Efficient Shortest Path Queries on 3D Weighted Terrain Surfaces for Moving Objects

Yinzhao Yan
The Hong Kong University of Science and Technology
yyanas@cse.ust.hk

Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology
raywong@cse.ust.hk

***Abstract*—Studying the shortest path query for moving objects on a terrain surface has aroused widespread concern in industry and academia. In this paper, we study the *weighted region problem*, which aims at finding the shortest path between two points passing different regions on a 3D weighted terrain surface and different regions are assigned different weights. We propose an efficient $(1+\epsilon)$-approximate on-the-fly algorithm to solve it. Our experimental results show that our algorithm is up to 1630 times and 40 times better than the best-known algorithm in terms of running time and memory usage in realistic settings[1].**

I. Introduction

The shortest path query for *moving objects* becomes increasingly widespread nowadays [6], [23], especially on terrain datasets. In industry, Metaverse and Google Earth use terrain datasets (e.g., mountains and valleys) with different features (e.g., water and forest) to help moving users efficiently reach destination. In academia, the shortest path query on terrain datasets is a prevalent research topic in the field of mobile data management [9], [11], [12], [24], [25], [26], [28], [29]. A terrain surface contains a set of *faces* each of which is denoted by a triangle. Each face consists of three *edges* connecting at three *vertices*. The *weighted* (resp. *unweighted*) *shortest path* on a terrain surface means the shortest path between a source s and a destination t passing on the face of the terrain where each face is assigned with a *weight* (resp. each face weight is set to a fixed value, e.g., 1). Figures 1 (a) and (b) show a real map and a terrain surface (of Valais, Switzerland [2] with an area of 20km^2) with weighted and unweighted shortest paths in the blue and purple dashed lines from s to t.

A. Motivation

Computing the weighted shortest path on terrain surfaces between two points is involved in numerous applications with different interpretations of the faces' weights on the terrain.

1) ***Earthquake and avalanche****:* After an earthquake or avalanche, rescue teams (as moving objects) need to efficiently find the shortest rescue paths for life-saving (i.e., they need to first find the path, and then follow the path as moving objects), and they can save 1 life every 5 minutes [17]. The death toll of the 7.6 magnitude earthquake on Jan 1, 2024 in Japan exceeded 200 [1]. The 4.1 magnitude earthquake on Oct 24, 2016 in Valais, Switzerland [2] caused an avalanche. In Figures 1 (a) and (b), the terrain surface consists of destroyed and non-destroyed regions (faces with green and white color). Since the rescue time for passing through the destroyed region is longer than the non-destroyed region, we set the weight of each terrain face to be the *damage level* [22] (a static value) of each region, i.e., destroyed / non-destroyed regions' terrain faces will have a larger / smaller weight. The blue path and purple dashed path between s (a rescue center) and t (a destroyed village) has a *weighted distance* of 5.1km and 109km, the rescue time of using these two paths are 3 min and 1.4 hours, so the rescue team will choose the blue path that does not pass the destroyed region. We cannot pre-compute the path before the earthquake. But, after an earthquake, satellites can collect the terrain surface in 10s and USD $48.72 [20] for a 1km^2 region, our algorithm can calculate a weighted shortest rescue path in 7.3s, and the rescue team can arrive at the destroyed village in only 3 min $\approx$ 3 min + 10s + 7.3s.

2) ***Path Advisor****: Path Advisor* [29] is a Geographic Information System (GIS) based mobile app to assist students (as moving objects) for finding the shortest path between two rooms in a university campus. Figure 1 (c) shows a weighted terrain surface (represents the building's floor) in Path Advisor. For safety, the path should maintain a minimum distance (e.g., 0.2m) from obstacles. So, with a weight-distance function [29], faces on the floor closer to the aisle boundaries / centers are assigned larger / smaller weights. The weighted shortest path in blue line is more realistic than the unweighted shortest path in purple dashed line, since the former path maintains a safe distance from obstacles. There are over 5,000 rooms in the university, so pre-computing the pairwise shortest paths is undesirable, even if we use an approximate oracle for indexing [25], [26]. Thus, it is necessary that the mobile app can efficiently calculate the path for real-time responses.

Motivated by these, we aim to efficiently solve the *weighted region problem*, i.e., finding the shortest path for moving objects between two points passing different regions on a 3D weighted terrain surface and different regions are assigned different weights depending on the application nature.

B. Snell's law

Consider a weighted terrain surface T with n vertices. Let V, E and F be the set of vertices, edges and faces of T. In Physics, when a light ray passes the boundary of two different media (e.g., air and glass), it bends at the boundary since light seeks the path with the minimum time. The angles of incidence

[1]Code: https://github.com/yanyinzhao/WeightedTerrainCode

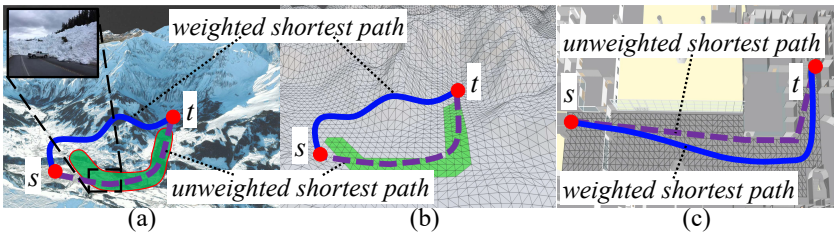

Fig. 1. Blue weighted and purple dashed unweighted shortest paths

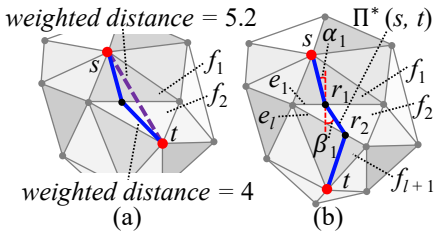

Fig. 2. Snell's law illustration

and refraction for the light satisfy *Snell's law* [19], which is an important geometric information of T and can be applied to calculating the weighted shortest path (see Figures 2 (a) and (b)). In Figure 2 (a), if all faces have the same weights (=1), the purple dashed line is the (unweighted) shortest path. When faces f_1 and f_2 have weights 3 and 2, the blue line with a weighted distance of 4 that satisfies Snell's law is the weighted shortest path, but the weighted distance of the purple dashed line is $5.2 > 4$. Figure 2 (b) shows a similar example.

C. Challenges

No algorithm can solve the weighted region problem *exactly* when the number of faces in T exceeds two [8], but two categories of algorithms can solve it on-the-fly *approximately*.

1) ***Steiner point approach****:* Algorithms [4], [11], [12], [15] place discrete (i.e., Steiner) points on edges in E, and use Dijkstra's algorithm [10] on a weighted graph constructed using these Steiner points and V to calculate the result path. The best-known algorithm [11], [15] calculates a $(1+\epsilon)$-approximate weighted shortest path in $O(n^3 \log n)$ time (where $\epsilon > 0$ is the *error parameter*), which is still very slow, since it does not utilize any geometric information on T. Algorithm [11] uses algorithm [15] as an on-the-fly algorithm to build an oracle, we regard them as one algorithm for the sake of illustration. Our experimental results show that the best-known algorithm [11], [15] runs in 119,000s $\approx$ 1.5 days on a terrain surface with 50k faces.

2) ***Edge sequence approach****:* Algorithm [24] uses algorithm [11], [15] with slight variations (without error guarantee) to calculate a path, and then uses Snell's law on the edge sequence passed by this path based on T to calculate an even shorter path. But, it does not have an error guarantee of the returned path's distance with a given time limit.

D. Our Efficient Algorithm

We propose an efficient on-the-fly two-step algorithm for solving the 3D weighted region problem using algorithm *Rough-Refine* (*Roug-Ref*), such that for a given source s and destination t on T, it returns a $(1+\epsilon)$-approximate weighted shortest path between s and t. Algorithm *Roug-Ref* is a step-by-step algorithm involving algorithm *Roug* and algorithm *Ref*. (1) In algorithm *Roug*, given T, s, t and ϵ, we efficiently find a $(1+\eta\epsilon)$-approximate *rough path* between s and t using Steiner points, where $\eta > 1$ is a constant and is calculated based on T and ϵ (note that $\eta \in (1, 2]$ on average in our experiments), as shown in Figures 3 (a) to (d). (2) In algorithm *Ref*, given the rough path, we efficiently *refine* it to be a $(1+\epsilon)$-approximate weighted shortest path using Snell's law, as shown in Figures 3 (e) to (i). Algorithm *Roug-Ref* achieves outstanding performance in terms of running time and memory usage due to the rough-refine concept, i.e., (1) a *novel* pruning step in algorithm *Roug* during the rough path calculation with error guarantee (achieved by transferring the pruned-out information from algorithm *Roug* to algorithm *Ref*, and after conducting necessary checks in algorithm *Ref*, there is no need to perform calculations on the pruned-out information anymore), (2) an *efficient* reduction of the search area in algorithm *Roug* before Snell's law refinement (achieved by the calculation of the rough path), and (3) the usage of *Snell's law* in algorithm *Ref* for efficient refinement.

We have four additional novel techniques for further speedup, including (1) an *efficient* Steiner point placement scheme in algorithm *Roug* for reducing the number of Steiner points during the rough path calculation (achieved by considering geometric information of each face in F on T, such as face weight, internal angle and edge length), (2) a *progressive* approach in algorithm *Ref* for minimizing the search area before Snell's law refinement (achieved by progressively exploring the local search area instead of directly using the global search area), (3) an *effective weight* pruning technique in algorithm *Ref* for faster processing during Snell's law refinement (achieved by considering additional information on T), and (4) a *novel* error guaranteed pruning technique in algorithm *Ref* for handling rare cases when we are unable to use Snell's law to refine a rough path to a $(1+\epsilon)$-approximate weighted shortest path, and then an additional step is required for error guarantee, but the algorithm total running time will not increase a lot (achieved by re-using the calculated information in algorithm *Roug*).

E. Contributions and Organization

We summarize our major contributions.

(1) We are the first to propose the efficient algorithm *Roug-Ref*, since the rough-refine concept and four additional techniques are absent in algorithms [4], [11], [12], [15], [24].

(2) We provide a thorough theoretical analysis on the running time, memory usage and error bound of our algorithm.

(3) Our experimental results show that our algorithm runs up to 1630 times faster than the best-known algorithm [11], [15] on benchmark real datasets with the same error ratio, e.g., for a terrain surface with 50k faces with $\epsilon = 0.1$, our algorithm runs in 73s $\approx$ 1.2 min and uses 43MB of memory,

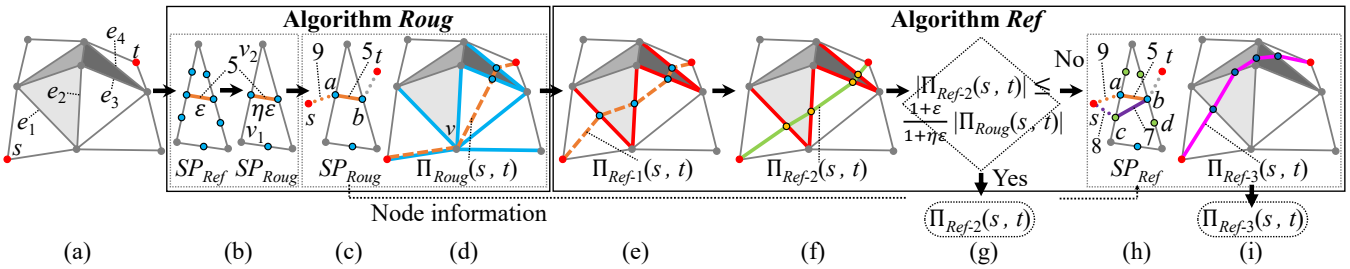

Fig. 3. Framework overview, where SP_{Roug} and SP_{Ref} are the set of Steiner points on each edge of E used in algorithm *Roug* and *Ref*, $\Pi_{Roug}(s,t)$ is the rough path calculated using algorithm *Roug*, $\Pi_{Ref\text{-}1}(s,t)$ is the modified rough path calculated using the full edge sequence conversion step of algorithm *Ref*, $\Pi_{Ref\text{-}2}(s,t)$ is the refined path calculated using the Snell's law path refinement step of algorithm *Ref*, and $\Pi_{Ref\text{-}3}(s,t)$ is the refined path calculated using the error guaranteed path refinement step of algorithm *Ref*

but the best-known $(1+\epsilon)$-approximate algorithm [11], [15] runs in 119,000s $\approx$ 1.5 days and uses 2.9GB of memory. Our algorithm is the optimal algorithm in earthquake rescue.

The remainder of the paper is organized as follows. Section II provides the preliminary. Section III covers the related work. Section IV presents our algorithm. Section V presents the experimental results and Section VI concludes the paper.

II. Preliminary

A. Notations and Definitions

1) ***Terrain surfaces and paths****:* Consider a weighted terrain surface T with n vertices. Let V, E and F be the set of vertices, edges and faces of T. Each vertex $v \in V$ has three coordinate values, x_v, y_v and z_v. Let L be the length of the longest edge of T, and N be the smallest integer value that is larger than or equal to the coordinate value of any vertex in V. If two faces share a common edge, they are said to be *adjacent*. Each face $f_i \in F$ is assigned a weight $w_i > 0$, and the weight of an edge is the smaller weight of the two faces containing the edge. The maximum and minimum weights of the face in F are denoted by W and w. The minimum height (of the 2D triangle with a base) of a face in F is denoted by h. Given a face f_i, and two points p and q on f_i, let $\overline{pq}$ be a line segment between p and q on f_i, $d(p,q)$ be the Euclidean distance between p and q on f_i, and $D(p,q) = w_i \cdot d(p,q)$ be the *weighted (surface) distance* from p to q on f_i. Given s and t in V, let $\Pi^*(s,t) = (s, r_1, \ldots, r_l, t)$ be the *optimal weighted shortest path* on T (with $l \geq 0$) such that $\sum_{i=0}^{l} D(r_i, r_{i+1})$ is the minimum, where $r_0 = s$ and $r_{l+1} = t$. Here, for each $i \in \{1, \ldots, l\}$, r_i is a point on an edge in E, is named as an *optimal intersection point* in $\Pi^*(s,t)$. The blue path in Figure 2 (b) shows an example of $\Pi^*(s,t) = (s, r_1, r_2, t)$ on T. We define $|\cdot|$ to be the weighted distance of a path, e.g., $|\Pi^*(s,t)|$ is the weighted distance of $\Pi^*(s,t)$. Let $\Pi(s,t)$ be the path result of our algorithm. If point s (or t) is not in V, we can regard it as a new vertex and then add three new triangles each involving this point and three vertices of the face containing this point. A notation table can be found in our technical report [30].

2) ***Snell's Law****:* Let $S = ((v_1, v_1'), \ldots, (v_l, v_l')) = (e_1, \ldots, e_l)$ be a sequence of edges that $\Pi^*(s,t)$ connects from s to t in order based on T, where S is said to be *passed by* $\Pi^*(s,t)$. Let l be the number of edges in S. Let $F(S) = (f_1, f_2, \ldots, f_l, f_{l+1})$ be an adjacent face sequence corresponds to S, such that for every f_i with $i \in \{2, \ldots l\}$, f_i is the face that contains e_i and e_{i+1} in S, while f_1 is the face that adjacent to f_2 at e_1 and f_{l+1} is the face that adjacent to f_l at e_l. We define α_i and β_i to be the incidence and refraction angles of $\Pi^*(s,t)$ on e_i for $i \in \{1, \ldots l\}$, respectively. In Figure 2 (b), $\Pi^*(s,t)$ satisfies Snell's law, $w_i \cdot \sin\alpha_i = w_{i+1} \cdot \sin\beta_i$, with $i \in \{1, \ldots, l\}$.

B. Problem

Given T, s and t, the problem is to calculate a weighted shortest path on T such that $|\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$.

III. Related Work

A. On-the-fly algorithms on the weighted terrain surface

1) ***Steiner point approach****:* Algorithms [4], [11], [12], [15] use Dijkstra's algorithm on the weighted graph constructed using Steiner points and V to calculate an approximate weighted shortest path. Algorithm *Fixed Steiner Point* (*FixSP*) [11], [15] and algorithm *Fixed Steiner Point No Weight Adaption* (*FixSP-NoWei-Adp*) [12] calculates the result path with an error $(1+\epsilon)$, and algorithm *Logarithmic Steiner Point* (*LogSP*) [4] calculates the result path with an error much larger than $(1+\epsilon)$. (i) Algorithm *FixSP* places a uniform number of Steiner points (i.e., $O(n^2)$) per edge. It runs in $O(n^3 \log n)$ time and is regarded as the best-known $(1+\epsilon)$-approximate algorithm for solving the weighted region problem. (ii) Algorithm [12] also places a uniform number of Steiner points per edge, but it was originally designed in the unweighted case. We adapt it to be algorithm *FixSP-NoWei-Adp* [12] in the weighted case, which runs in $O(n^3 \log n)$ time and equivalent to *FixSP*, by assigning a weight to each face, and use the same surface distance definition in Section II-A1. (iii) In algorithm *LogSP*, given an error parameter $\epsilon' > 0$ that determines how far the Steiner points are placed (different from our ϵ that controls the distance error bound), it places $O(\log \frac{c}{\epsilon'})$ Steiner points per edge non-uniformly, i.e., places more Steiner points near a vertex, and has a distance error $(1 + (2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon')$, where $c \in [0.2, 1]$ is a constant depending on T. We adapt it to be algorithm *LogSP-Adp* that runs in $O(n \log \frac{c}{\epsilon} \log(n \log \frac{c}{\epsilon}))$ time, such that given ϵ, we can place Steiner points using ϵ

and have a distance error $(1+\epsilon)$, by finding the relationship between ϵ and ϵ', so these two algorithms can place the same number of Steiner points per edge, and their distance errors are the same, i.e., $1+(2+\frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = 1+\epsilon$, and get $\epsilon' = \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4}$.

Drawbacks of algorithm *FixSP* and *FixSP-NoWei-Adp*: They are very slow due to two reasons. (i) *Absence of Snell's law*: They do not utilize any geometric information *between two adjacent faces in F that share one edge* on T, i.e., Snell's law. So, they place many Steiner points on edges in E. But, we utilize Snell's law to avoid this. (ii) *Uniform number of Steiner points per edge*: They do not utilize any geometric information of *each face in F* on T, such as face weight, internal angle and edge length, and always place a uniform number of Steiner points (i.e., $O(n^2)$) per edge (the distance between two adjacent Steiner points on the same edge is the same) to bound the error. But, we utilize this information and place only $O(\log c')$ Steiner points per edge (the distance between two adjacent Steiner points on the same edge is different), where $c' \in [2,5]$ is a constant depending on T and ϵ. Figures 4 (a) and (b) show the placement of Steiner points in these two algorithms and our algorithm *Roug-Ref* with the same error. Our experimental results show that for a terrain surface with 50k faces and $\epsilon = 0.1$, *Roug-Ref* just places 10 Steiner points per edge to find a rough path in 71s $\approx$ 1.2 min, and finds a refined path in 2s, but both *FixSP* and *FixSP-NoWei-Adp* place more than 600 Steiner points per edge to find the result path in 119,000s $\approx$ 1.5 days.

Drawback of algorithm *LogSP* and *LogSP-Adp*: (i) *LogSP* has the *larger distance error* drawback, since its distance error is always larger than that of other algorithms, making it difficult to compare with other algorithms. (ii) Both of them also have the *absence of Snell's law* drawback. In the same experimental setting of the previous paragraph, *Roug-Ref* runs in 73s $\approx$ 1.2 min, but *LogSP-Adp* runs in 220s $\approx$ 3.7 min.

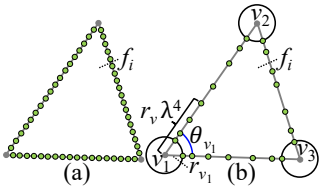

Fig. 4. Steiner points in (a) *FixSP* and *FixSP-NoWei-Adp*, and (b) *Roug-Ref*

TABLE I
ALGORITHM COMPARISONS

Algorithm	Time/Size	Error
EdgSeq [24]	Large	Large
EdgSeq-Adp [24]	Large	Small
FixSP [11], [15]	Large	Small
FixSP-NoWei-Adp [12]	Large	Small
LogSP [4]	Medium	Large
LogSP-Adp [4]	Medium	Small
***Roug-Ref* (ours)**	**Small**	**Small**

*2) **Edge sequence approach**:* Algorithm *Edge Sequence* (*EdgSeq*) [24] uses *FixSP* with only slight variations (i.e., places a uniform and constant number of Steiner points, e.g., 3 and not $O(n^2)$, per edge) to calculate a path without error guarantee, and then uses Snell's law on the edge sequence passed by this path based on T to calculate a shorter path. We adapt it to be algorithm *EdgSeq-Adp*, that uses *FixSP* to calculate a $(1+\epsilon)$-approximate shortest path, and then uses Snell's law on the edge sequence of this path to compute a shorter path that runs in $O(n^3 \log n + n^4 \log(\frac{n^2 NWL}{w\epsilon}))$ time.

Drawbacks of algorithm *EdgSeq* and *EdgSeq-Adp*: (i) *EdgSeq* does not have an error guarantee of the returned path's distance with a given time limit. (ii) *EdgSeq-Adp* is still very slow due to two reasons. (1) *Absence of pruning technique for the edge sequence finding*: It uses Snell's law after *FixSP*, so its running time is an additive result of the running time of *FixSP* and usage of Snell's law. But, we use a pruning technique to further prune out more than half of the Steiner points. (2) *Absence of pruning technique when using Snell's law*: It solely uses binary search in Snell's law on the edge sequence. But, we use *effective weight* information of T for pruning. In the same experimental setting of the previous three paragraphs, *EdgSeq-Adp* runs in 131,000s $\approx$ 1.7 days, but *Roug-Ref* runs in 73s $\approx$ 1.2 min. If we substitute *FixSP* to *LogSP-Adp* in *EdgSeq-Adp*, the new algorithm has the same drawback as of *EdgSeq-Adp*, and its running time is always larger than that of *LogSP-Adp*, so there is no need to consider this adaption.

Comparisons: We compare different algorithms in Table I. *Roug-Ref* has the smallest query time and memory usage.

B. Other related studies

There are some other studies related to our problem but are not exactly our problem. (1) *On-the-fly algorithms on the unweighted terrain surface* [27]: Algorithm [27] is the best-known exact on-the-fly algorithm for the unweighted shortest path calculation, but no known algorithm can adapt it to the weighted case. (2) *Oracles on the terrain surface* [9], [11], [21], [25], [26], [28]: Study [11] uses algorithm *FixSP* [15] to build an oracle on a weighted terrain surface, and studies [25], [26] use algorithm [12] to build an oracle on an unweighted terrain surface. Studies [9], [21], [28] build oracles on an unweighted terrain surface for the k-nearest neighbor query. But, we focus on the *on-the-fly* algorithm. (3) *On-the-fly algorithms on the road network and point cloud* [6], [23], [31]: Algorithms [6], [23] answer the shortest path query on the road network and algorithm [31] answer the shortest path query on the point cloud, which are different from us.

IV. METHODOLOGY

A. Overview

*1) **Concepts**:* We give four concepts:

(i) **The weighted graph**: It is used in Dijkstra's algorithm. Let G_A be a weighted graph used in algorithm *Roug* or *Ref*, $G_A.V$ and $G_A.E$ be the sets of nodes and weighted edges of G_A, where A is a placeholder that can be *Roug* or *Ref*. To build G_A, we define a set of Steiner points on each edge of E as SP_A. Let $G_A.V = SP_A \bigcup V$. For each node p and q in $G_A.V$, if p and q lie on the same face in F, we connect them with a weighted edge $\overline{pq} \in G_A.E$, with the weighted (surface) distance $w_{pq} \cdot d(p,q)$, where w_{pq} means the weight associated with the face or the edge that p and q lie on. In Figure 3 (b), the blue nodes are SP_{Ref} and SP_{Roug}, the blue and gray nodes are $G_{Ref}.V$ and $G_{Roug}.V$, the orange lines are the weighted edge with weighted distance 5 in $G_{Roug}.E$ and $G_{Ref}.E$.

(ii) **The removing value**: It is a constant (denoted by k and usually set to 2) used for calculating SP_{Roug}. In Figure 3 (b),

we have a set of Steiner points SP_{Ref}. When moving from v_1 to v_2, if we encounter a Steiner point, we iteratively keep one and remove the next $k = 1$ point(s). We repeat it for all edges in E to obtain a set of remaining Steiner points SP_{Roug}.

(iii) **The node information**: It is calculated in algorithm *Roug*, and is used to reduce the running time in algorithm *Ref*. In Dijkstra's algorithm, given a source node s, a set of nodes $G_A.V$ where A can be *Roug* or *Ref*, for each $u \in G_A.V$, we define $dist_A(u)$ to be the weighted shortest distance from s to u, define $prev_A(u)$ to be the previous node of u along the weighted shortest path from s to u. Give s, after running algorithm *Roug*, *node information* stores $dist_{Roug}(u)$ and $prev_{Roug}(u)$ (based on s) for each $u \in G_{Roug}.V$. In Figure 3 (c), suppose that the weight of this face is 1. After running algorithm *Roug*, the weighted shortest distance from s to a is 9, the Euclidean distance of the orange edge is 5, the weighted shortest path from s to b is $s \to a \to b$, so we have $dist_{Roug}(b) = 9 + 1 \times 5 = 14$ and $prev_{Roug}(b) = a$ as node information of b.

(iv) **The full or non-full edge sequence**: Given an edge sequence S, if the length of each edge in S is larger than 0 (resp. there exists at least one edge in S whose length is 0), then S is a *full* (resp. *non-full*) *edge sequence* or S is *full* (resp. *non-full*). In Figure 5 (a), given the path in the purple dashed line and orange line between s and t, the edge sequence S_1 and S_2 passed by the two paths are full and non-full, since the length of each edge in S_1 is larger than 0, while the edge length at ϕ_2, ϕ_3 and ϕ_4 in S_2 are 0. Figure 5 (b) has a similar case. As we will discuss later, a non-full edge sequence reduces increases algorithm *Ref*'s running time.

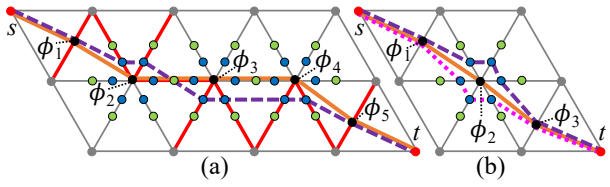

Fig. 5. Full edge sequence conversion

*2) **Overview of algorithm Roug**:* Given T, s, t, ϵ and k, we find a $(1 + \eta\epsilon)$-approximate rough path between s and t, and calculate the node information for s in two steps:

(i) η **calculation**: In Figure 3 (a), given T, ϵ and k, we first use ϵ and an efficient Steiner point placement scheme to get SP_{Ref}, use k to remove some Steiner points and get SP_{Roug} in Figure 3 (b), and then use SP_{Roug} to calculate $\eta\epsilon$ (using constrained programming and the reverse technique of calculating SP_{Ref} with ϵ).

(ii) **Rough path calculation**: In Figure 3 (c), given T, s, t and SP_{Roug}, we use Dijkstra's algorithm on G_{Roug} constructed by SP_{Roug} and V (we must include V in SP_{Roug} for error guarantee) to calculate a $(1 + \eta\epsilon)$-approximate rough path between s and t in Figure 3 (d) (denoted by $\Pi_{Roug}(s,t)$, i.e., the orange dashed line), and store $dist_{Roug}(u)$ and $prev_{Roug}(u)$ (based on s) for each $u \in G_{Roug}.V$ as node information.

*3) **Overview of algorithm Ref**:* Given T, s, t, ϵ, $\Pi_{Roug}(s,t)$ and the node information, we refine $\Pi_{Roug}(s,t)$ and calculate a $(1+\epsilon)$-approximate weighted shortest path in four steps:

(i) **Full edge sequence conversion**: In Figure 3 (d), given $\Pi_{Roug}(s,t)$ whose corresponding edge sequence S' is non-full (the edge length at v is 0), we progressively convert it to a modified rough path in Figure 3 (e) (denoted by $\Pi_{Ref\text{-}1}(s,t)$, i.e., the orange dashed line) whose corresponding edge sequence $S = (e_1, e_2, e_3, e_4)$ (edges in red) is full.

(ii) **Snell's law path refinement**: In Figure 3 (f), given T, s, t and S, we use Snell's law and S to efficiently get a refined path (denoted by $\Pi_{Ref\text{-}2}(s,t)$, i.e., the green line) on S.

(iii) **Path checking**: In Figure 3 (g), given $\Pi_{Roug}(s,t)$, $\Pi_{Ref\text{-}2}(s,t)$, ϵ and η, if $|\Pi_{Ref\text{-}2}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s,t)|$, since we guarantee $|\Pi_{Roug}(s,t)| \leq (1+\eta\epsilon)|\Pi^*(s,t)|$ due to the error bound of Dijkstra's algorithm (see Theorem 3.1 of study [14]), we have $|\Pi_{Ref\text{-}2}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, and we return $\Pi_{Ref\text{-}2}(s,t)$. Otherwise, we need the next step.

(iv) **Error guaranteed path refinement**: In Figures 3 (h), given T, s, t, SP_{Ref} and the node information (based on s), we use Dijkstra's algorithm on G_{Ref} constructed by SP_{Ref} and V to efficiently calculate a $(1+\epsilon)$-approximate weighted shortest path between s and t in Figure 3 (i) (denoted by $\Pi_{Ref\text{-}3}(s,t)$, i.e., the purple line). We can guarantee $|\Pi_{Ref\text{-}3}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$ due to the error bound of Dijkstra's algorithm.

B. Key Ideas of Rough-Refine Concept

Algorithm *Roug-Ref* is efficient due to the rough-refine concept, which involves the following three techniques.

*1) **Novel Steiner point pruning step** (in the η calculation step):* In Figure 3 (b), we prune out some Steiner points in SP_{Ref} using k, and use the remaining Steiner points SP_{Roug} for the rough path calculation, i.e., $SP_{Roug} \subseteq SP_{Ref}$ and $G_{Roug}.V \subseteq G_{Ref}.V$. The pruned Steiner points are transferred to the error guaranteed path refinement step, and our experimental results show that it is very likely after the Snell's law path refinement, the path checking finds that there is no need to perform Dijkstra's algorithm on these pruned Steiner points.

*2) **Efficient reduction of the search area** (in the rough path calculation step):* In Figure 3 (d), the edge sequence S' passed by $\Pi_{Roug}(s,t)$ is used in the Snell's law path refinement step. If we do not know S', we need to try Snell's law on different combinations of edges in E (so the search area is large) and select the shortest result path, which is time-consuming.

*3) **Usage of Snell's law** (in the Snell's law path refinement step):* In Figure 3 (e), we utilize Snell's law on S' to efficiently calculate $\Pi_{Ref\text{-}2}(s,t)$, such that the distance between the intersection point of $\Pi_{Ref\text{-}2}(s,t)$ on each edge of S' and that of $\Pi^*(s,t)$ is smaller than an error value $\delta = \frac{h\epsilon w}{6lW}$, and there is no need to place many Steiner points on edges in E.

C. Key Ideas of Additional Techniques

We also have the following four additional novel techniques to make algorithm *Roug-Ref* more efficient.

*1) **Efficient Steiner point placement scheme with a $(1+\epsilon)$ error bound** (in the η calculation step):* In Figures 3 (b) and 4 (b), we have an efficient Steiner point placement scheme (i) by considering the additional geometric information of T, such

as face weight, internal angle, and edge length, etc., so that we can place *different* numbers of Steiner points on different edges of E (for minimizing the total number of Steiner points), and (ii) by using mathematical transformations to express the error in terms of $(1+\epsilon)$ (so that with the η calculation step, the rough path has an error $(1+\eta\epsilon)$).

2) ***Novel full edge sequence conversion technique using progressive approach*** *(in the full edge sequence conversion step):* In Figure 3 (d), S' passed by $\Pi_{Roug}(s,t)$ is non-full at vertex v. If we use Snell's law on S' to calculate a refined path, after the path exits v, we do not know where it goes next, so we need to add *all* the edges connected to v n S' (edges in blue) for error guarantees, and try Snell's law on different combinations of edge sequences to select the shortest result path. Indeed, only a subset of these edges is required. To solve it, we need the full edge sequence conversion step.

Illustration: In Figure 5 (a), given a rough path $(s,\phi_1,\phi_2,\phi_3,\phi_4,\phi_5,t)$ (i.e., the orange line) whose corresponding edge sequence is non-full, we divide it into a smaller segment $(\phi_1,\phi_2,\phi_3,\phi_4,\phi_5)$ that all the edges passed by this segment have length equal to 0, i.e., at ϕ_2, ϕ_3 and ϕ_4. We add more Steiner points to *progressively* find a new path segment, i.e., the purple dashed line between ϕ_1 and ϕ_5, until the edge sequence (in red) passes by this path segment is full. If the distance of the new path segment is smaller than that of the original one, we replace the new one with the original one, and obtain a modified rough path (i.e., the purple dashed line between s and t). Figure 5 (b) shows a similar example.

3) ***Novel effective weight pruning technique*** *(in the Snell's law path refinement step):* In Figure 3 (f), we use Snell's law to find optimal intersection points on each edge of S, and then connect them to form $\Pi_{Ref\text{-}2}(s,t)$. The basic idea is to use binary search, but we can efficiently prune out some checking by utilizing *effective weight* information on T.

Illustration: In Figure 6 (a), we select the midpoint m_1 on e_1, and trace a blue light ray that follows Snell's law from s to m_1. We use binary search to adjust the position of m_1 to the left or right by checking whether t is on the left or right of this ray, and repeat until the ray passes the entire S, i.e., the purple ray from s to m_1^1. In Figure 6 (b), we regard all the faces in $F(S)$ except f_1 as one *effective face* $\triangle u_p p_1 q_1$, and use the ray in the purple line for calculating its effective weight. Then, we can calculate the position of m_{ef} on e_1 in one quartic equation using the weight of f_1 and the effective weight of $\triangle u_p p_1 q_1$. In Figure 6 (c), we trace the ray starting from s and passing m_{ef} (i.e., the dark blue line). We iterate the midpoint selection step until (i) the ray hits t or (ii) the distance between the new m_1 and the previous m_1 is smaller than δ. In Figure 6 (d), we continue on other edges in S.

4) ***Novel error guaranteed path refinement using pruning technique*** *(in the error guaranteed path refinement step):* In Figure 3 (g), for the path checking step, when $k \leq 2$, only in a rare case (i.e., the edge sequence passed by the rough path differs from that of the optimal weighted shortest path) which never occurs in our experiments, we need the error guaranteed path refinement step in Figures 3 (h) and (i). When k is larger, more Steiner points are removed, and η is larger, so the chance that $|\Pi_{Ref\text{-}2}(s,t)| > \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s,t)|$ becomes larger, we need this step to ensure error guarantee and hope that the running time of algorithm *Roug-Ref* will not increase a lot, by using the node information.

Illustration: (i) In the rough path calculation step, see Figure 3 (c), we have $dist_{Roug}(b) = 9+1\times 5 = 14$ and $prev_{Roug}(b) = a$ as node information of b. (ii) In the error guaranteed path refinement step, see Figure 3 (h), since $G_{Roug}.V \subseteq G_{Ref}.V$, we know $dist_{Ref}(b) = 14$ and $prev_{Ref}(b) = a$. We maintain a *priority queue* [7] $Q = \{\{u_1, dist_{Ref}(u_1)\},\dots\}$ in Dijkstra's algorithm on G_{Ref} that stores a set of nodes $u_i \in G_{Ref}.V$ waiting for processing. Suppose we need to process a and c, so Q stores $\{\{a,9\},\{c,8\}\}$. We dequeue c with a shorter distance value ($8<9$), and one adjacent node of c is b. Since $dist_{Ref}(b) = 14 < dist_{Ref}(c) + w_{bc}\cdot d(b,c) = 8 + 1\times 7 = 15$, we do not need to insert b and $dist_{Ref}(b) = 15$ into the queue for time-saving. Since $G_{Ref}.V$ contains $G_{Roug}.V$ and the removed Steiner points, the running time by performing Dijkstra's algorithm on G_{Ref} without the node information is the same as the time of the rough path calculation step plus the time of error guaranteed path refinement step.

D. Implementation Details of Algorithm Roug

We give implementation details of algorithm *Roug* (see Figures 3 (a) to (d)). We mainly discuss our efficient Steiner points placement scheme with a $(1+\epsilon)$ error bound, (i.e., the technique in algorithm *LogSP-Adp*).

Detail: Given a vertex v in V, we let h_v be the minimum height starting from v on the faces containing v, let C_v be a *sphere* centered at v with radius $r_v = \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4}\cdot h_v$, and let θ_v be the angle between any two edges of T that are incident to v. In Figure 4 (b), there is a sphere C_{v_1} centered at v_1, with the radius r_{v_1} and the angle θ_{v_1} of v_1. For each vertex v of face f_i, we place Steiner points $p_1, p_2, \dots, p_{\tau_p-1}$ on two edges of f_i incident to v, such that $|\overline{vp_j}| = r_v\lambda^{j-1}$, where $\tau_p = \log_\lambda \frac{|e_p|}{2\cdot r_v}$ for every integer $2 \leq j \leq \tau_p - 1$, and $\lambda = (1+\frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4}\cdot\sin\theta_v)$. Algorithm *LogSP* has $r_v = \epsilon'\cdot h_v$ and $\lambda = (1+\epsilon'\cdot\sin\theta_v)$. Since we adapt *LogSP* to be *LogSP-Adp* by setting $\epsilon' = \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4}$, we obtain r_v and λ w.r.t. ϵ.

E. Implementation Details of Algorithm Ref

We give implementation details of algorithm *Ref*.

1) ***Full edge sequence conversion****:* See Figures 3 (e) and 5.

Detail and example: A point is said to be *on the edge* (resp. *vertex*) if it lies in the internal of an edge in E (resp. it lies on the vertex in V). In both Figures 5 (a) and (b), ϕ_1 is on the edge, and ϕ_2 is on the vertex. Algorithm 1 shows this step, and the following is an example.

(i) *Successive point*: Lines 4-14, see Figure 5 (a) with the orange path as input. $v_s = \phi_1$ is the *start vertex* and $v_n = \phi_5$ is the *end vertex*. The blue points are the new Steiner points. The orange line and purple dashed line between ϕ_1 and ϕ_5 are $\Pi_{Roug}(v_s,v_e)$ and $\Pi'_{Ref\text{-}1}(v_s,v_e)$.

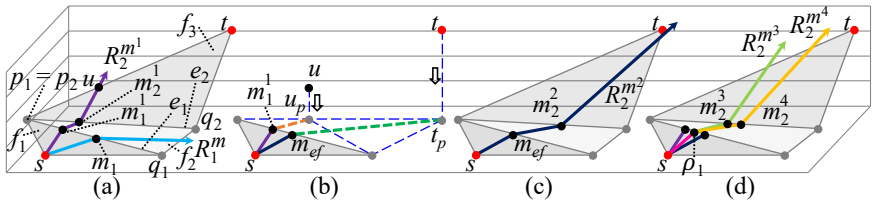

Fig. 6. Snell's law path refinement step in *Ref* (a) with initial ray for calculating effective weight on the effective face $\triangle u_p p_1 q_1$, (b) for calculating m_{ef}, (c) with final ray passing through m_{ef}, and (d) processing on the remaining edges

TABLE II
DATASETS

Name	$\lvert F \rvert$
Original dataset	
BearHead (*BH*) [25], [26]	280k
EaglePeak (*EP*) [25], [26]	300k
SeaBed (*SB*) [5]	2k
ValaisSwitzerland (*VS*) [2]	2k
PathAdvisor (*PA*) [29]	1k
Small-version dataset	
BH small-version (*BH-small*)	3k
EP small-version (*EP-small*)	3k
Multi-resolution dataset	
Multi-resolution of *BH*	1M, 2M, 3M, 4M, 5M
Multi-resolution of *BH-small*	10k, 20k, 30k, 40k, 50k
Multi-resolution of *EP*	1M, 2M, 3M, 4M, 5M
Multi-resolution of *EP-small*	10k, 20k, 30k, 40k, 50k

Algorithm 1 *EdgSeqConv* ($\Pi_{Roug}(s,t)$, ζ)

Input: the rough path $\Pi_{Roug}(s,t)$ and ζ (a constant and normally set as 10)
Output: the edge sequence S of the modified rough path $\Pi_{Ref\text{-}1}(s,t)$

$v_s \leftarrow NULL, v_e \leftarrow NULL, E' \leftarrow \emptyset, \Pi_{Ref\text{-}1}(s,t) \leftarrow \Pi_{Roug}(s,t)$
for each point v that $\Pi_{Roug}(s,t)$ intersects with an edge in E (except s and t), such that v is on the vertex **do**
  $v_c \leftarrow v$, $v_n \leftarrow v_c.next$, $v_p \leftarrow v_c.prev$
  if v_n is on the vertex and v_p is on the edge **then**
    $v_s \leftarrow v_c$, $E' \leftarrow E' \cup$ edges with v_c as one endpoint
  else if both v_n and v_p are on the edge **then**
    $E' \leftarrow E' \cup$ edges with v_c as one endpoint
  else if v_n is on the edge and v_n is on the vertex **then**
    $v_e \leftarrow v_n$, $E' \leftarrow E' \cup$ edges with v_c as one endpoint
    add new Steiner points at the midpoints between the vertices and original Steiner points on E'
    $\Pi'_{Roug}(v_s, v_e) \leftarrow$ path calculated using Dijkstra's algorithm on the weighted graph constructed by these new Steiner points and V
    $\Pi'_{Ref\text{-}1}(v_s, v_e) \leftarrow$ *EdgSeqConv* ($\Pi'_{Roug}(v_s, v_e)$, ζ)
    if $|\Pi'_{Ref\text{-}1}(v_s, v_e)| < |\Pi_{Roug}(v_s, v_e)|$ **then**
      $\Pi_{Ref\text{-}1}(s,t) \leftarrow \Pi_{Ref\text{-}1}(s, v_s) \cup \Pi'_{Ref\text{-}1}(v_s, v_e) \cup \Pi_{Ref\text{-}1}(v_e, t)$
  else if both v_p and v_n are on the edge **then**
    for $i \leftarrow 1$ to ζ **do**
      add new Steiner points at the midpoints between v_c and the nearest Steiner points of v_c on the edges that adjacent to v_c
      $\Pi_B(v_p, v_n) \leftarrow$ path passes the set of newly added Steiner points on the X side of the path (v_p, v_c, v_n), where $B = \{left, right\}$
      if $|\Pi_B(v_p, v_n)| < |\Pi_{Roug}(v_p, v_n)|$ **then**
        $\Pi_{Ref\text{-}1}(s,t) \leftarrow \Pi_{Ref\text{-}1}(s, v_p) \cup \Pi_B(v_p, v_n) \cup \Pi_{Ref\text{-}1}(v_n, t)$
        break
return the edge sequence S of $\Pi_{Ref\text{-}1}(s,t)$ based on T

Algorithm 2 *SneLawRef* (s, t, δ, S)

Input: source s, destination t, user parameter δ and edge sequence S
Output: the refined path $\Pi_{Ref\text{-}2}(s,t)$

1: $\Pi_{Ref\text{-}2}(s,t) \leftarrow \{s\}$, $root \leftarrow s$
2: **for** each $e_i \in S$ with $i \leftarrow 1$ to $|S|$ **do**
3: $a_i \leftarrow e_i$ left endpoint, $b_i \leftarrow e_i$ right endpoint, $[a_i, b_i] \leftarrow$ an interval
4: **for** each $e_i \in S$ with $i \leftarrow 1$ to $|S|$ **do**
5: **while** $|a_i b_i| \geq \delta$ **do**
6: $m_i \leftarrow$ midpoint of $[a_i, b_i]$ and calculate a surface ray with $\Pi_m = (root, m_i, \ldots, m_g, R_g^m)$ with $g \leq l$
7: **if** Π_m does not pass the whole S, i.e., $g < l$ **then**
8: **if** e_{g+1} is on the left (resp. right) side of R_g^m **then**
9: $[a_j, b_j] \leftarrow [a_j, m_j]$ (resp. $[m_j, b_j]$) for each $j \leftarrow i$ to g
10: **else if** Π_m passes the whole S, i.e., $g = l$ **then**
11: **if** t is on R_g^m **then**
12: $\Pi_{Ref\text{-}2}(s,t) \leftarrow \Pi_{Ref\text{-}2}(s,t) \cup \{m_i, \ldots, m_g, t\}$
13: **return** $\Pi_{Ref\text{-}2}(s,t)$
14: **else if** t is on the left (resp. right) side of R_g^m **then**
15: $[a_j, b_j] \leftarrow [a_j, m_j]$ (resp. $[m_j, b_j]$) for each $j \leftarrow i$ to g
16: **if** have not used effective weight pruning on e_i **then**
17: $u \leftarrow$ the intersection point between R_l^m and one of the two edges that are adjacent to t in the last face f_{l+1} in $F(S)$
18: $u_p \leftarrow$ projected point of u on the first face f_1 in $F(S)$
19: $f_{ef} \leftarrow$ *effective face* contains all faces in $F(S) \setminus \{f_1\}$
20: $w_{ef} \leftarrow$ *effective weight* for f_{ef}, calculated using $\overline{sm_i}$, $\overline{m_i u_p}$, f_1, f_{ef}, w_1 (the weight for f_1) and Snell's law
21: $t_p \leftarrow$ the projected point of t on f_1
22: $m_{ef} \leftarrow$ effective intersection point on e_1, calculated using w_1, w_{ef}, s, t_p and Snell's law in a quartic equation
23: $m_i \leftarrow m_{ef}$, compute $\Pi_m = (root, m_i, \ldots, m_g, R_g^m)$
24: update $[a_j, b_j]$ for each $j \leftarrow i$ to g same as in lines 11-15
25: $\rho_i \leftarrow [a_i, b_i]$ midpoint, $\Pi_{Ref\text{-}2}(s,t) \leftarrow \Pi_{Ref\text{-}2}(s,t) \cup \{\rho_i\}$, $root \leftarrow \rho_i$
26: $\Pi_{Ref\text{-}2}(s,t) \leftarrow \Pi_{Ref\text{-}2}(s,t) \cup \{t\}$
27: **return** $\Pi_{Ref\text{-}2}(s,t)$

(ii) *Single point*: Lines 15-21, see Figure 5 (b) with the orange path as input. The blue points are new Steiner points. The orange, purple line-dashed and pink dot-dashed lines between ϕ_1 and ϕ_3 are $\Pi_{Roug}(v_p, v_n)$, $\Pi_{left}(v_p, v_n)$ and $\Pi_{right}(v_p, v_n)$.

*2) **Snell's law path refinement**:* See Figures 3 (f) and 6.

Detail and example: Let $\Pi_{Ref\text{-}2}(s,t) = (s, \rho_1, \ldots, \rho_l, t)$, where ρ_i for $i \in \{1, \ldots, l\}$ is a point on an edge in E. Given S, s and a point c_1 on $e_1 \in S$, we can obtain a *surface ray* $\Pi_c = (s, c_1, \ldots, c_g, R_g^c)$ starting from s, hitting c_1 and following Snell's law on S, where $1 \leq g \leq l$, each c_i for $i \in \{1, \ldots, g\}$ is an intersection point in Π_c, and R_g^c is the last out-ray at $e_g \in S$. In Figure 6 (a), $\Pi_m = (s, m_1, R_1^m)$ in blue line does not pass the whole $S = (e_1, e_2)$, but $\Pi_{m^1} = (s, m_1^1, m_2^1, R_2^{m^1})$ in purple line passes the whole S. Algorithm 2 shows this step, and the following is an example.

(i) *Binary search initial path finding*: Lines 6-15, see Figure 6 (a). In lines 6-9, we first have the blue ray $\Pi_m = (s, m_1, R_1^m)$ that does not pass the whole S, we set $[a_1, b_1] = [p_1, m_1]$. In lines 10-15, we then have the purple ray $\Pi_{m^1} = (s, m_1^1, m_2^1, R_2^{m^1})$ passes the whole S, we set $[a_1, b_1] = [m_1^1, m_1]$ and $[a_2, b_2] = [m_2^1, q_2]$.

(ii) *Effective weight pruning*: Lines 16-24. The purple ray passes the whole S for the first time, we can use effective weight pruning. In line 17 and Figure 6 (a), we get u. In lines 18-22 and Figure 6 (b), we (1) get u_p and $f_{ef} = \triangle u_p p_1 q_1$, (2) calculate w_{ef} using purple line $\overline{sm_1^1}$, the orange dashed line $\overline{m_1^1 u_p}$, f_1, f_{ef}, w_1 and Snell's law, (3) get t_p, (4) set m_{ef} to be unknown and use Snell's law in vector form [3], build a quartic equation with unknown at the power of four using w_1, w_{ef}, the dark blue line $\overline{sm_{ef}}$ and the green dashed line $\overline{m_{ef} t_p}$, and use

root formula [16] to solve m_{ef}. In lines 23-24 and Figure 6 (c), we compute the dark blue ray $\Pi_{m^2} = (s, m_{ef}, m_2^2, R_2^{m^2})$, and set $[a_1, b_1] = [m_1^1, m_{ef}]$ and $[a_2, b_2] = [m_2^1, m_2^2]$.

(iii) *Binary search refined path finding*: Lines 2, 6-15, 25-26, see Figure 6 (d). In lines 6-15, we iterate until $|a_1 b_1| = |m_1^1 m_{ef}| < \delta$. In line 25, we have ρ_1, the pink dashed link $\Pi_{Ref\text{-}2}(s,t) = (s, \rho_1)$, and $root = \rho_1$. In line 2, we iterate to obtain the green ray $\Pi_{m^3} = (\rho_1, m_2^3, R_2^{m^3})$ and the yellow ray $\Pi_{m^4} = (\rho_1, m_2^4, R_2^{m^4})$. Until we process all the edges in $S = (e_1, e_2)$, we get result path $\Pi_{Ref\text{-}2}(s,t) = (s, \rho_1, \rho_2, t)$.

*3) **Error guaranteed path refinement**:* See Figure 3 (h).

Detail and example: Algorithm 3 shows this step, and the following is an example.

Algorithm 3 *ErrGuarRef* $(s, t, SP_{Ref}, NodeInfo)$

Input: source s, destination t, Steiner points SP_{Ref}, node information $dist_{Roug}(u)$ and $prev_{Roug}(u)$ (based on s) for each $u \in G_{Roug}.V$
Output: the refined path $\Pi_{Ref\text{-}3}(s,t)$
1: build a weighted graph G_{Ref} using SP_{Ref}, enqueue $\{s, 0\}$ into Q
2: **for** each $u \in G_{Ref}.V$ **do**
3: **if** $u \in G_{Ref}.V \setminus G_{Roug}.V$ (resp. $u \in G_{Roug}.V$) **then**
4: $dist_{Ref}(u) \leftarrow \infty$ (resp. $dist_{Roug}(u)$)
5: $prev_{Ref}(u) \leftarrow NULL$ (resp. $prev_{Roug}(u)$)
6: **while** Q is not empty and the to-be-dequeued node is not t **do**
7: dequeue node v from Q with smallest distance value $dist_{Ref}(v)$
8: **for** each adjacent vertex v' of v, such that $\overline{vv'} \in G_{Ref}.E$ **do**
9: **if** $dist_{Ref}(v') > dist_{Ref}(v) + w_{vv'} \cdot d(v, v')$ **then**
10: $dist_{Ref}(v') \leftarrow dist_{Ref}(v) + w_{vv'} \cdot d(v, v')$, $prev_{Ref}(v') \leftarrow v$
11: enqueue $\{v', dist_{Ref}(v')\}$ into Q
12: $u \leftarrow prev_{Ref}(t)$, $\Pi_{Ref\text{-}3}(s,t) \leftarrow \{t\}$
13: **while** $u \neq s$ **do**
14: $\Pi_{Ref\text{-}3}(s,t) \leftarrow \Pi_{Ref\text{-}3}(s,t) \cup \{u\}$, $u \leftarrow prev_{Ref}(u)$
15: $\Pi_{Ref\text{-}3}(s,t) \leftarrow \Pi_{Ref\text{-}3}(s,t) \cup \{s\}$, reverse $\Pi_{Ref\text{-}3}(s,t)$
16: **return** $\Pi_{Ref\text{-}3}(s,t)$

(i) *Distance and previous node initialization*: Lines 2-5. For d, $dist_{Ref}(d) = \infty$ and $prev_{Ref}(d) = NULL$; for b, $dist_{Ref}(b) = dist_{Roug}(b) = 9 + 1 \times 5 = 14$ and $prev_{Ref}(b) = prev_{Roug}(b) = a$.

(ii) *Priority queue looping*: Lines 6-11. Suppose Q stores $\{\{a, 9\}, \{c, 8\}\}$, we dequeue c. One adjacent node of c is b, since $dist_{Ref}(b) = 14 < dist_{Ref}(c) + w_{bc} \cdot d(b, c) = 8 + 1 \times 7 = 15$, there is no need to enqueue $\{b, dist_{Ref}(b) = 15\}$ into Q.

(iii) *Path retrieving*: Lines 16-19. We obtain $\Pi_{Ref\text{-}3}(s,t)$.

F. Theoretical Analysis

Theorem 1 shows the analysis of algorithm *Roug-Ref*.

Theorem 1. *The running time and memory usage for algorithm Roug-Ref is $O(n \log n + l)$ and $O(n + l)$. It guarantees that $|\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$.*

Proof Sketch. (1) The *running time* contains (i) $O(n \log n)$ for the rough path calculation due to Dijkstra's algorithm on G_{Roug} with n nodes, (ii) $O(n \log n)$ for the full edge sequence conversion due to Dijkstra's algorithm, (iii) $O(l)$ for the Snell's law path refinement due to l edges in S and $O(1)$ time in finding the optimal intersection point on each edge, and (iv) $O(n \log n)$ for the error guaranteed path refinement due to Dijkstra's algorithm on G_{Ref} with n nodes and the node information. (2) The *memory usage* contains (i) $O(n)$ for the rough path calculation, (ii) $O(n)$ for the full edge sequence conversion, (iii) $O(l)$ for the Snell's law path refinement, and (iv) $O(n)$ for the error guaranteed path refinement. (3) The *error bound* is due to the rough path calculation, and the path checking and the error guaranteed path refinement. The detailed proof appears in our technical report [30]. □

V. Empirical Studies

A. Experimental Setup

We conducted the experiments on a Linux machine with 2.67 GHz CPU and 48GB memory. All algorithms were implemented in C++. The experimental setup followed the setup used in previous studies [12], [13], [18], [25], [26].

Datasets: We conducted our experiment on 27 real terrain datasets in Table II, where *EP* has more mountains compared with the other 4 original datasets. For small-version and multi-resolution datasets, we generated them using *BH* and *EP* following the procedure in [18], [25], [26] (which creates a terrain surface with different resolutions). We use the slope of a face in terrain datasets as the weight of that face [11].

Algorithms: We compared *Roug-Ref* with $(1+\epsilon)$-approximate algorithms, i.e., (1) *EdgSeq-Adp* [24], (2) the best-known algorithm *FixSP* [11], [15], (3) *FixSP-NoWei-Adp* [12], (4) *LogSP-Adp* [4], and (5) variations of *Roug-Ref*. We also compared with (1) *EdgSeq* [24] without error bound, (2 & 3) *LogSP* [4] and *Roug* with distance errors larger than $(1+\epsilon)$ in our technical report [30].

Query Generation: We randomly chose pairs of vertices in V as source and destination, and we report the average, minimum, and maximum results of 100 queries.

Factors and Measurements: We studied three factors, namely (1) k, (2) ϵ, and (3) dataset size (i.e., the number of faces in a terrain surface). We used five measurements to evaluate the algorithm performance, namely (1) *query time*, (2) *memory usage*, (3) *chances of using error guaranteed path refinement step*, (4) *average number of Steiner points per edge (used in path calculation)*, and (5) *distance error* (we use *EdgSeq-Adp* with $\epsilon = 0.05$ to simulate the exact result since no algorithm can solve the weighted region problem exactly).

B. Experimental Results

We compared (1) all algorithms on datasets with less than 250k faces, and (2) algorithms not involving *FixSP* components on datasets with more than 250k faces due to the expensive running time. The vertical bar means the minimum and maximum results.

*1) **Ablation study of Roug-Ref**:* In our algorithm *Roug-Ref*, we have 4 variations: (i) we do not use our efficient Steiner points placement scheme, i.e., we use algorithm *FixSP* for Steiner point placement, (ii) we remove the full edge sequence conversion step, (iii) we remove the effective weight pruning out sub-step in the Snell's law path refinement step, and (iv) we do not use the node information for pruning in the error guaranteed path refinement step, for ablation study (they correspond to four techniques in Section IV-C). We use (i) *Roug-Ref-NoEffSP*, (ii) *Roug-Ref-NoEdgSeqConv*, (iii) *Roug-Ref-NoEffWeig*, and (iv) *Roug-Ref-NoPrunDijk*, to denote these

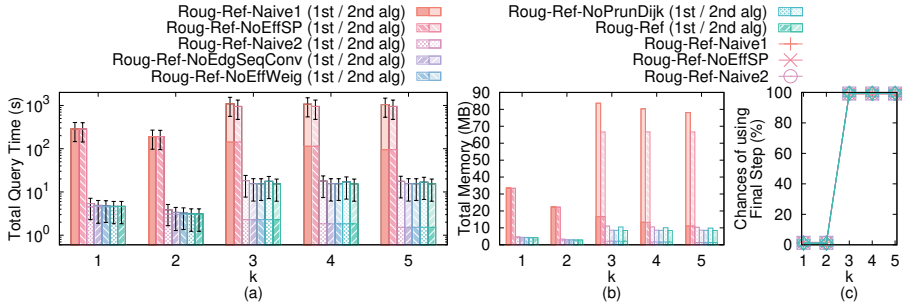

Fig. 7. Ablation study (effect of k on *BH-small* dataset)

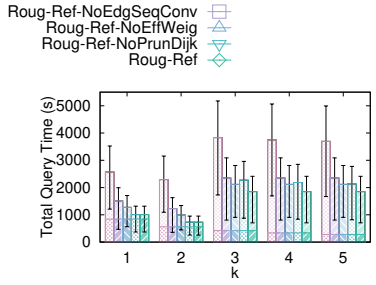

Fig. 8. Ablation study (effect of k on *BH* dataset)

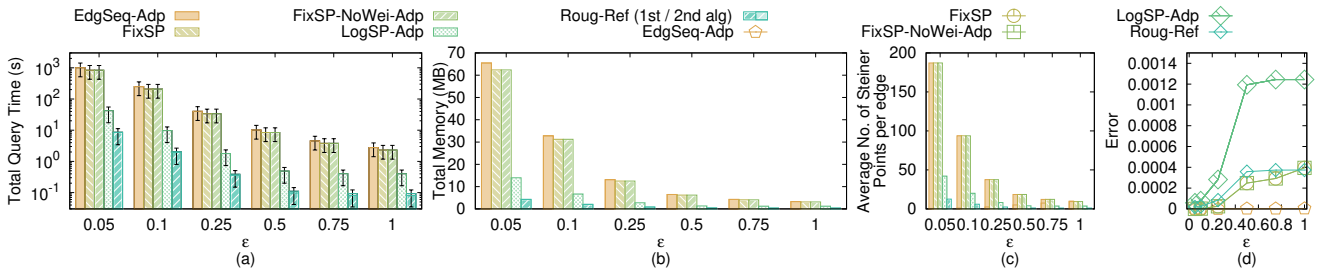

Fig. 9. Baseline comparisons (effect of ϵ on *EP-small* dataset)

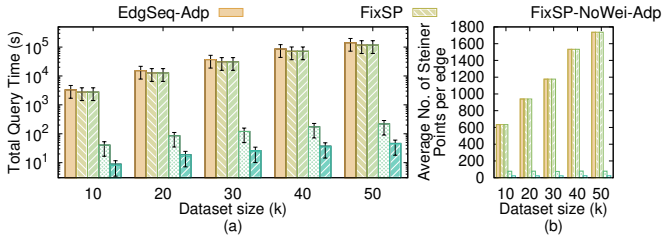

Fig. 10. Baseline comparisons (effect of dataset size on multi-resolution of *EP-small* datasets)

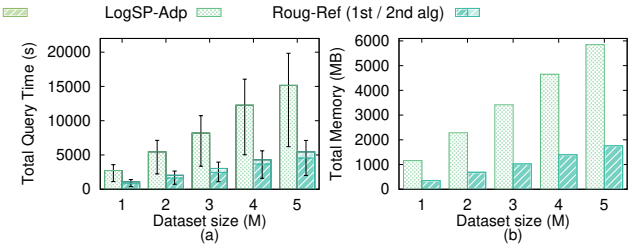

Fig. 11. Scalability test

variations. If we do not use the rough-refine concept, *Roug-Ref* becomes *LogSP-Adp*. We adapt *FixSP*, *FixSP-NoWei-Adp* and *EdgSeq-Adp* by using our rough-refine concept, and denote it as *Roug-Ref-Navie1*. We adapt *LogSP-Adp* similarly to be *Roug-Ref-Navie2*. Since k will only affect our algorithm *Roug-Ref* and its variations, we study the effect of k here.

Effect of *k*: In Figure 7 and Figure 8, we tested 5 values of k from $\{1, 2, 3, 4, 5\}$ on *BH-small* and *BH* dataset by setting ϵ to be 0.1 and 0.25 for ablation study, respectively. (i) When $k \leq 2$ and k increases, more Steiner points are removed and *Roug* runs faster, so the query time and memory usage of these algorithms decreases. But when $k > 2$, it has a higher chance (more than 99%) that *Ref* needs to perform the error guaranteed path refinement step, so the query time and memory usage have a sudden increase. Thus, the optimal k is 2 (also verified on other datasets shown in our technical report [30]). (ii) When $k = 2$, the query time of *Roug-Ref-Naive1* and *Roug-Ref-NoEffSP* are 230s and 210s on *BH-small* dataset, but their difference is small due to the log-scale in Figure 7 (a). Due to the same reason, the difference in query time of the other four algorithms on *BH-small* dataset is also small, so we compare them with linear-scale in Figure 8 (which shows the usefulness of each component).

*2) **Baseline comparisons**:* We then study the effect of ϵ and dataset size on other baselines when $k = 2$.

Effect of ϵ: In Figure 9, we tested 6 values of ϵ from $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on *EP-small* dataset. (i) When ϵ increases, fewer Steiner points are required so the query time and memory usage decrease. (ii) The query time and memory usage of *Ref* is much smaller than that of *Roug*. (iii) *Roug-Ref* performs better than all other algorithms in terms of query time and memory usage, and it is clear to observe the superior performance of *Roug-Ref*, due to the rough-refine concept and four novel techniques. The distance error of all algorithms is very small, the value is 0.0004 when $\epsilon = 1$ for *Roug-Ref*. (iv) When $\epsilon = 0.05$, *Roug-Ref* has 160 fewer Steiner points per edge than *FixSP* and *FixSP-NoWei-Adp*, the query time and calculated path's distance of *Roug-Ref* is 14.6s and 105.3m, but are both 23,800s $\approx$ 7.2 hours and 105.2m for *FixSP* and *FixSP-NoWei-Adp*, since *Roug-Ref* uses the rough-refine concept. *EdgSeq-Adp* performs worse than *FixSP* since *EdgSeq-Adp* first uses *FixSP* and then uses Snell's law for the weighted shortest path calculation. *LogSP-Adp* does not perform well since it does not utilize Snell's law.

Effect of dataset size: In Figure 10, we tested 5 values of dataset size from {10k, 20k, 30k, 40k, 50k} on multi-resolution of *EP-small* datasets by setting ϵ to be 0.1. When the dataset size is 50k, *Roug-Ref*'s query time is 10^3 times, 10^3 times, 10^3 times and 6 times smaller than that of *EdgSeq-Adp*, *FixSP*, *FixSP-NoWei-Adp* and *LogSP-Adp*, respectively.

3) ***Scalability test****:* In Figure 11, we tested 5 values of dataset size from {1M, 2M, 3M, 4M, 5M} on multi-resolution of *EP* datasets by setting ϵ to be 0.25. When the dataset size is 5M, *Roug-Ref*'s query time is still reasonable. However, the query times for *EdgSeq-Adp*, *FixSP* and *FixSP-NoWei-Adp* are larger than 7 days, so they are excluded from the figure.

4) ***Other algorithms comparisons****:* Given ϵ, *Roug-Ref* can calculate a path with distance d. But, *EdgSeq* does not have error bound, *LogSP* and *Roug* have distance errors larger than $(1+\epsilon)$. We finetune their input to make their calculated path also with distance d. When $\epsilon = 0.1$, *Roug-Ref*, *EdgSeq*, *LogSP* and *Roug* run in 3s, 140s, 113s and 110s on *BH-small* dataset.

5) ***Case study****:* We conducted a case study on the earthquake rescue in Section I-A. The rescue team can walk and drive cars with an average speed of 3km/h and 90km/h to pass through the destroyed and non-destroyed region. In Figure 1 (b), the weighted and unweighted shortest path has a distance of 4.6km and 4.3km. So, the rescue time of using the weighted and unweighted shortest paths are 3 min $\approx \frac{4.6\text{km}}{90\text{km/h}} \times 60\text{min/h}$ and 1.4 hours $\approx \frac{4.3\text{km}}{3\text{km/h}}$. The query time for the best-known algorithm *FixSP* and *Roug-Ref* are 11,900s $\approx$ 3.3 hours and 7.3s. So, the weighted shortest path calculated by *Roug-Ref* is the best rescue path.

6) ***Summary****: Roug-Ref* is up to 1630 times and 40 times better than the best-known algorithm *FixSP* in terms of running time and memory usage. When the dataset size is 50k with $\epsilon = 0.1$, *Roug-Ref*'s running time is 73s $\approx$ 1.2 min, and memory usage is 43MB, but *FixSP*'s running time is 119,000s $\approx$ 1.5 days, and memory usage is 2.9GB.

VI. Conclusion

We propose a two-step $(1+\epsilon)$-approximate algorithm *Roug-Ref* for solving the 3D weighted region problem. Future work can be introducing a new pruning step (by considering other geometric information of the weighted terrain surface) to further reduce the algorithm's running time.

Acknowledgements

We are grateful to the anonymous reviewers for their constructive comments. The research of Yinzhao Yan and Raymond Chi-Wing Wong is supported by GZSTI16EG24.

References

[1] "Japan earthquake," 2024. [Online]. Available: https://www.accuweather.com/en/weather-news/japan-earthquake-death-toll-reaches-206-as-government-includes-indirect-deaths/1611292

[2] "Moderate mag. 4.1 earthquake - 6.3 km north-east of sierre, valais, switzerland," 2024. [Online]. Available: https://www.volcanodiscovery.com/earthquakes/quake-info/1451397/mag4quake-Oct-24-2016-Leukerbad-VS.html

[3] "Snell's law in vector form," 2024. [Online]. Available: https://physics.stackexchange.com/questions/435512/snells-law-in-vector-form

[4] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack, "An ε-approximation algorithm for weighted shortest paths on polyhedral surfaces," in *Workshop on Algorithm Theory*. Springer, 1998.

[5] C. Amante and B. W. Eakins, "Etopo1 arc-minute global relief model: procedures, data sources and analysis," 2009.

[6] C. Bassem, S. Honcharuk, and M. Mokbel, "Route recommendation to facilitate carpooling," *MDM*, pp. 29–34, 2022.

[7] M. Chen, R. A. Chowdhury, V. Ramachandran, D. L. Roche, and L. Tong, "Priority queues and dijkstra's algorithm," 2007.

[8] J.-L. De Carufel, C. Grimm, A. Maheshwari, M. Owen, and M. Smid, "A note on the unsolvability of the weighted region shortest path problem," *Computational Geometry*, vol. 47, no. 7, pp. 724–727, 2014.

[9] K. Deng, X. Zhou, H. T. Shen, Q. Liu, K. Xu, and X. Lin, "A multi-resolution surface distance model for k-nn query processing," *VLDBJ*, vol. 17, no. 5, pp. 1101–1119, 2008.

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[11] B. Huang, V. J. Wei, R. C.-W. Wong, and B. Tang, "Ear-oracle: on efficient indexing for distance queries between arbitrary points on terrain surface," *SIGMOD*, vol. 1, no. 1, pp. 1–26, 2023.

[12] M. Kaul, R. C.-W. Wong, and C. S. Jensen, "New lower and upper bounds for shortest distance queries on terrains," *VLDB*, vol. 9, no. 3, pp. 168–179, 2015.

[13] M. Kaul, R. C.-W. Wong, B. Yang, and C. S. Jensen, "Finding shortest paths on terrains by killing two birds with one stone," *VLDB*, vol. 7, no. 1, pp. 73–84, 2013.

[14] M. Lanthier, "Shortest path problems on polyhedral surfaces." Ph.D. dissertation, Carleton University, 2000.

[15] M. Lanthier, A. Maheshwari, and J.-R. Sack, "Approximating shortest paths on weighted polyhedral surfaces," *Algorithmica*, vol. 30, no. 4, pp. 527–562, 2001.

[16] S. H. Lee, S. M. Im, and I. S. Hwang, "Quartic functional equations," *Journal of Mathematical Analysis and Applications*, vol. 307, no. 2, pp. 387–394, 2005.

[17] H. Li and Z. Huang, "82 die in sichuan quake, rescuers race against time to save lives," 2024. [Online]. Available: https://www.chinadailyhk.com/article/289413#82-die-in-Sichuan-quake-rescuers-race-against-time-to-save-lives

[18] L. Liu and R. C.-W. Wong, "Finding shortest path on land surface," *SIGMOD*, pp. 433–444, 2011.

[19] B. Max and W. Emil, "Principles of optics," 1959.

[20] J. E. Nichol, A. Shaker, and M.-S. Wong, "Application of high-resolution stereo satellite images to detailed landslide hazard assessment," *Geomorphology*, vol. 76, no. 1-2, pp. 68–75, 2006.

[21] C. Shahabi, L.-A. Tang, and S. Xing, "Indexing land surface for efficient knn query," *VLDB*, vol. 1, no. 1, pp. 1020–1031, 2008.

[22] A. Suppasri, K. Pakoksung, I. Charvet, C. T. Chua, N. Takahashi, T. Ornthammarath, P. Latcharote, N. Leelawat, and F. Imamura, "Load-resistance analysis: an alternative approach to tsunami damage assessment applied to the 2011 great east japan tsunami," *Natural Hazards and Earth System Sciences*, vol. 19, no. 8, pp. 1807–1822, 2019.

[23] X. Teng, G. Trajcevski, J.-S. Kim, and A. Züfle, "Semantically diverse path search," *MDM*, pp. 69–78, 2020.

[24] N. Tran, M. J. Dinneen, and S. Linz, "Close weighted shortest paths on 3d terrain surfaces," in *SIGSPATIAL*, 2020, pp. 597–607.

[25] V. J. Wei, R. C.-W. Wong, C. Long, and D. M. Mount, "Distance oracle on terrain surface," *SIGMOD*, pp. 1211–1226, 2017.

[26] V. J. Wei, R. C.-W. Wong, C. Long, D. M. Mount, and H. Samet, "Proximity queries on terrain surface," *TODS*, 2022.

[27] V. J. Wei, R. C.-W. Wong, C. Long, D. M. Mount, and H. Samet, "On efficient shortest path computation on terrain surface: A direction-oriented approach," *TKDE*, no. 1, pp. 1–14, 2024.

[28] S. Xing, C. Shahabi, and B. Pan, "Continuous monitoring of nearest neighbors on land surface," *VLDB*, vol. 2, no. 1, pp. 1114–1125, 2009.

[29] Y. Yan and R. C.-W. Wong, "Path advisor: a multi-functional campus map tool for shortest path," *VLDB*, vol. 14, no. 12, pp. 2683–2686, 2021.

[30] Y. Yan and R. C.-W. Wong, "Efficient shortest path queries on 3d weighted terrain surfaces for moving objects (technical report)," 2024. [Online]. Available: https://github.com/yanyinzhao/WeightedTerrainCode/blob/master/TechnicalReport.pdf

[31] Y. Yan and R. C.-W. Wong, "Proximity queries on point clouds using rapid construction path oracle," *SIGMOD*, vol. 2, no. 1, pp. 1–26, 2024.