^2 \cdot N \log N)$ since they involve the expensive construction of the query graph. Besides, the running time of all DOA algorithms (resp. NNA) is quadratic to k (resp. N) and thus, cannot scale well to a large number of POIs (resp. large terrain). In contrast, the running time of our algorithm is $O(k \cdot N \log^2 N)$ which is linear to both k and $N \log^2 N$. The running time of TPES is also small since it find paths in the 2D Euclidean space.
- *Space overhead*: TPES, all GDBA algorithms and all DOA algorithms have a space overhead larger than $O(k^2)$ due to the space complexity of the query graph G_E or G_Q. This prevents their usage from the applications with a lot of POIs (e.g.,≥1000 POIs as motivated in Section 1). Besides, each DOA algorithm also needs to maintain a bulky distance oracle on the terrain surface. The space overhead of NNA, and our algorithm is $O(k)$.
- *Approximation Ratio*: The approximation ratios of NNA and FS-Adapt are large (e.g., $\geq k$). Worse still, the approximation ratio of TPES is even larger due to the significant difference between the geodesic distance and Euclidean distance. Thus, their returned trips can be very long and the users' experience will be degraded when there are a lot of POIs (e.g.,≥1000 POIs). Other algorithms have a small constant of approximation ratio.

As discussed above, none of the adaptations of existing algorithms can be efficient, space-saving and accurate simultaneously for CPTS. Our algorithm is the only one capable of *striking a balance among the running time, space overhead, and approximation ratio.*

5 Experiment

The experiment was conducted on a Linux machine (3.60 GHz Intel Core CPU and 32 GB RAM). Algorithms were implemented in C++.

5.1 Experimental Setting

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 [28]. Following the existing study [9], we first considered three 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

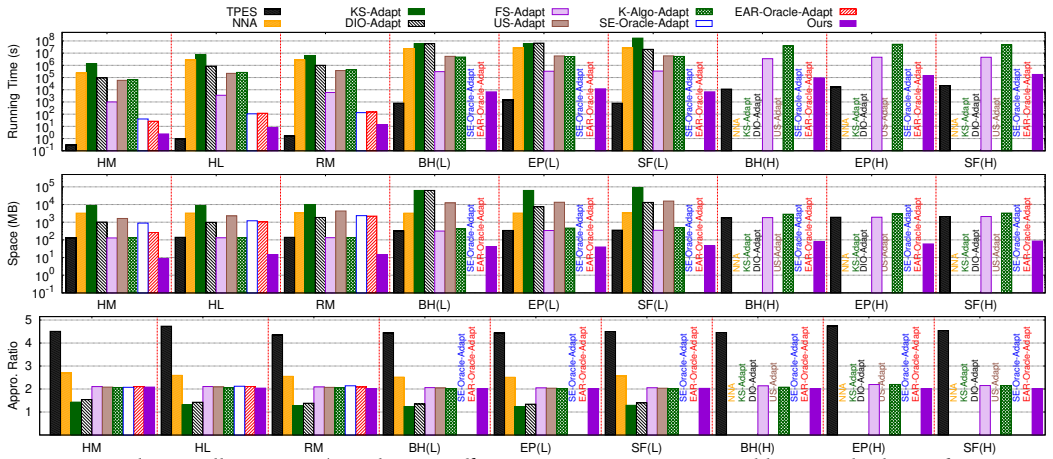


Fig. 4. Results on All Datasets (Results on Different Datasets are Separated by a Dashed Line for Better Visualization)

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 [9]. 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 4.2 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 reported in [9, 31], *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, (2) the number k of points-of-interest in the query, and (3) the error parameter ϵ. By default, ϵ is set to be 0.2 and k is set to be 1000, to simulate the needs of real-world applications [1, 16, 22, 29].

We consider the following measurements for each algorithm in the experiment. (1) The running time for processing a query, (2) the space consumption in the query processing, and (3) the *empirical* 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 [33]. 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 adopt a standard technique for *Travel Salesman Walk Problem (TSWP)* algorithm [17] to find the exact solution on G_Q.

5.2 Experimental Results

Results on All Real Datasets. The running time, space consumption, and empirical 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 to be elaborated shortly). We explicitly mark those baselines in the bar chart. As shown in Figure 4, we observe the following, which conform to our theoretical analysis.

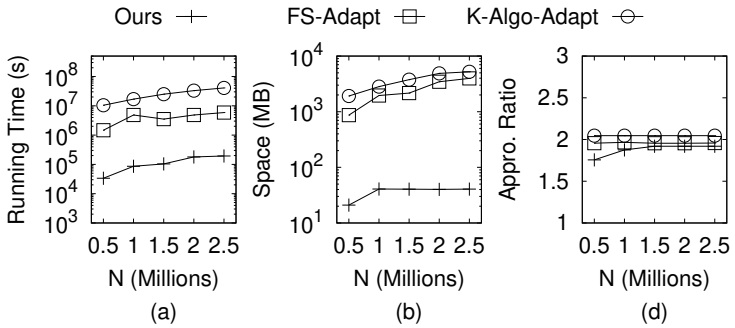


Fig. 5. Effect of N (No. of Vertices on the Terrain Surface)

(1) Our algorithm is the fastest method (except *TPES*) among those with a small constant of approximation ratio. Note that although *TPES* is faster than ours, it has (a) an unsatisfactory theoretical approximation ratio, (b) an unfavorable theoretical space complexity, and (c) the worst empirical approximation ratio. The running time of other baselines 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 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 three smallest datasets. Moreover, *US-Adapt* introduced too many auxiliary points, leading to a large space overhead. Consequently, it could not scale to the three high resolution datasets.

(3) The empirical approximation ratio of our algorithm was comparable to most of the other algorithms (which is around 2). In comparison, the empirical approximation ratio of *NNA* and *TPES* was the worst. Although *KS-Adapt* and *DIO-Adapt* had the smallest empirical approximation ratio, 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 of subsequent experiments for clarity by omitting the following baselines: (1) DOA (including *SE-Oracle-Adapt* and *EAR-Oracle-Adapt*) and *US-Adapt*, since their space consumption is too large to fit our limited memory budget, especially in the three largest datasets; (2) NNA, *DIO-Adapt*, and *KS-Adapt*, since their running time of is large and they cannot be finished in a reasonable time (i.e., a month) in the three largest datasets; (3) TPES, since it has a forbiddingly large approximation ratio which prevents its usage in real applications. Below we present the sensitivity test of our algorithm.

Effect of N. To test the scalability of each algorithm, we adopted five large terrain datasets generated from *EP (H)*. Specifically, we first enlarged the size of *EP (H)* by an up-sampling method, which inserted a new vertex at the center of each face of *EP (H)* and introduced a new edge between the new 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. It was up to two orders of magnitude faster than the baseline methods, while its space consumption was significantly smaller.

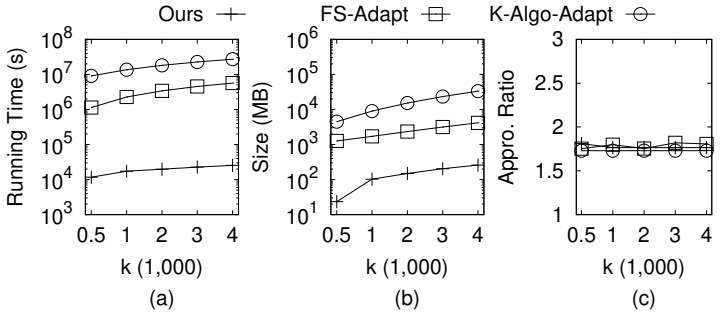


Fig. 6. Effect of k

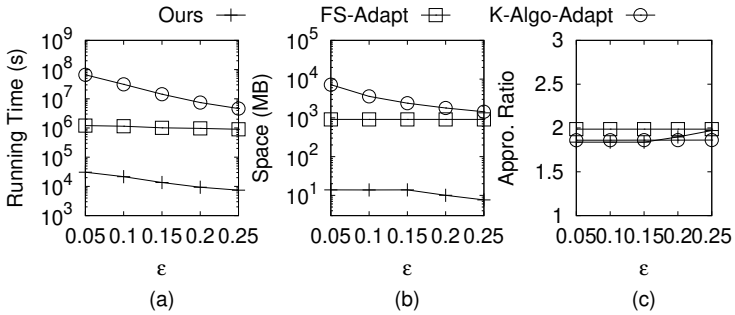


Fig. 7. Effect of ϵ

Its empirical approximation ratio also had a slight improvement. These findings were consistent with the theoretical analysis.

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 empirical approximation ratio of each algorithm was not sensitive to the values of k. The observations above were also consistent with the theoretical analysis. As the figure shows, our algorithm outperformed the two baselines by 1-2 orders of magnitude in both running time and space overhead, while keeping a comparable approximation 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. 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 approximation ratio.

5.3 Case Study

In this section, we have undertaken a case study within a specific application context of the *Path Advisor* application [38]. This app, renowned for its web and mobile functionality, is extensively utilized by a particular university. It offers location-based services such as navigation and point of interest (POI) suggestions, utilizing a 3D terrain with 1,789 vertices unique to that university's campus.

We consider a campus visit which can be formulated as a constrained shortest path query as follows. The visit starts from the south entrance of the university and the destination is the north

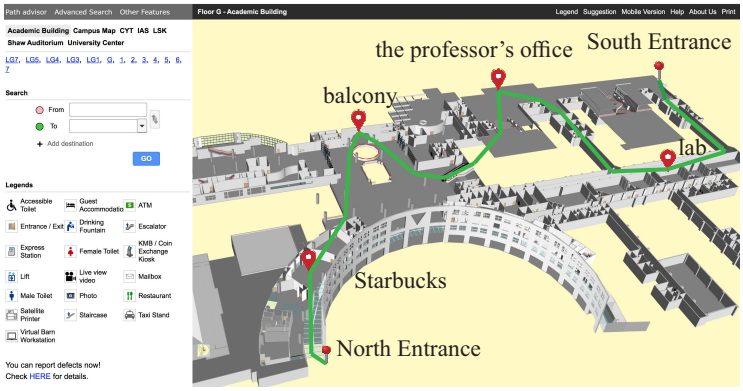


Fig. 8. An Illustration of Path Advisor in A Campus

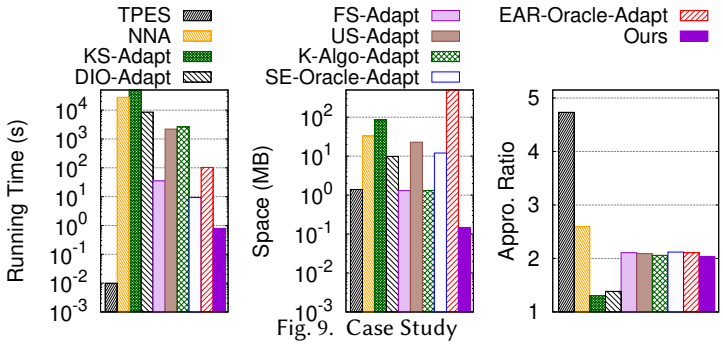


Fig. 9. Case Study

entrance. The visit tour should pass through one professor's office (to visit the professor), the balcony of the atrium for sightseeing, the lab of his friends, and the Starbucks for coffee. Figure 8 illustrates the user interface of the Path Advisor and displays the specifics of the previously mentioned query. The image reveals a menu on the left that offers search functionalities and includes legends. To the right, the figure presents a 3D representation of the campus terrain. The route of interest is depicted as a green line within Figure 8.

Figure 9 provides the experimental outcomes of our algorithm, along with the results of all the baseline methods that were evaluated. As shown there, we observe that (1) *TPES* is the fastest algorithm but it has an unsatisfactory empirical approximation ratio which prevents its usage in practice, and (2) our algorithm is superior compared other baselines in terms of both running time and space overhead with a comparable empirical approximation 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) TPES suffers from a large and uncontrollable approximation ratio; (2) DOA (including *SE-Oracle-Adapt* and *SE-Oracle-Adapt*) all have a forbiddingly large space consumption; and (3) *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 while maintaining, and in some cases improving, the empirical approximation ratio.

6 Conclusion

In this paper, we study a fundamental and important problem called *Constrained Shortest Path Finding on Terrain Surfaces (CPTS)*, which finds a plethora of applications in GIS, military tactical analysis, map services, etc. We prove that CPTS is NP-hard and propose an efficient approximation algorithm. 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 comparable accuracy.

Acknowledgments

We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Victor Junqiu WEI was supported in part by the Guangdong Basic and Applied Basic Research Foundation (Grant No.: GDST23EG32, Project No.: 2024A1515011667), Macau Science and Technology Development Fund (FDCT-25-074-SCSE, Project No.: 0094/2025/ITP2) and Faculty Research Grant of MUST (FRG-25-080-FIE). The research of Min Xie was supported by Guangdong Science and Technology Department 2024TQ08X196. The research of Weicheng Wang was supported in part by the National Natural Science Fund of China under Grant 25IAA02099.

References

[1] Joel Alberts, Dean Edwards, Terence Soule, Mike Anderson, and Michael O'Rourke. 2008. Autonomous navigation of an unmanned ground vehicle in unstructured forest terrain. In *2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*. IEEE, 103–108.

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

[3] 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*.

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

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

[6] 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)*.

[7] 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*.

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

[9] 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).

[10] 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)*.

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

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

[13] 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*.

[14] 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*.

[15] 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.

[16] 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*.

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

[18] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. In *Algorithmica*.
[19] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. 2005. On trip planning queries in spatial databases. In *International symposium on spatial and temporal databases*. Springer, 273–290.
[20] Lian Liu and Raymond Chi-Wing Wong. 2011. Finding shortest path on land surface. In *SIGMOD*.
[21] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. In *SIAM Journal on Computing*.
[22] Panagiotis Papadakis. 2013. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence* 26, 4 (2013), 1373–1385.
[23] Christos H Papadimitriou. 1977. The Euclidean travelling salesman problem is NP-complete. *Theoretical computer science* 4, 3 (1977), 237–244.
[24] Louis V Quintas and Fred Supnick. 1965. On some properties of shortest Hamiltonian circuits. *The American Mathematical Monthly* 72, 9 (1965), 977–980.
[25] Jagan Sankaranarayanan and Hanan Samet. 2009. Distance oracles for spatial networks.. In In *Proceedings of the 25th IEEE International Conference on Data Engineering*.
[26] Jagan Sankaranarayanan and Hanan Samet. 2010. Query processing using distance oracles for spatial networks. In *IEEE Transactions on Knowledge and Data Engineering*, (Best Papers of ICDE 2009 Special Issue.).
[27] Cyrus Shahabi, Lu-An Tang, and Songhua Xing. 2008. Indexing land surface for efficient knn query. In *VLDB*.
[28] 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
[29] Nicolas Vandapel, Raghavendra Rao Donamukkala, and Martial Hebert. 2006. Unmanned ground vehicle navigation using aerial ladar data. In *The International Journal of Robotics Research*.
[30] Vishal Verma and Jack Snoeyink. 2009. Reducing the memory required to find a geodesic shortest path on a large mesh. In *SIGSPATIAL*.
[31] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, and David M. Mount. 2017. Distance Oracle on Terrain Surface. In *SIGMOD*.
[32] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, David M Mount, and Hanan Samet. 2022. Proximity Queries on Terrain Surface. *TODS* (2022).
[33] 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).
[34] Victor Junqiu Wei, Min Xie, and Weicheng Wang. 2026. Constrained Shortest Path Finding on Terrain Surfaces (technical report). In *https://github.com/ItachiUchihaVictor/TPTS/blob/main/technical.pdf*.
[35] Shi-Qing Xin and Guo-Jin Wang. 2009. Improving Chen and Han's Algorithm on the Discrete Geodesic Problem. In *TOG*.
[36] S. Xing, C. Shahabi, and B. Pan. 2009. Continuous monitoring of nearest neighbors on land surface. In *VLDB*.
[37] D. Yan, Z. Zhao, and W. Ng. 2012. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *CIKM*.
[38] Yinzhao Yan and Raymond Chi wing Wong. 2021. Path Advisor: A Multi-Functional Campus Map Tool for Shortest Path. *VLDB* (2021).
[39] Yinzhao Yan and Raymond Chi-Wing Wong. 2024. Efficient Shortest Path Queries on 3D Weighted Terrain Surfaces for Moving Objects. In *2024 25th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 11–20.
[40] Yinzhao Yan, Raymond Chi-Wing Wong, and Christian S Jensen. 2024. An Efficiently Updatable Path Oracle for Terrain Surfaces. *IEEE Transactions on Knowledge and Data Engineering* (2024).

Received October 2025; revised January 2026; accepted February 2026