Efficiently Finding Shortest Path 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.**

I. Introduction

The shortest path query for moving objects becomes increasingly widespread nowadays [6], [28], especially on terrain datasets. In industry, many companies and applications, including Metaverse and Google Earth, use terrain datasets (e.g., mountains and valleys) with different features (e.g., water and forest) to help moving users compute the shortest path to the destination. In academia, the shortest path query on terrain datasets is a prevalent research topic in the field of mobile data management [11], [13], [14], [30], [29], [31], [32], [33]. 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 [3] with an area of 20km^2) with weighted (resp. unweighted) shortest paths in the blue (resp. purple dashed) line 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. The death toll of the 7.6 magnitude earthquake on Jan 1, 2024 in Japan exceeded 200 [2]. The 4.1 magnitude earthquake on Oct 24, 2016 in Valais, Switzerland [3] caused an avalanche. In Figures 1 (a) and (b), the terrain surface consists of destroyed regions (the faces with green color) and non-destroyed regions (the faces with white color). Since the recuse time for passing through the destroyed region is larger than the non-destroyed region, we set the weight of each terrain face equal to the *damage level* [27] of each region, i.e., we set terrain faces corresponding to the destroyed (resp. non-destroyed) region with a larger (resp. smaller) weight. So, the *weighted length* of the purple dashed path between s (a recuse center) and t (a destroyed village) that passes destroyed regions is larger, and the recuse team will choose the blue path that does not pass destroyed regions (with a smaller weighted length). Our case study shows that due to the different recuse speeds for passing through different regions, the recuse time of using the weighted (resp. unweighted) shortest path is 3 min (resp. 1.1 hours). 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 [25] for a 1km^2 region, and our algorithm can calculate a weighted shortest recuse path in 3s in realistic settings.

2) ***Path Advisor****: Path Advisor* [33] is a mobile and web-based app for calculating the shortest path between two rooms in a university campus. Figure 1 (c) shows a weighted terrain surface (which represents the floor of the building) in Path Advisor. For safety, the path should maintain a minimum distance (e.g., at least 0.2 meters) from obstacles. So, with a weight-distance function [33], faces on the floor closer to the aisle boundaries (resp. centers) are assigned higher (resp. lower) weights. In this figure, 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 [30], [31]. 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

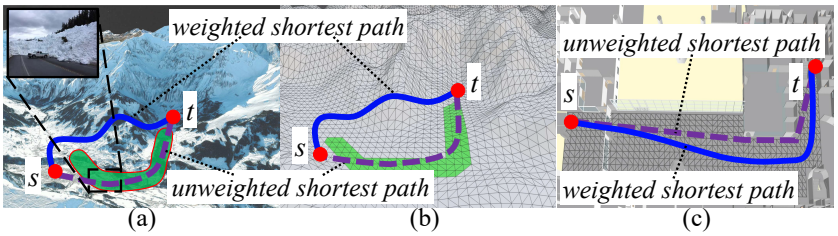

Fig. 1. Blue weighted and purple dashed unweighted shortest paths

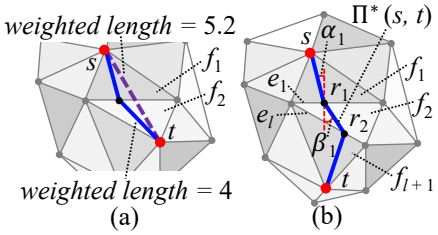

Fig. 2. Snell's law illustration

and refraction for the light satisfy *Snell's law* [23], 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 length of 4 that satisfies Snell's law is the weighted shortest path, but the weighted length 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 [10], but two categories of algorithms can solve it on-the-fly *approximately*.

1) ***Steiner point approach****:* Algorithms [4], [13], [14], [18] place discrete (i.e., Steiner) points on edges in E, and use Dijkstra's algorithm [12] on a weighted graph constructed using these Steiner points and V to calculate the result path. The best-known algorithm [13], [18] 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 [13] uses algorithm [18] 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 [13], [18] runs in 119,000s $\approx$ 1.5 days on a terrain surface with 50k faces.

2) ***Edge sequence approach****:* Algorithm [29] first uses the best-known algorithm [13], [18] to calculate a $(1+\epsilon)$-approximate weighted shortest path and an edge sequence that this path passes based on T, and then uses Snell's law on the edge sequence to calculate an even shorter result path. It runs in more than $O(n^3 \log n)$ time since it does not consider any pruning technique in the edge sequence finding.

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 (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, 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*).

In the earthquake and avalanche recuse, and Path Advisor example, we prioritize algorithm efficiency and we can tolerate a certain error. For the earthquake recuse, our calculated weighted shortest recuse path with $\epsilon = 0.05$, which implies that the recuse time of using the exact weighted shortest recuse path is at least 2.7 $(= \frac{3}{1.1})$ min, but it takes 7,900s $\approx$ 2.2 hours to calculate a path with $\epsilon = 0.01$ using the same algorithm.

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

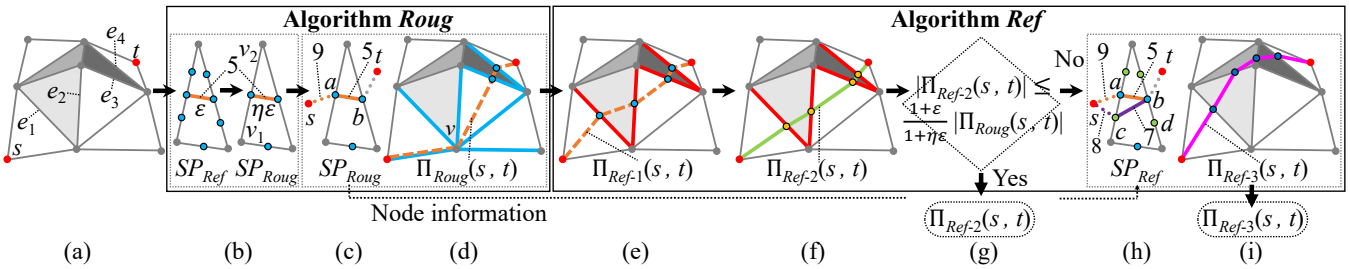

Fig. 3. Framework overview

techniques are all absent in algorithms [4], [29] and the best-known algorithm [13], [18].

(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 [13], [18] 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, but the best-known $(1+\epsilon)$-approximate algorithm [13], [18] runs in 119,000s $\approx$ 1.5 days and uses 2.9GB of memory. Our algorithm is the optimal algorithm in earthquake recuse.

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 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 [34].

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), $\Pi^*(s,t)$ stratifies 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

Recall that there are two categories of on-the-fly approximate algorithms on the weighted terrain surface.

1) ***Steiner point approach:*** Algorithms [4], [13], [14], [18] 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*) [18], [13] and algorithm *Fixed Steiner Point No Weight Adaption* (*FixSP-NoWei-Adp*) [14] (resp. algorithm *Logarithmic Steiner Point* (*LogSP*) [4]) calculates the result path with an error $(1+\epsilon)$ (resp. much larger than $(1+\epsilon)$). (i) Algorithm *FixSP*, which runs in $O(n^3 \log n)$ time, is regarded as the best-known $(1+\epsilon)$-approximate algorithm for solving the weighted region problem. (ii) Algorithm [14], which runs in $O((n+n')\log(n+n'))$ time, was originally designed for finding the unweighted shortest path, we adapt it to be algorithm *FixSP-NoWei-Adp* [14] in the weighted case, which runs in $O(n^3 \log n)$ and equivalent to algorithm *FixSP*, by assigning a weight to each face, and use the same surface distance definition in Section II-A1, where n' is the total number of Steiner points and $n' = O(n^3)$ [18]. (iii) In algorithm *LogSP*, given an error parameter $\epsilon' > 0$ that determines how far the Steiner points are placed (which is different from our ϵ, the error parameter that controls the distance error bound), it places $O(\log \frac{c}{\epsilon'})$ Steiner points per

edge 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. So, they always place a uniform number of (i.e., $O(n^2)$) Steiner points per edge to bound the error [18]. But, we utilize this information and place only $O(\log c')$ Steiner points per edge, 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*: They 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.

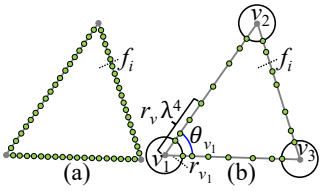

Fig. 4. Steiner points in (a) *FixSP* and *FixSP-NoWei-Adp*, and (b) *Roug-Ref*

TABLE I
ALGORITHM COMPARISONS

Algorithm	Time/Size	Error
EdgSeq [29]	Large	Small
FixSP [13], [18]	Large	Small
FixSP-NoWei-Adp [14]	Large	Small
LogSP [4]	Medium	Large
LogSP-Adp [4]	Medium	Small
***Roug-Ref* (ours)**	**Small**	**Small**

*2) **Edge sequence approach**:* Algorithm *Edge Sequence* (*EdgSeq*) [29] first employs algorithm *FixSP* to calculate a $(1+\epsilon)$-approximate weighted shortest path, then determines the edge sequence passed by this path based on T, and finally utilizes Snell's law on the edge sequence to compute an even shorter path in $O(n^3\log n + n^4\log(\frac{n^2NWL}{w\epsilon}))$ time.

Drawbacks of algorithm *EdgSeq*: It is still very slow due to three reasons. (i) *Absence of pruning technique for the edge sequence finding*: It simply 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, such that even with our efficient Steiner point placement scheme, we can further prune out more than half of the Steiner points. (ii) *Uniform number of Steiner points per edge*: It has the second drawback as of *FixSP*. (iii) *Absence of pruning technique when using Snell's law*: After finding an edge sequence, it solely relies on binary search to find a weighted shortest path using Snell's law on the edge sequence, without utilizing any weight information of T for pruning. But, we use *effective weight* information for pruning. In the same experimental setting of the previous three paragraphs, *EdgSeq* runs in 131,000s $\approx$ 1.7 days, but *Roug-Ref* runs in 73s $\approx$ 1.2 min.

Comparisons: We compare these algorithms and our algorithm in Table I. *Roug-Ref* is the best algorithm since it 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* [8]: Algorithm [8] 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* [11], [13], [26], [30], [31], [32]: Study [13] uses algorithm *FixSP* [18] (resp. studies [30], [31] use algorithm [14]) to pre-compute the weighted (resp. unweighted) shortest paths and build an oracle on a weighted (resp. unweighted) terrain surface for the shoretst path query. Studies [11], [26], [32] build oracles on an unweighted terrain surface for the k-nearest neighbor query. But, we focus on the *on-the-fly* algorithm, which is a fundamental operator for oracle construction. (3) *On-the-fly algorithms on the road network* [6], [28]: Algorithms [6], [28] answer the shortest path query on the road network, which is totally 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 $s \rightarrow a \rightarrow b$, 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 (resp. orange) line between s and t, the edge sequence passed by this path is full (resp. non-full) since the length of each edge is larger than 0 (resp. the edge length at ϕ_2, ϕ_3 and ϕ_4 are 0). Figure 5 (b) has a similar case. As we will discuss later, a full (resp. non-full) edge sequence reduces (resp. increases) algorithm *Ref*'s running time.

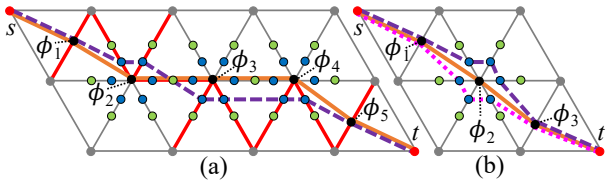

Fig. 5. Full edge sequence conversion

*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}, and then 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 the reverse technique of using ϵ to calculate SP_{Ref}).

(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 [17]), 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 (with only one edge length equal to 0).

*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, 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* [9] $Q = \{\{u_1, dist_{Ref}(u_1)\}, \dots\}$ 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. Since r_v (as the denominator) cannot be 0, we cannot set ϵ to 0.

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. 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 (resp. purple dashed) line between ϕ_1 and ϕ_5 represents $\Pi_{Roug}(v_s, v_e)$ (resp. $\Pi'_{Ref\text{-}1}(v_s, v_e)$).

(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

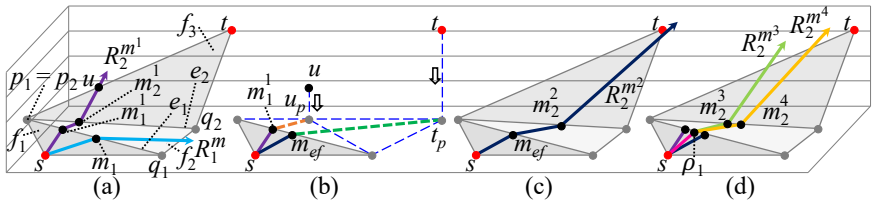

Fig. 6. Snell's law path refinement step in *Ref* (a) with initial ray for calculating effective weight on the effective face $\triangle u_p p_1 q_1$, (b) for calculating m_{ef}, (c) with final ray passing through m_{ef}, and (d) processing on the remaining edges

TABLE II
DATASETS

Name	$\lvert F \rvert$
Original dataset	
BearHead (*BH*) [30], [31]	280k
EaglePeak (*EP*) [30], [31]	300k
SeaBed (*SB*) [5]	2k
ValaisSwitzerland (*VS*) [3]	2k
PathAdvisor (*PA*) [33]	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** $\Pi_{Ref\text{-}2}(s,t)$

between ϕ_1 and ϕ_3 represent $\Pi_{Roug}(v_p, v_n)$, $\Pi_{left}(v_p, v_n)$ 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 [1], then 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 [19] 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 in Section IV-A2, and the path checking and the error guaranteed path refinement in Section IV-A3. The detailed proof appears in our technical report [34]. □

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 [14], [15], [22], [30], [31].

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 [22], [30], [31] (which creates a terrain surface with different resolutions). We use the slope of a face in terrain datasets as the weight of that face [13], due to the distinct face slope, each face has a distinct weight.

Algorithms: We mainly compared our algorithm *Roug-Ref*, and the baseline algorithms with a distance error $(1+\epsilon)$, i.e., (1) *EdgSeq* [29], (2) the best-known algorithm *FixSP* [13], [18], (3) *FixSP-NoWei-Adp* [14], and (4) *LogSP-Adp* [4] in the experiment. We also compared variations of *Roug-Ref* for the ablation study, and compared *Roug-Ref* with *LogSP* [4] and *Roug* with a distance error larger than $(1+\epsilon)$ for completeness.

Query Generation: We randomly chose pairs of vertices in V as source and destination (the corresponding path has different distances and covers different faces with different weights), 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* 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

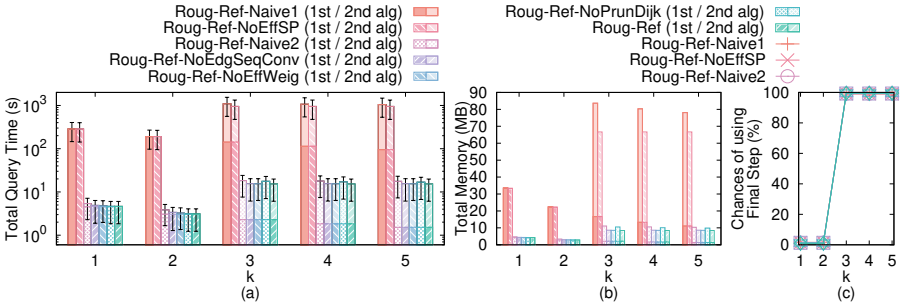

Fig. 7. Ablation study (effect of k on *BH-small* dataset)

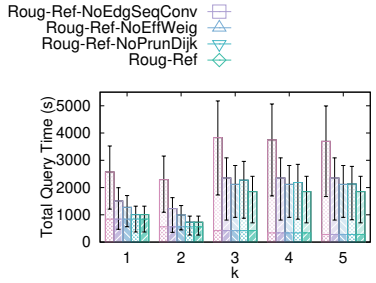

Fig. 8. Ablation study (effect of k on *BH* dataset)

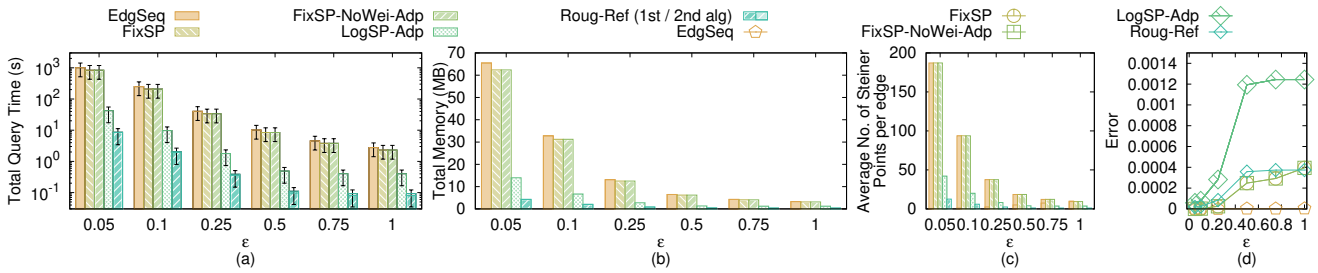

Fig. 9. Baseline comparisons (effect of ϵ on *EP-small* dataset)

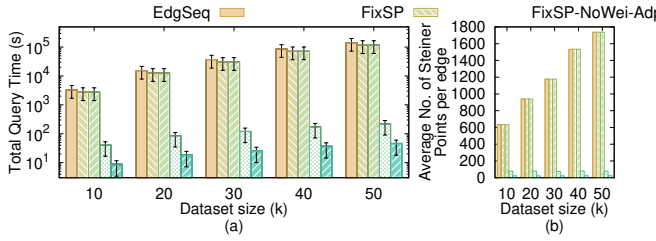

Fig. 10. Baseline comparisons (effect of dataset size on multi-resolution of *EP-small* datasets)

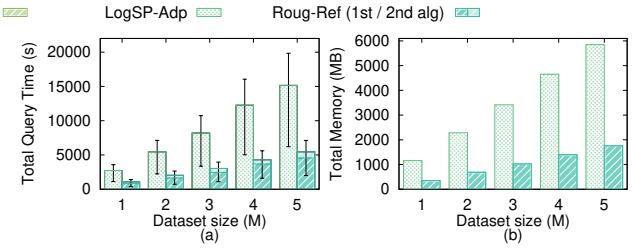

Fig. 11. Scalability test

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

Effect of *k*: In Figure 7 (resp. Figure 8), we tested 5 values of k from $\{1, 2, 3, 4, 5\}$ on *BH-small* (resp. *BH*) dataset by setting ϵ to be 0.1 (resp. 0.25) for ablation study (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 [34]). (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 clearer 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* performs worse than *FixSP*

since *EdgSeq* 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*, *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*, *FixSP* and *FixSP-NoWei-Adp* are larger than 7 days, so they are excluded from the figure.

4) ***Large distance error algorithm comparisons****:* Given ϵ, *Roug-Ref* can calculate a path with distance d. But, *LogSP* and *Roug* have distance errors larger than $(1+\epsilon)$. We finetune their input error to make their calculated path also with distance d. We compare them in our technical report [34]. *Roug-Ref*, *LogSP* and *Roug* run in 4s, 160 and 120s when $\epsilon = 0.1$ on *BH-small* dataset due to the rough-refine concept in *Roug-Ref*.

5) ***Case study****:* We conducted a case study on the earthquake as mentioned in Section I-A. The rescue teams can drive cars (resp. walk) with an average speed 90km/h (resp. 3km/h) to pass through non-destroyed (resp. destroyed) regions. In Figure 1 (b), the weighted (resp. unweighted) shortest path has distance 4.6km (resp. 4.3km). So, the recuse times of using these two paths are 3 min ($\approx \frac{4.6\text{km}}{90\text{km/h}} \times 60\text{min/h}$) and 1.1 hours ($\approx \frac{4.3\text{km}}{4\text{km/h}}$). The query time for the best-known algorithm *FixSP* and *Roug-Ref* are 11,900s $\approx$ 3.3 hours and 7.3s. Since a rescue team can save 1 life every 5 minutes [20], the recuse team can arrive at the destroyed village in 3 min($\approx 3\text{min} + 7.3\text{s}$) using the weighted shortest path calculated by *Roug-Ref* to recuse more lives.

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. The user study confirms that *Roug-Ref* is the optimal algorithm in earthquake recuse.

VI. Conclusion

We propose a two-step $(1+\epsilon)$-approximate algorithm *Roug-Ref* for solving the 3D weighted region problem, which runs up to 1630 times faster than the best-known algorithm. 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.

References

[1] "Snell's law in vector form," 2023. [Online]. Available: https://physics.stackexchange.com/questions/435512/snells-law-in-vector-form

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

[3] "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

[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," in *MDM*. IEEE, 2022, pp. 29–34.

[7] C. Bueger, T. Liebetrau, and J. Franken, "Security threats to undersea communications cables and infrastructure–consequences for the eu," *Report for SEDE Committee of the European Parliament, PE702*, vol. 557, 2022.

[8] J. Chen and Y. Han, "Shortest paths on a polyhedron," in *SOCG*, New York, NY, USA, 1990, p. 360–369.

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] J. Kim, "Submarine cables: the invisible fiber link enabling the internet," 2022. [Online]. Available: https://dgtlinfra.com/submarine-cables-fiber-link-internet/

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

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

[19] 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.

[20] 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

[21] J.-F. Libert and G. Waterworth, "13 - cable technology," in *Undersea Fiber Communication Systems (Second Edition)*. Academic Press, 2016, pp. 465–508.

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

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

[24] J. S. Mitchell and C. H. Papadimitriou, "The weighted region problem: finding shortest paths through a weighted planar subdivision," *Journal of the ACM (JACM)*, vol. 38, no. 1, pp. 18–73, 1991.

[25] 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.

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

[27] 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.

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

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

[30] V. J. Wei, R. C.-W. Wong, C. Long, and D. M. Mount, "Distance oracle on terrain surface," in *SIGMOD*, 2017, pp. 1211–1226.
[31] V. J. Wei, R. C.-W. Wong, C. Long, D. M. Mount, and H. Samet, "Proximity queries on terrain surface," *TODS*, 2022.
[32] S. Xing, C. Shahabi, and B. Pan, "Continuous monitoring of nearest neighbors on land surface," *VLDB*, vol. 2, no. 1, pp. 1114–1125, 2009.
[33] 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.
[34] Y. Yan and R. C.-W. Wong, "Efficiently finding shortest path on 3d weighted terrain surfaces for moving objects (technical report)," 2024. [Online]. Available: https://github.com/yanyinzhao/WeightedTerrainCode/blob/master/TechnicalReport.pdf

APPENDIX A
MOTIVATION

We give another motivation for our study.

Cable placement: Nowadays, over 1.35×10^5km of undersea optical fiber cable have been deployed on the seabed (represented as a weighted terrain surface) [16]. We want to minimize the weighted length of the cable for cost saving. Deeper sea levels result in higher hydraulic pressure, which reduces the cable's lifespan and increases repair and maintenance costs [7]. We assign larger weights to the terrain faces representing deeper sea levels. So, we can avoid placing the cable in these regions and reduce costs. In Figure 12, the weighted shortest path in blue line (resp. unweighted shortest path in purple dashed line) is routed in the regions with shallower (resp. deeper) sea levels. Our case study shows that the estimated cost of the cable following the weighted (resp. unweighted) shortest path is USD $45.8M (resp. $54.8M), which shows the usefulness of finding the weighted shortest path. Since we only need to calculate each path once, there is no need to pre-compute it. But, due to the large terrain datasets with more than 100M faces (e.g., for the cable between the US and China, the cable needs to pass through the whole seabed of the Pacific Ocean, which is a very large region), we need to efficiently calculate the path.

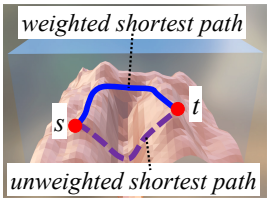

Fig. 12. Blue weighted and purple dashed unweighted shortest paths in cable placement

APPENDIX B
SUMMARY OF FREQUENT USED NOTATIONS

Table III shows a summary of frequent used notations.

TABLE III
SUMMARY OF FREQUENT USED NOTATIONS

Notation	Meaning
T	The weighted terrain surface
$V/E/F$	The set of vertices / edges / faces of T
n	The number of vertices of T
L	The length of the longest edge in T
N	The smallest integer value which is larger than or equal to the coordinate value of any vertex
W/w	The maximum / minimum weights of T
h	The minimum height of any face in F
ϵ	The error parameter
η	The constant calculated using the remaining Steiner points for controlling the error
k	The removing value
$\Pi^*(s,t)$	The optimal weighted shortest path
r_i	The intersection point of $\Pi^*(s,t)$ and an edge in E
$\Pi(s,t)$	The final calculated weighted shortest path
$\lvert\Pi(s,t)\rvert$	The weighted distance of $\Pi(s,t)$
$\overline{pq}$	A line between two points p and q on a face
$\Pi_{Roug}(s,t)$	The rough path calculated using algorithm *Roug*
$\Pi_{Ref\text{-}1}(s,t)$	The modified rough path calculated using the full edge sequence conversion step of algorithm *Ref*
$\Pi_{Ref\text{-}2}(s,t)$	The refined path calculated using the Snell's law path refinement step of algorithm *Ref*
$\Pi_{Ref\text{-}3}(s,t)$	The refined path calculated using the error guaranteed path refinement step of algorithm *Ref*
ρ_i	The intersection point of $\Pi_{Ref\text{-}2}(s,t)$ and an edge in E
ξ	The iteration counts of single endpoint cases in the full edge sequence conversion step of algorithm *Ref*
S	The edge sequence that $\Pi_{Ref\text{-}1}(s,t)$ connects from s to t in order based on T
l	The number of edges in S
Π_c	A 3D surface Snell's ray
SP_{Roug} / SP_{Ref}	The set of Steiner points on each edge of E used in algorithm *Roug* / *Ref*
G_{Roug} / G_{Ref}	The connected weighted graph used in algorithm *Roug* / *Ref*
$G_{Roug}.V$ / $G_{Ref}.V$	The set of vertices in the connected weighted graph used in algorithm *Roug* / *Ref*
$G_{Roug}.E$ / $G_{Ref}.E$	The set of edges in the connected weighted graph used in algorithm *Roug* / *Ref*

APPENDIX C
EXAMPLE ON THE GOOD PERFORMANCE OF THE EFFECTIVE WEIGHT PRUNING SUB-STEP IN THE SNELL'S LAW PATH REFMENT STEP OF ALGORITHM *Ref*

We use a 1D example to illustrate why the effective weight pruning sub-step in the Snell's law path refinement step of algorithm *Ref* can prune out unnecessary checking. Let 0 and 100 to be the position of the two endpoints of e_1, and we have $[a_1 b_1] = [0, 100]$. Assume that the position of the optimal intersection point r_1 on the first edge is 87.32. Then, without the effective weight pruning sub-step in the Snell's law path refinement step of algorithm *Ref*, the searching interval will be $[50, 100]$, $[75, 100]$, $[75, 87.5]$, $[81.25, 87.5]$, $[84.375, 87.5]$, $[85.9375, 87.5]$, $[86.71875, 87.5]$, $[87.109375, 87.5] \cdots$. But, with the effective weight pruning sub-step in the Snell's law path refinement step of algorithm *Ref*, assume that we still need to use several iterations of the original binary search to let Π_m pass the whole S based on T, and we need to check $[50, 100]$, $[75, 100]$, $[75, 87.5]$. After checking the interval $[75, 87.5]$, we get a Π_m that passes the whole S based on T. Assume we calculate m_{ef} as 87 using effective weight pruning step. As a result, in the next checking, we can directly limit the searching interval to be $[87, 87.5]$, which can prune out some unnecessary interval checking including $[81.25, 87.5]$, $[84.375, 87.5]$, $[85.9375, 87.5]$, $[86.71875, 87.5]$, and thus, the effective weight pruning sub-step in the Snell's law path refinement step of algorithm *Ref* can save the running

time and memory usage.

APPENDIX D
EMPIRICAL STUDIES

A. Experimental Results

(1) Figure 13 and Figure 14, (2) Figure 15 and Figure 16 show the result on the *BH-small* dataset when varying k and ϵ, respectively. (3) Figure 17 and Figure 18, (4) Figure 19 and Figure 20 show the result on the *BH* dataset when varying k and ϵ, respectively. (5) Figure 21 and Figure 22, (6) Figure 9 and Figure 23 show the result on the *EP-small* dataset when varying k and ϵ, respectively. (7) Figure 24 and Figure 25, (8) Figure 26 and Figure 27 show the result on the *EP-small* dataset when varying k and ϵ, respectively. (9) Figure 28 and Figure 29 show the result on multi-resolution of *BH-small* datasets when varying dataset size. (10) Figure 30 and Figure 31 show the result on multi-resolution of *BH* datasets when varying dataset size. (11) Figure 32 and Figure 33 show the result on multi-resolution of *EP-small* datasets when varying dataset size. (12) Figure 34 and Figure 35 show the result on multi-resolution of *EP* datasets when varying dataset size.

*1) **Ablation study of Roug-Ref**:* In our algorithm *Roug-Ref*, recall that we have six variations of algorithms, i.e., (1) *Roug-Ref-NoEffSP*, (2) *Roug-Ref-NoEdgSeqConv*, (3) *Roug-Ref-NoEffWeig*, (4) *Roug-Ref-NoPrunDijk*, (5) *Roug-Ref-Naive1*, and (6) *Roug-Ref-Naive2*, for ablation study. Since k will only affect the variations of *Roug-Ref* and our algorithm *Roug-Ref*, we study the effect of k in this subsection.

Effect of k. In Figure 13, Figure 17, Figure 21 and Figure 24, we tested 5 values of k from $\{1, 2, 3, 4, 5\}$ on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting ϵ to be 0.1 on *BH-small* and *EP-small* datasets, and 0.25 on *BH* and *EP* datasets. Figure 14, Figure 18, Figure 22 and Figure 25 are the separated query time, memory usage, and the average number of Steiner points per edge in two steps for these results.

(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 [34]). (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**:* From the previous subsection, *Roug-Ref* is the best algorithm among different variations. Starting from this subsection, we study the effect of ϵ and the dataset size by comparing different baselines with *Roug-Ref* when $k = 2$. But, they will not affect the chance of using error guaranteed path refinement step in *Roug-Ref*, and the chance is always smaller than 1% when $k = 2$, so we omit it in the following comparisons.

Effect of ϵ. In Figure 15, Figure 19, Figure 9 and Figure 26, we tested 6 values of ϵ from $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting k to be 2. Figure 16, Figure 20, Figure 23 and Figure 27 are the separated query time and memory usage in two steps for these results. Since when $k = 2$, there is no need to place Steiner points in *Ref* (i.e., all Steiner points are used in *Roug*), so there is no need to show the separated average number of Steiner points per edge in two steps. (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 clearer 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* performs worse than *FixSP* since *EdgSeq* 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 28 and Figure 32, we tested 5 values of dataset size from $\{$10k, 20k, 30k, 40k, 50k$\}$ on multi-resolution of *BH-small* AND *EP-small* datasets by setting ϵ to be 0.1 and k to be 2. Figure 29 and Figure 33 are the separated query time and memory usage in two steps for these results. Since when $k = 2$, there is no need to place Steiner points in *Ref* (i.e., all Steiner points are used in *Roug*), so there is no need to show the separated average number of Steiner points per edge in two steps. *Roug-Ref* performs better than *FixSP*, *FixSP-NoWei-Adp* and *LogSP*. (1) When the dataset size increases, the query time and the average number of Steiner points per edge of all algorithms increase. (2) *Roug-Ref* has the smallest query time. 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*, *FixSP*, *FixSP-NoWei-Adp* and *LogSP-Adp*, respectively.

*3) **Scalability test**:* In Figure 30 and Figure 34, we tested 5 values of dataset size from $\{$1M, 2M, 3M, 4M, 5M$\}$ on multi-resolution of *BH* and *EP* datasets by setting ϵ to be 0.25 and k to be 2. Figure 31 and Figure 35 are the separated query time and memory usage in two steps for these results. Since when $k = 2$, there is no need to place Steiner points in *Ref* (i.e., all Steiner points are used in *Roug*), so there is no need to show the separated average number of Steiner points per edge in two steps. (i) *Roug-Ref* can still beat *LogSP-Adp* in terms of the query time and memory usage. (ii) When the dataset size is 5M, *Roug-Ref*'s query time is still reasonable. However, the query times for *EdgSeq*, *FixSP* and *FixSP-NoWei-Adp* are

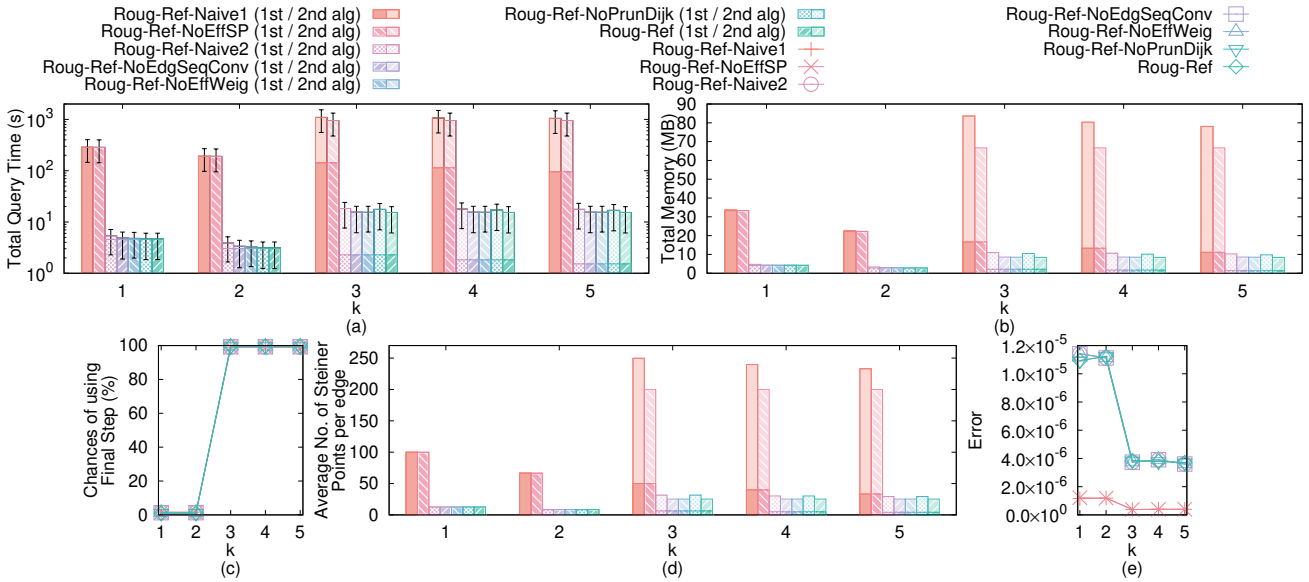

Fig. 13. Ablation study (effect of k on *BH-small* dataset)

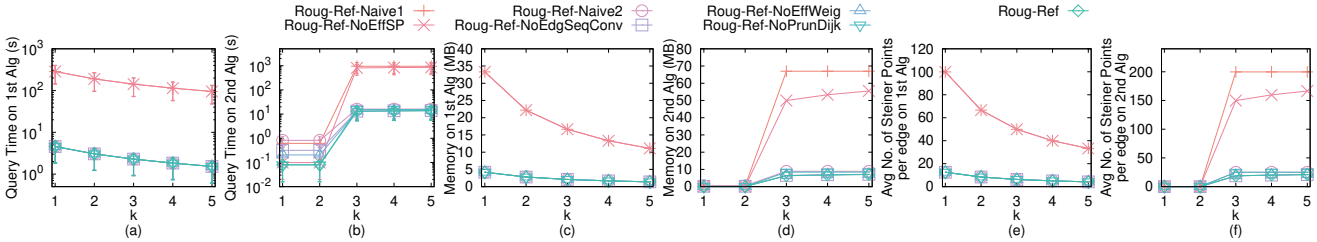

Fig. 14. Ablation study (effect of k on *BH-small* dataset with separated query time, memory usage, and the average number of Steiner points per edge in two steps)

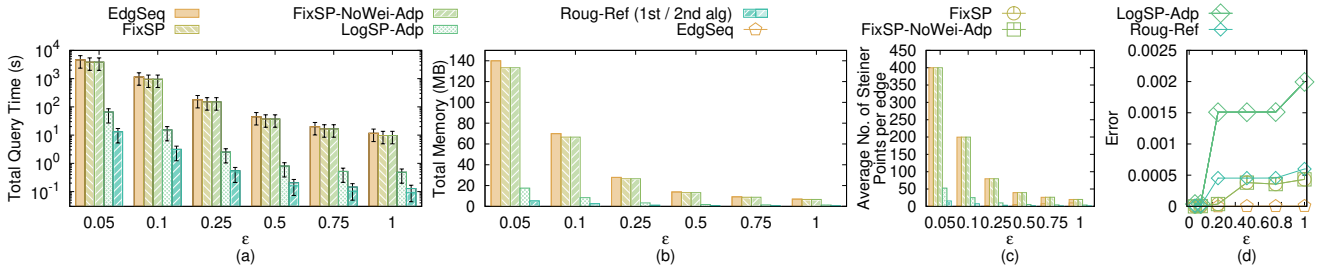

Fig. 15. Baseline comparisons (effect of ϵ on *BH-small* dataset)

larger than 7 days, so they are excluded from the figure.

4) ***Large distance error algorithm comparisons****:* In Figure 36 and Figure 37, we compare *LogSP* and *Roug* with *Roug-Ref* using the method mentioned in Section V-B4. *Roug-Ref*, *LogSP* and *Roug* run in 4s, 160 and 120s when $\epsilon = 0.1$ on *BH-small* dataset due to the rough-refine concept in *Roug-Ref*. It seems that *LogSP* and *Roug* have larger errors theoretically, so they can run faster. But, when we need to calculate the path with the same distance, their performances are worse than *Roug-Ref*.

5) ***Case study (earthquake and avalanche)****:* Figure 38 and Figure 39 show the result for the earthquake and avalanche recuse case study when varying ϵ. Our motivation and user study has already shown that the weighted shortest path is better than the unweighted shortest path due to the small recuse time. When $\epsilon = 0.05$, the average query time for the state-of-the-art algorithm *FixSP* and our algorithm *Roug-Ref* are 11,900s $\approx$ 3.3 hours and 7.3s, respectively. So, the weighted shortest path calculated by *Roug-Ref* is the most suitable algorithm for earthquake and avalanche recuse.

6) ***Case study (Path Advisor)****:* We conducted a case study on *Path Advisor* [33] as mentioned in Section I-A by using *PA* dataset [33]. We chose two places in Path Advisor as source and destination, and repeated it for 100 times to calculate the path (with $\epsilon = 0.5$). In Figure 1 (c), the weighted (resp. unweighted) shortest path has distance 105.8m (resp. 98.4m).

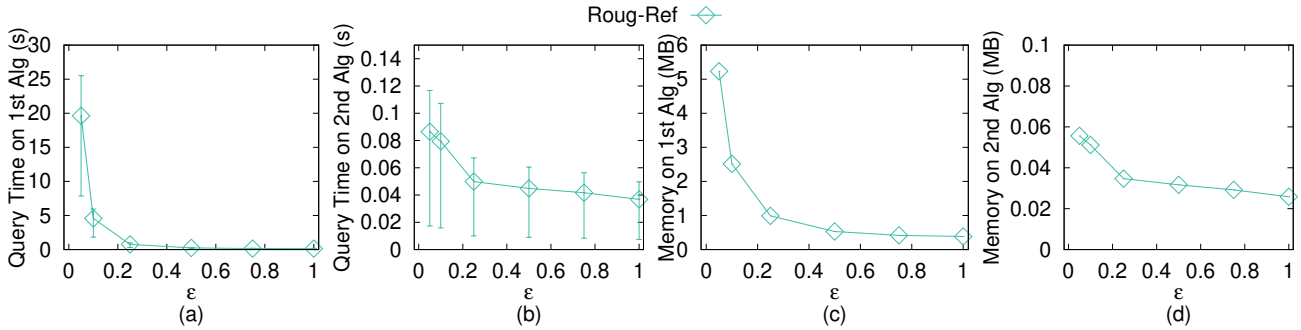

Fig. 16. Baseline comparisons (effect of ϵ on *BH-small* dataset with separated query time and memory usage in two steps)

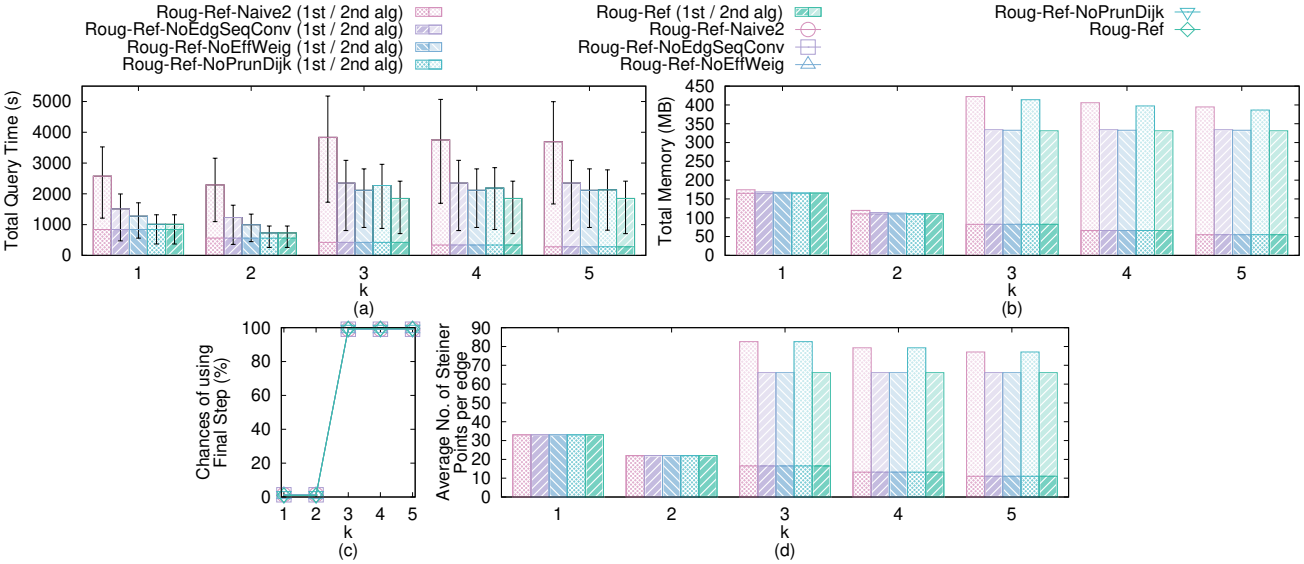

Fig. 17. Ablation study (effect of k on *BH* dataset)

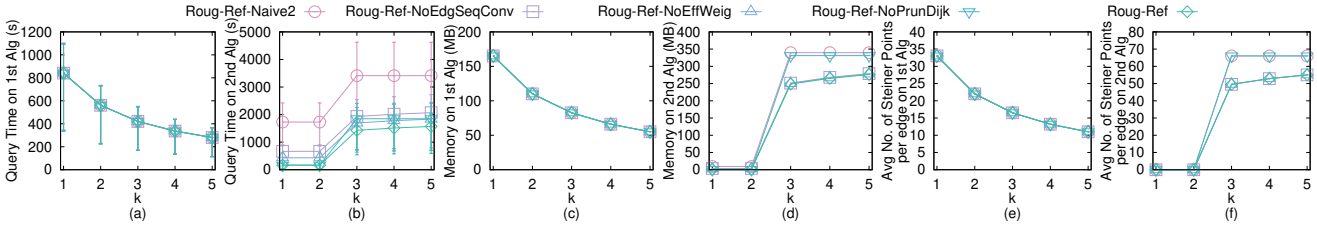

Fig. 18. Ablation study (effect of k on *BH* dataset with separated query time, memory usage, and the average number of Steiner points per edge in two steps)

We presented the figure and the path distance result to 30 users (i.e., university students), and 96.7% of users think the blue path is the more realistic since it maintains a safe distance from obstacles and avoids abrupt changes in direction. The average query time for the best-known algorithm *FixSP* and *Roug-Ref* are 16.62s and 0.1s, respectively.

Figure 40 and Figure 41 show the result for Path Advisor user study when varying ϵ. When $\epsilon = 0.5$, the average query time for the state-of-the-art algorithm *FixSP* and our algorithm *Roug-Ref* are 16.62s and 0.1s, respectively. In addition, in a map application, the query time is the most crucial factor since users would like to get the result in a shorter time. Thus, *Roug-Ref* is the most suitable algorithm for Path Advisor.

*7) **Case study (cable placement)**:* We conducted a case study on the cable placement as mentioned in Section A by using *SB* dataset [5]. The average life expectancy of a cable is 25 years [21], and if the cable is in deep waters (e.g., 8.5km or greater), the cable needs to be repaired frequently (e.g., its life expectancy is reduced to 20 years) [21]. We randomly chose pairs of points as source and destination, respectively, and repeated it 100 times to calculate the path (with $\epsilon = 0.5$). In Figure 12, the weighted (resp. unweighted) shortest path has distance 457.9km (resp. 438.3km). Given the cost of undersea cable, i.e., USD $25,000/km [16], when constructing a cable that will be used for 100 years [21], the total estimated cost for the blue and dashed purple paths are USD $45.8M (=

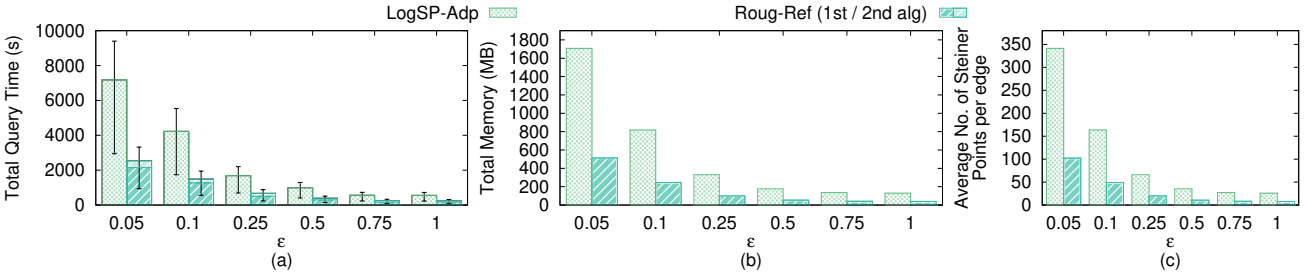

Fig. 19. Baseline comparisons (effect of ϵ on *BH* dataset)

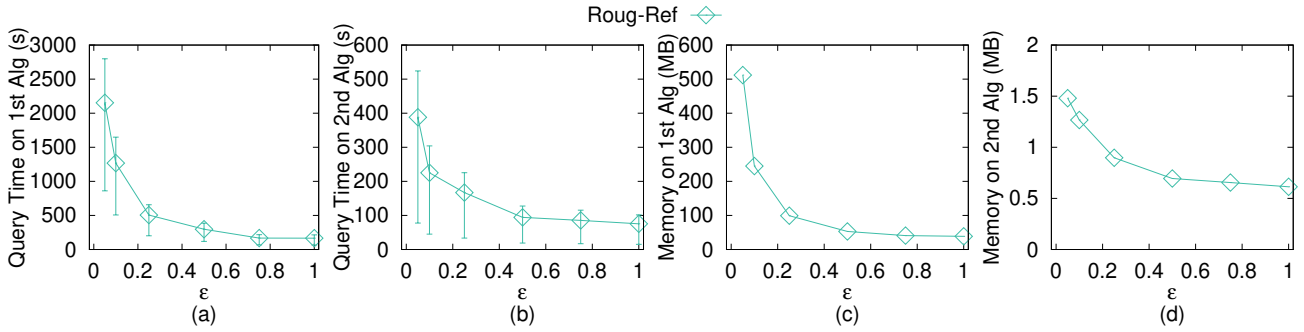

Fig. 20. Baseline comparisons (effect of ϵ on *BH* dataset with separated query time and memory usage in two steps)

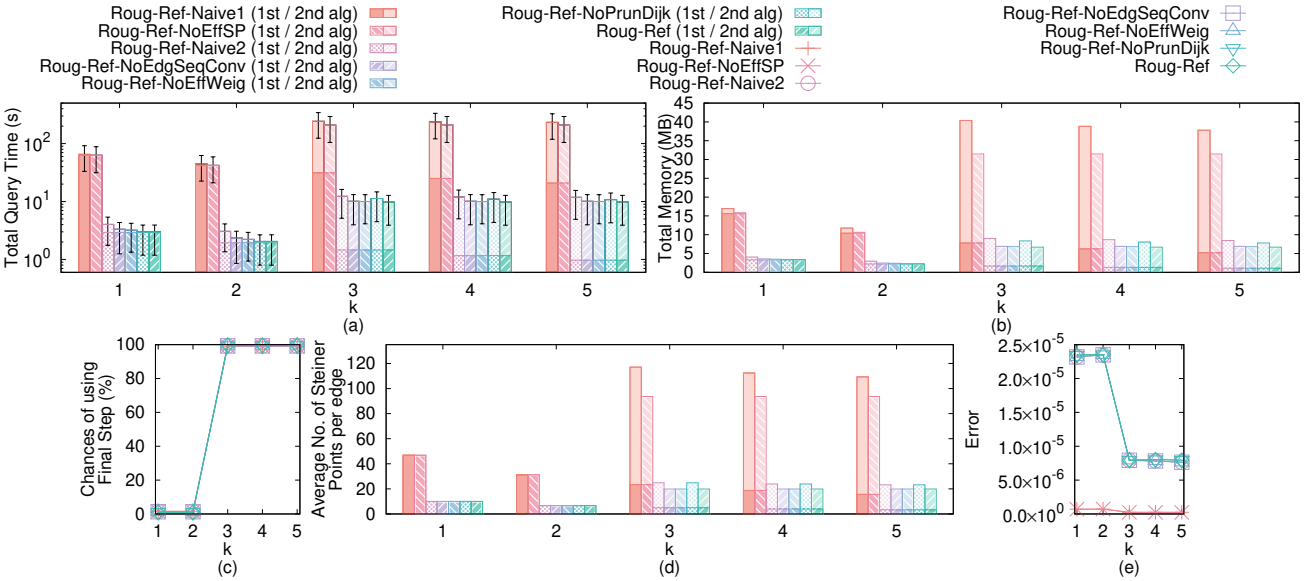

Fig. 21. Ablation study (effect of k on *EP-small* dataset)

$\frac{100\text{years}}{25\text{years}} \times 457.9\text{km} \times \$25,000/\text{km}$) and \$54.8M ($= \frac{100\text{years}}{20\text{years}} \times 438.3\text{km} \times \$25,000/\text{km}$), respectively. The average query time for the best-known algorithm *FixSP* and *Roug-Ref* are 22.50s and 0.2s, respectively.

Figure 42 and Figure 43 show the result for seabed motivation study when varying ϵ. *Roug-Ref* still has the smallest query time and memory usage.

B. Generating Datasets with Different Dataset Sizes

The procedure for generating the datasets with different dataset sizes is as follows. We mainly follow the procedure for generating datasets with different dataset sizes in the study [22], [30], [31]. Let $T_t = (V_t, E_t, F_t)$ be our target terrain that we want to generate with ex_t edges along x-coordinate, ey_t edges along y-coordinate and dataset size of DS_t, where $DS_t = 2 \cdot ex_t \cdot ey_t$. Let $T_o = (V_o, E_o, F_o)$ be the original terrain that we currently have with ex_o edges along x-coordinate, ey_o edges along y-coordinate and dataset size of DS_o, where $DS_o = 2 \cdot ex_o \cdot ey_o$. We then generate $(ex_t + 1) \cdot (ey_t + 1)$ 2D points (x, y) based on a Normal distribution $N(\mu_N, \sigma_N^2)$, where $\mu_N = (\overline{x} = \frac{\sum_{v_o \in V_o} x_{v_o}}{(ex_o+1)\cdot(ey_o+1)}, \overline{y} = \frac{\sum_{v_o \in V_o} y_{v_o}}{(ex_o+1)\cdot(ey_o+1)})$ and $\sigma_N^2 =$

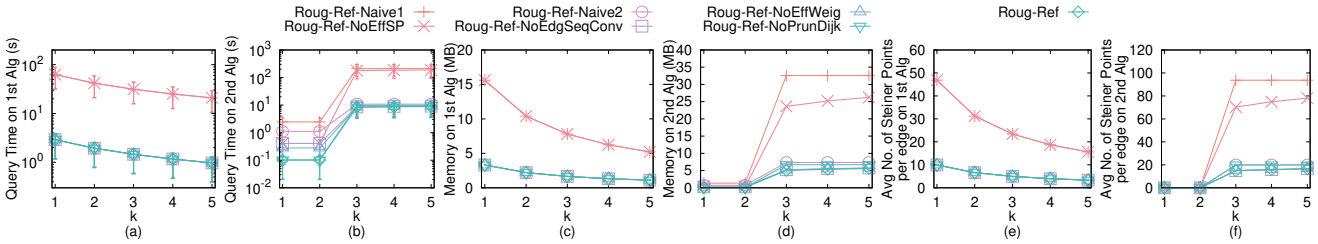

Fig. 22. Ablation study (effect of k on *EP-small* dataset with separated query time, memory usage, and the average number of Steiner points per edge in two steps)

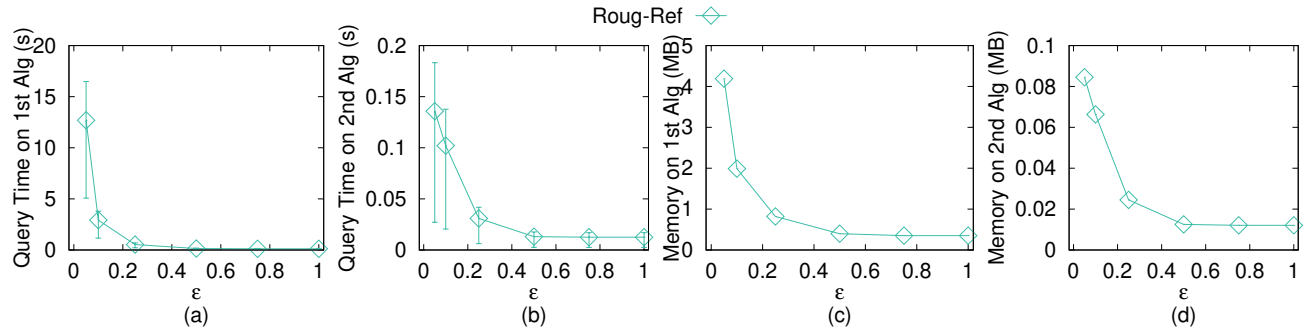

Fig. 23. Baseline comparisons (effect of ϵ on *EP-small* dataset with separated query time and memory usage in two steps)

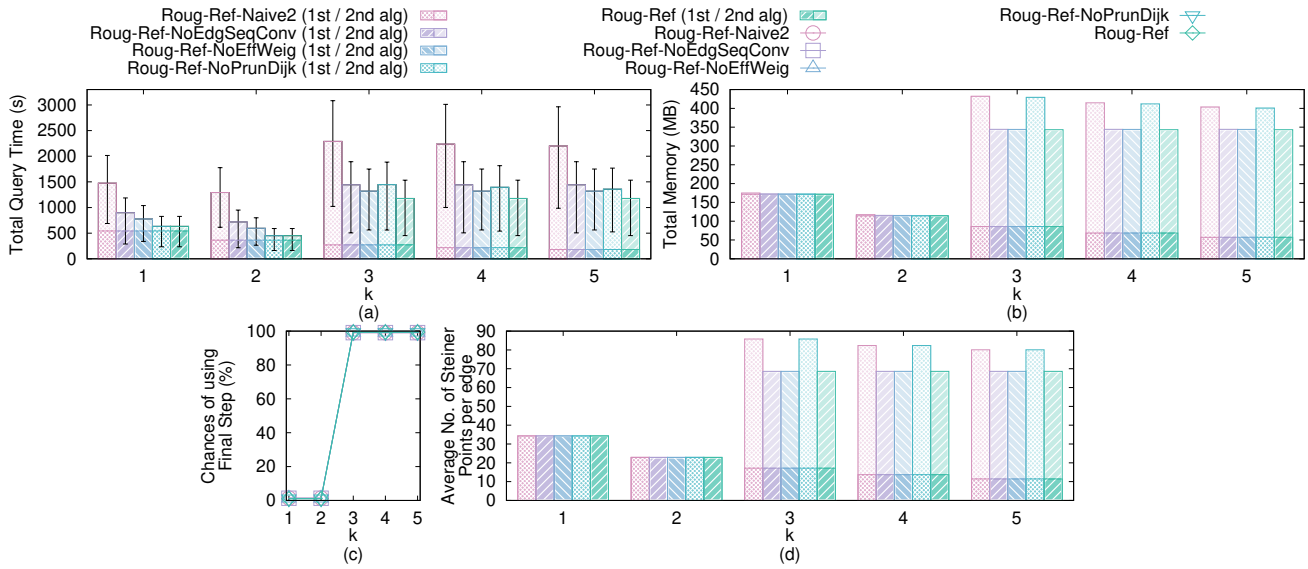

Fig. 24. Ablation study (effect of k on *EP* dataset)

$(\frac{\sum_{v_o \in V_o}(x_{v_o}-\overline{x})^2}{(ex_o+1)\cdot(ey_o+1)}, \frac{\sum_{v_o \in V_o}(y_{v_o}-\overline{y})^2}{(ex_o+1)\cdot(ey_o+1)})$. In the end, we project each generated point (x, y) to the surface of T_o and take the projected point (also add edges between neighbours of two points to form edges and faces) as the newly generate T_t.

Appendix E
Comparison of All Algorithms

Table IV shows a comparison of all algorithms.

Appendix F
Proofs

Lemma 1. *There are at most $k_{SP} \leq 2(1+\log_\lambda \frac{L}{2\cdot r})$ Steiner points on each edge in E when placing Steiner point based on ϵ in algorithm Roug, where r is the minimum r_v for all $v \in V$.*

Proof. We prove it for the extreme case, i.e., k_{SP} is maximized. This case happens when the edge has maximum length L and it joins two vertices has minimum radius r. Since each edge contains two endpoints, we have two sets of Steiner points from both endpoints, and we have the factor 2. When placing Steiner point based on ϵ in algorithm *Roug*, each set of Steiner points contains at most $(1+\log_\lambda \frac{L}{2\cdot r})$ Steiner points, where the 1 comes from the first Steiner point that is the nearest one from the endpoint. Therefore, we have $k_{SP} \leq 2(1+\log_\lambda \frac{L}{2\cdot r})$. □

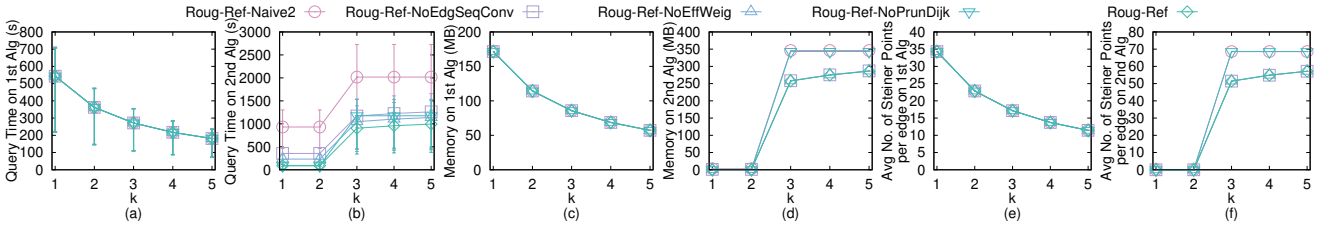

Fig. 25. Ablation study (effect of k on *EP* dataset with separated query time, memory usage, and the average number of Steiner points per edge in two steps)

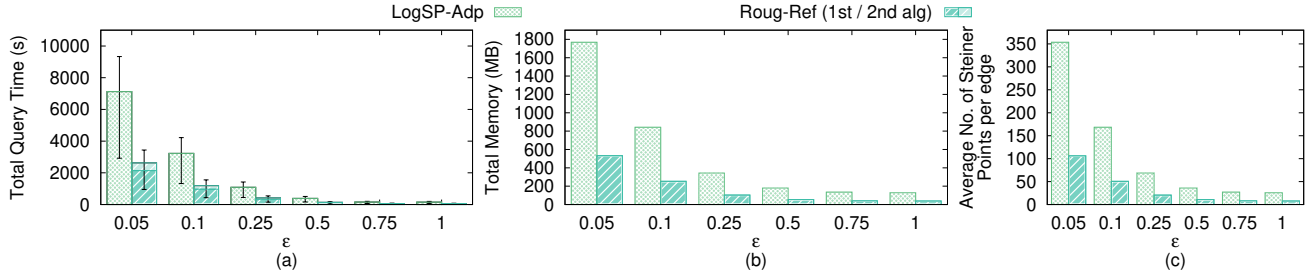

Fig. 26. Baseline comparisons (effect of ϵ on *EP* dataset)

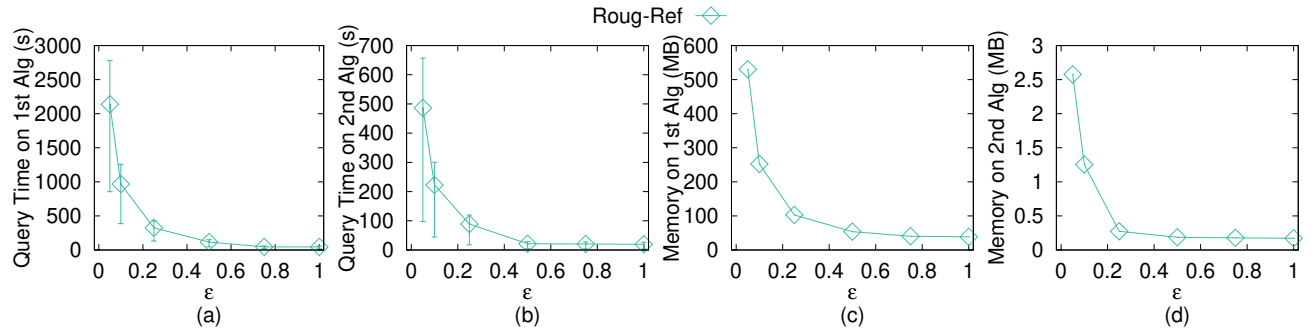

Fig. 27. Baseline comparisons (effect of ϵ on *EP* dataset with separated query time and memory usage in two steps)

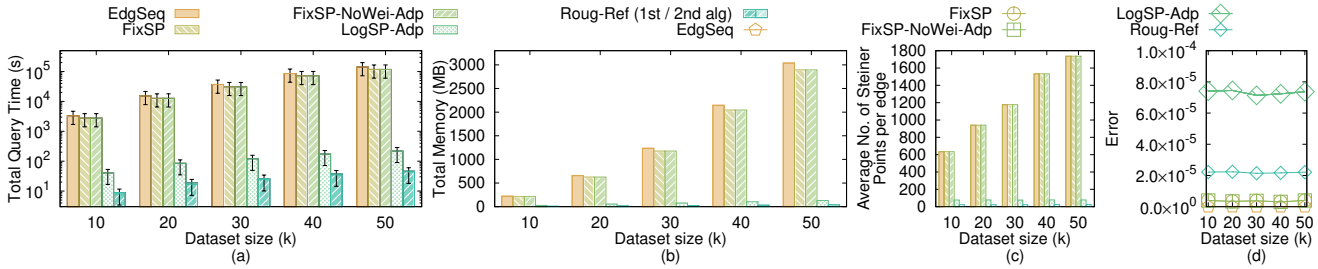

Fig. 28. Baseline comparisons (effect of dataset size on multi-resolution of *BH-small* datasets)

Lemma 2. *When placing Steiner point based on ϵ in algorithm Roug, based on $1 + (2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = 1 + \epsilon$, we obtain $\epsilon' = \frac{1+\epsilon+\frac{W}{w} - \sqrt{(1+\epsilon+\frac{W}{w})^2 - 4\epsilon}}{4}$ with $0 < \epsilon' < \frac{1}{2}$ and $\epsilon > 0$.*

Proof. The mathematical derivation is like we regard ϵ' as an unknown and solve a quadratic equation. The derivation is as follows.

$$1 + (2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = 1 + \epsilon$$

$$(2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = \epsilon$$

$$2 + \frac{2W}{(1-2\epsilon')\cdot w} = \frac{\epsilon}{\epsilon'}$$

$$\frac{2W}{(1-2\epsilon')\cdot w} = \frac{\epsilon - 2\epsilon'}{\epsilon'}$$

$$2\frac{W}{w}\epsilon' = \epsilon - (2+2\epsilon)\epsilon' + 4\epsilon'^2$$

$$4\epsilon'^2 - (2 + 2\epsilon + 2\frac{W}{w})\epsilon' + \epsilon = 0$$

$$\epsilon' = \frac{2 + 2\epsilon + 2\frac{W}{w} \pm \sqrt{4(1+\epsilon+\frac{W}{w})^2 - 16\epsilon}}{8}$$

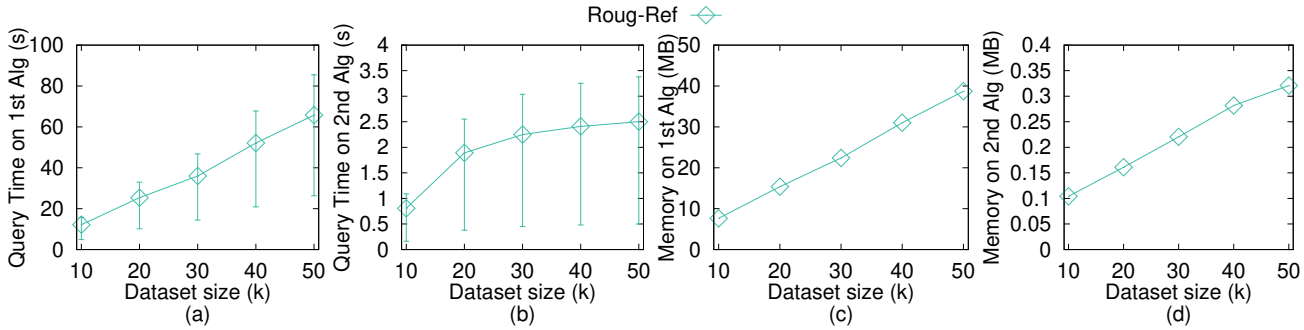

Fig. 29. Baseline comparisons (effect of dataset size on multi-resolution of *BH-small* datasets with separated query time and memory usage in two steps)

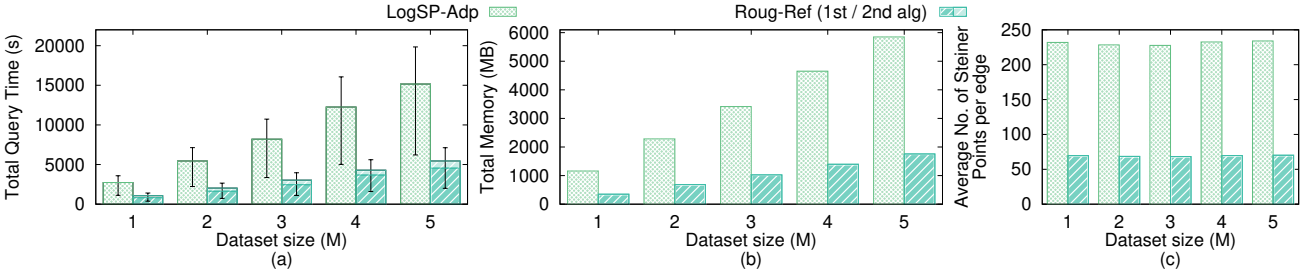

Fig. 30. Scalability test (effect of dataset size on multi-resolution of *BH* datasets)

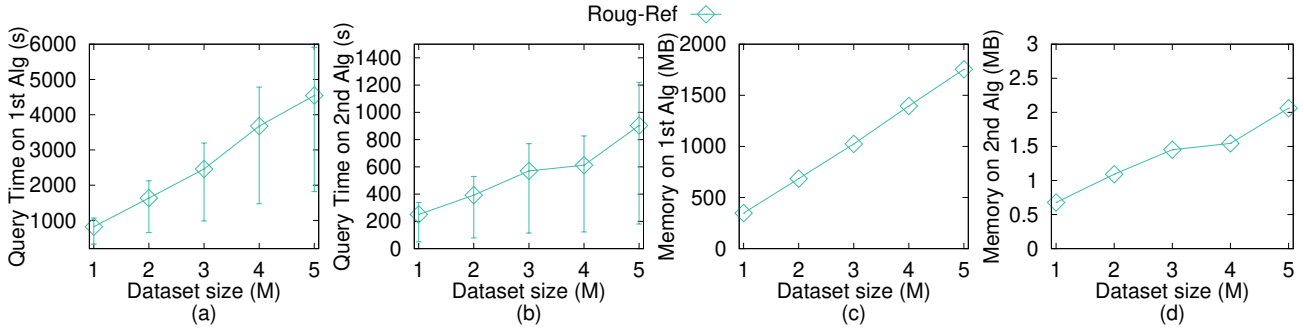

Fig. 31. Scalability test (effect of dataset size on multi-resolution of *BH* datasets with separated query time and memory usage in two steps)

$$\epsilon' = \frac{1+\epsilon+\frac{W}{w} \pm \sqrt{(1+\epsilon+\frac{W}{w})^2 - 4\epsilon}}{4}$$

Finally, we take $\epsilon' = \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4}$ since $0 < \epsilon' < \frac{1}{2}$ (we can plot the figure for this expression, and will found that the upper limit is always $\frac{1}{2}$ if we use $-$). □

Lemma 3. *Let h be the minimum height of any face in F whose vertices have non-negative integer coordinates no greater than N. Then, $h \geq \frac{1}{N\sqrt{3}}$.*

Proof. Let a and b be two non-zero vectors with non-negative integer coordinates no greater than N, and a and b are not co-linear. Since we know $\frac{|a\times b|}{2}$ is the face area of a and b, we have $h = \min_{a,b} \frac{|a\times b|}{|b|} = \min_{a,b} \frac{\sqrt{\omega}}{\sqrt{x_a^2+y_a^2+z_a^2}} \geq \frac{1}{N\sqrt{3}} \min_{a,b} \sqrt{\omega} \geq \frac{1}{N\sqrt{3}}$, where $\omega = (y_a z_b - z_a y_b)^2 + (z_a x_b - x_a z_b)^2 + (x_a y_b - y_a x_b)^2$. □

Theorem 2. *The running time for algorithm Roug is $O(n \log n)$ and the memory usage is $O(n)$.*

Proof. Originally, if we do not remove Steiner points in algorithm *Roug*, i.e., we place Steiner point based on ϵ, then following Lemma 1, the number of Steiner points k_{SP} on each edge is $O(\log_\lambda \frac{L}{r})$, where $\lambda = (1 + \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4} \cdot \sin\theta)$, $r = \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4} \cdot h$ and θ is the minimum θ_v for all $v \in V$. Following Lemma 2 and Lemma 3, $r = O(\frac{\epsilon}{N})$, and thus $k_{SP} = O(\log \frac{LN}{\epsilon})$. But, since we have removed Steiner points for η calculation and rough path calculation, the remaining Steiner points on each edge is $O(1)$. So $|G_{Roug}.V| = n$. Since we know for a graph with n vertices, the running time and memory usage of Dijkstra's algorithm on this graph are $O(n \log n)$ and n, so the running time of algorithm *Roug* is $O(n \log n)$ and the memory usage is $O(n)$. □

Theorem 3. *The running time for the full edge sequence conversion step in algorithm Ref is $O(n \log n)$ and the memory usage is $O(n)$.*

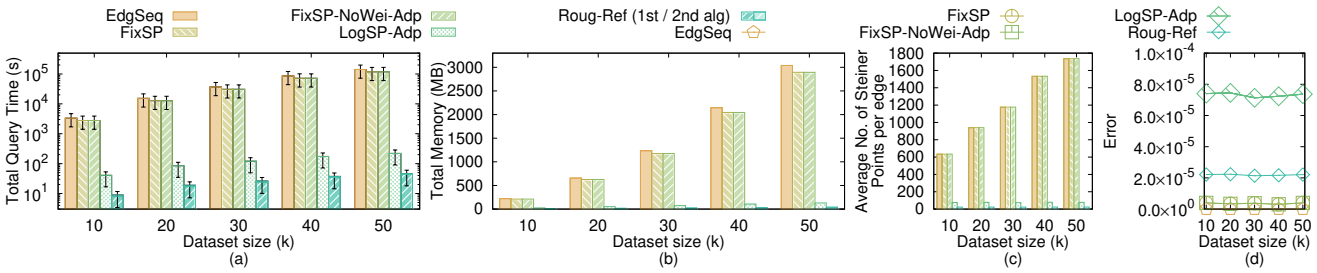

Fig. 32. Baseline comparisons (effect of dataset size on multi-resolution of *EP-small* datasets)

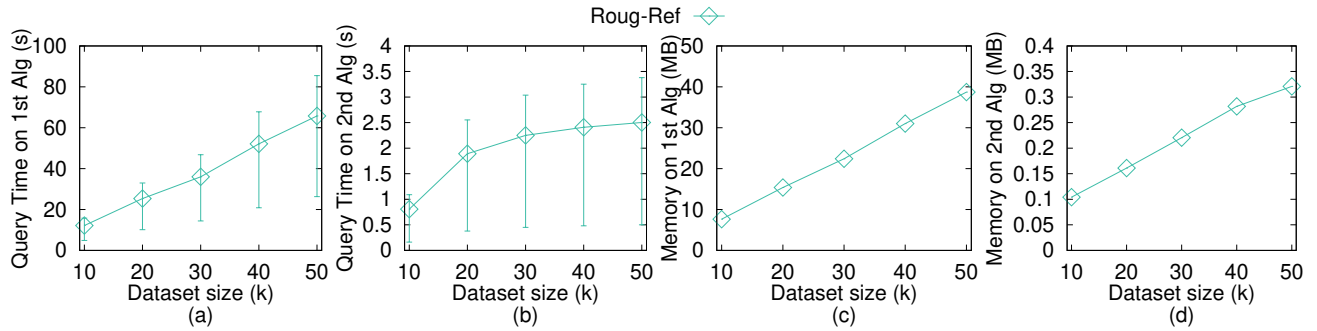

Fig. 33. Baseline comparisons (effect of dataset size on multi-resolution of *EP-small* datasets with separated query time and memory usage in two steps)

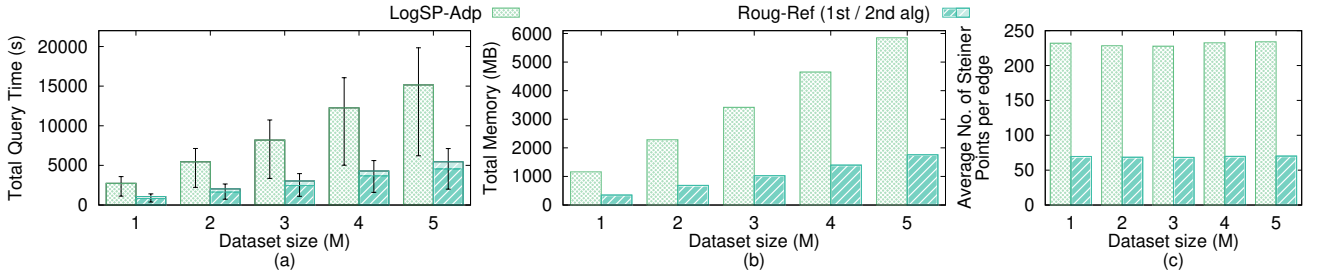

Fig. 34. Scalability test (effect of dataset size on multi-resolution of *EP* datasets)

Proof. Firstly, we prove the *running time*. Given $\Pi_{Roug}(s,t)$, there are three cases on how to apply the full edge sequence conversion step in algorithm *Ref* on $\Pi_{Roug}(s,t)$, i.e., (1) some segments of $\Pi_{Roug}(s,t)$ passes on the edges (i.e., no need to use algorithm *Ref* full edge sequence conversion step), (2) some segments of $\Pi_{Roug}(s,t)$ belongs to single endpoint case, and (3) some segments of $\Pi_{Roug}(s,t)$ belongs to successive endpoint case. For the first case, there is no need to case about it. For the second case, we just need to add more Steiner points on the edges adjacent to the vertices passed by $\Pi_{Roug}(s,t)$, and using Dijkstra's algorithm to refine it, and the running time is the same as the one in algorithm *Roug*, which is $O(n \log n)$. For the third case, we just need to add more Steiner points on the edge adjacent to the vertex passed by $\Pi_{Roug}(s,t)$, and find a shorter path by running for ζ times, and there are at most $O(n)$ such vertices, so the running time is $O(\zeta n) = O(n)$. Therefore, the running time for the full edge sequence conversion step in algorithm *Ref* is $O(n \log n)$.

Secondly, we prove the *memory usage*. Algorithm *Roug* needs $O(n)$ memory since it is a common Dijkstra's algorithm, whose memory usage is $O(|G_{Roug}.V|)$, where $|G_{Roug}.V|$ is size of vertices in the Dijkstra's algorithm. Handling one single endpoint case needs $O(1)$ memory. Since there can be at most n single endpoint cases, the memory usage is $O(n)$. Handling successive endpoint cases needs $O(n)$ memory since algorithm *Roug* needs $O(n)$ memory. Therefore, the memory usage for the full edge sequence conversion step in algorithm *Ref* is $O(n)$. □

Theorem 4. *The running time for the Snell's law path refinement step in algorithm Ref is $O(l)$, and the memory usage is $O(l)$.*

Proof. Firstly, we prove the *running time*. Let l be the number of edges in S. Since the effective weight pruning sub-step can directly find the optimal position of the intersection point on each edge in S in $O(1)$ time, the running time of the Snell's law path refinement step in algorithm *Ref* is $O(l)$.

Secondly, we prove the *memory usage*, since the refined path will pass l edges, so the memory usage of the Snell's law path refinement step in algorithm *Ref* is $O(l)$. □

Proof of Theorem 1. Firstly, we prove the *total running time*. (1) In most of the cases, there is no need to use the error

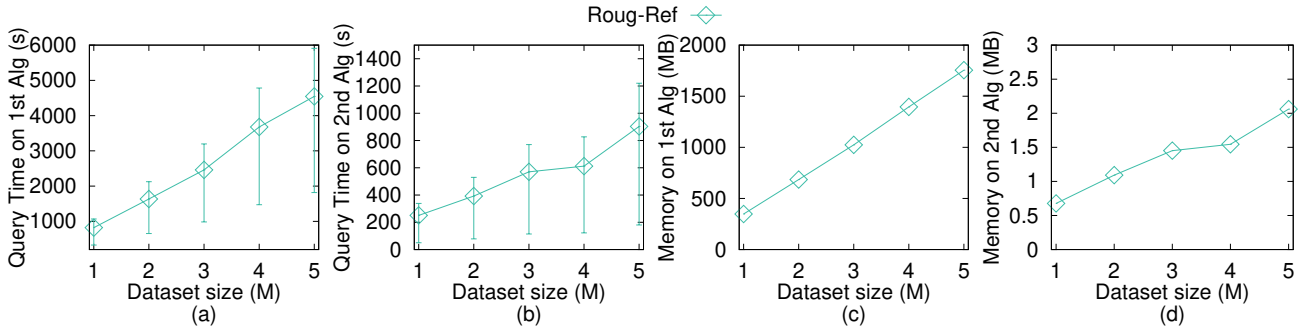

Fig. 35. Scalability test (effect of dataset size on multi-resolution of *EP* datasets with separated query time and memory usage in two steps)

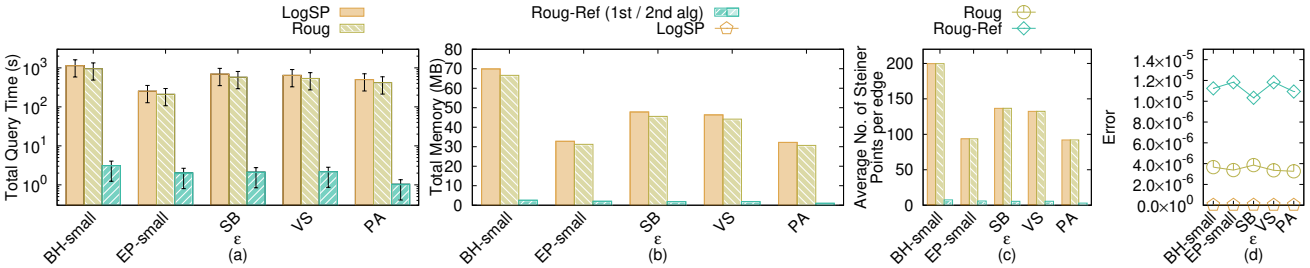

Fig. 36. Large distance error algorithm comparisons

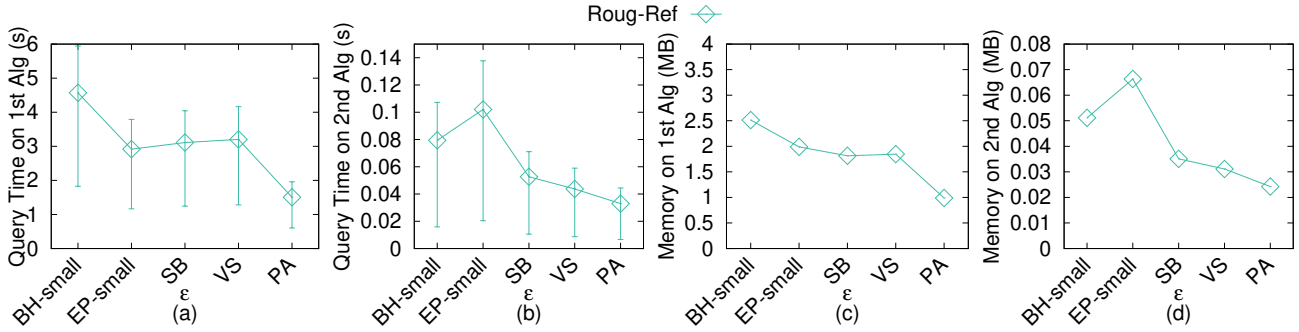

Fig. 37. Large distance error algorithm comparisons with separated query time and memory usage in two steps

guaranteed path refinement step in algorithm *Ref*. In this case, the total running time is the sum of the running time using algorithm *Roug* and the first three steps in algorithm *Ref*. From Theorem 2, Theorem 3 and Theorem 4, we obtain the total running time $O(n \log n + l)$. (2) In some special cases, we need to use the error guaranteed path refinement step in algorithm *Ref* for error guarantee. The sum of the running time of algorithm *Roug* and the error guaranteed path refinement step in algorithm *Ref* is exactly the same as the running time that we perform Dijkstra's algorithm on the weighted graph G_{Ref} constructed by the original Steiner points (i.e., $k_{SP} = O(\log \frac{LN}{\epsilon})$ Steiner points per edge) and V, which is $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$. But the constant term $O(\log \frac{LN}{\epsilon})$ is not important and can be omitted, so we obtain the total running time $O(n \log n + l)$. (3) In general, the total running time is $O(n \log n + l)$.

Secondly, we prove the *total memory usage*. (1) In most of the cases, there is no need to use the error guaranteed path refinement step in algorithm *Ref*. In this case, the total memory usage is the sum of the memory usage using algorithm *Roug* and the first three steps in algorithm *Ref*. From Theorem 2, Theorem 3 and Theorem 4, we obtain the average case total memory usage $O(n + l)$. (2) In some cases, we need to use the error guaranteed path refinement step in algorithm *Ref* for error guarantee. The sum of the memory usage of algorithm *Roug* and the error guaranteed path refinement step in algorithm *Ref* is exactly the same as the memory usage that we perform Dijkstra's algorithm on the weighted graph G_{Ref} constructed by the original Steiner points (i.e., $k_{SP} = O(\log \frac{LN}{\epsilon})$ Steiner points per edge) and V, which is $O(n \log \frac{LN}{\epsilon} + l)$. But the constant term $O(\log \frac{LN}{\epsilon})$ is not important and can be omitted, so we obtain the total memory usage $O(n + l)$. (3) In general, the total memory usage is $O(n + l)$.

Finally, we prove the *error bound*. Recall one baseline algorithm *LogSP*. We define the path calculated by algorithm *LogSP* between s and t to be $\Pi_{LogSP}(s, t)$. The study [4], [17] show that $|\Pi_{LogSP}(s, t)| \leq (1 + (2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon')|\Pi^*(s, t)|$.

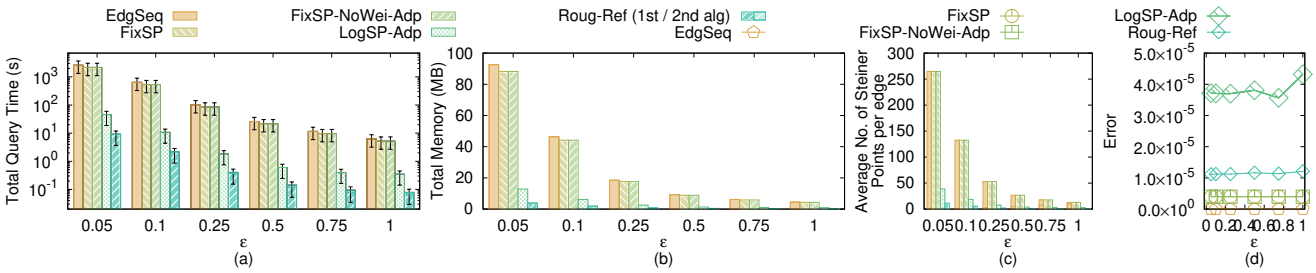

Fig. 38. Earthquake and avalanche user study (effect of ϵ on *VS* dataset)

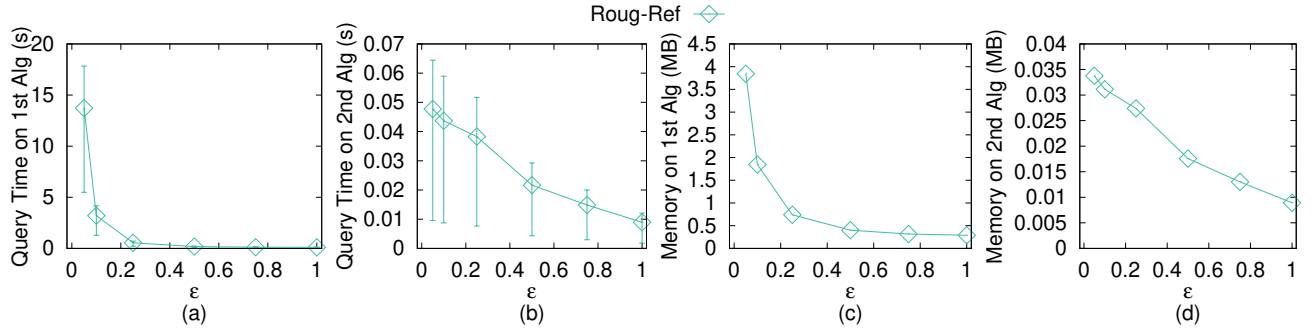

Fig. 39. Earthquake and avalanche user study (effect of ϵ on *VS* dataset with separated query time and memory usage in two steps)

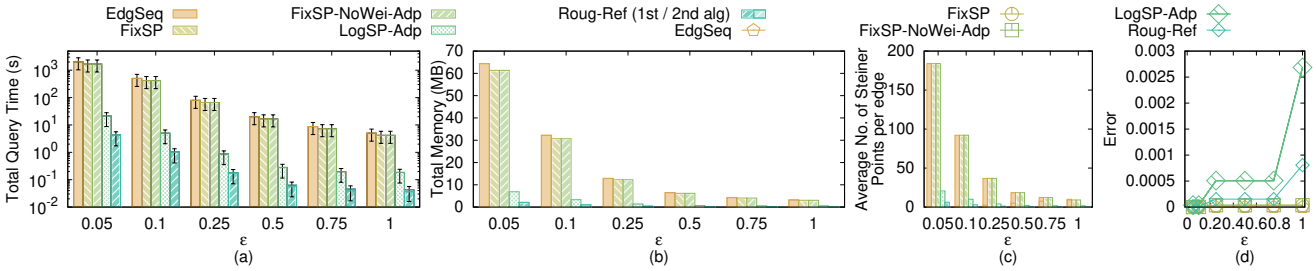

Fig. 40. Path Advisor user study (effect of ϵ on *PA* dataset)

A proof sketch appears in Theorem 1 of study [4] and a detailed proof appears in Theorem 3.1 of study [17]. We adapt algorithm *LogSP* to be algorithm *LogSP-Adp* by substituting $(2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = \epsilon$ with $0 < \epsilon' < \frac{1}{2}$ and $\epsilon > 0$, we have $|\Pi_{\textit{LogSP-Adp}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$ where $\epsilon > 0$ and $\Pi_{\textit{LogSP-Adp}}(s,t)$ is the path result returned by algorithm *LogSP-Adp*. Recall that algorithm *LogSP-Adp* corresponds to the efficient Steiner point placement scheme in algorithm *Roug*. This error bound is always true no matter whether the edge sequence passed by $\Pi_{\textit{LogSP-Adp}}(s,t)$ is the same as the edge sequence passed by $\Pi^*(s,t)$ or not. Then, in algorithm *Roug*, we first remove some Steiner points in the rough path calculation step, and then calculate $\eta\epsilon$ based on the remaining Steiner points, and then use $\eta\epsilon$ as the input error to calculate $\Pi_{\textit{Roug}}(s,t)$ in the rough path calculation step, so by adapt $\eta\epsilon$ as the input error in algorithm *LogSP-Adp*, we have $|\Pi_{\textit{Roug}}(s,t)| \leq (1+\eta\epsilon)|\Pi^*(s,t)|$. Next, in the path checking step of algorithm *Ref*, if $|\Pi_{\textit{Ref-2}}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{\textit{Roug}}(s,t)|$, we return $\Pi_{\textit{Ref-2}}(s,t)$ as output $\Pi(s,t)$, which implies that $|\Pi_{\textit{Ref-2}}(s,t)| = |\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. Otherwise, we use the error guaranteed path refinement step in algorithm *Ref*, and we return $\Pi_{\textit{Ref-3}}(s,t)$ as output $\Pi(s,t)$, where the error bound is the same as in algorithm *LogSP-Adp*, i.e., $|\Pi_{\textit{Ref-3}}(s,t)| = |\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. Therefore, algorithm *Roug-Ref* guarantees that $|\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. □

Theorem 5. *The running time for algorithm FixSP is $O(n^3 \log n)$, and the memory usage is $O(n^3)$. It guarantees that $|\Pi_{\textit{FixSP}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{FixSP}}(s,t)$ is the path result returned by algorithm FixSP.*

Proof. Firstly, we prove both the *running time* and *memory usage*. By using algorithm *FixSP*, we need to place $O(n^2)$ Steiner points per edge on E [18]. Since there are total n edges, there are total $O(n^3)$ Steiner points in the weighted graph, so the running time and memory usage of using Dijkstra's algorithm, i.e., algorithm *FixSP*, on this graph are $O(n^3 \log n)$ and $O(n^3)$.

Secondly, we prove the *error bound*. Theorem 2.6 of study [17] shows that $|\Pi_{\textit{FixSP}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. □

Theorem 6. *The running time for algorithm FixSP-NoWei-Adp is $O(n^3 \log n)$, and the memory usage is $O(n^3)$. It guarantees that $|\Pi_{\textit{FixSP-NoWei-Adp}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where*

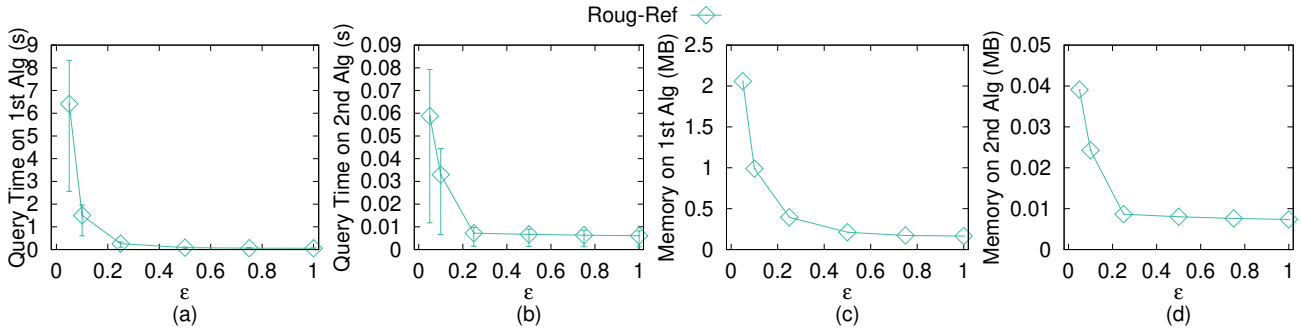

Fig. 41. Path Advisor user study (effect of ϵ on *PA* dataset with separated query time and memory usage in two steps)

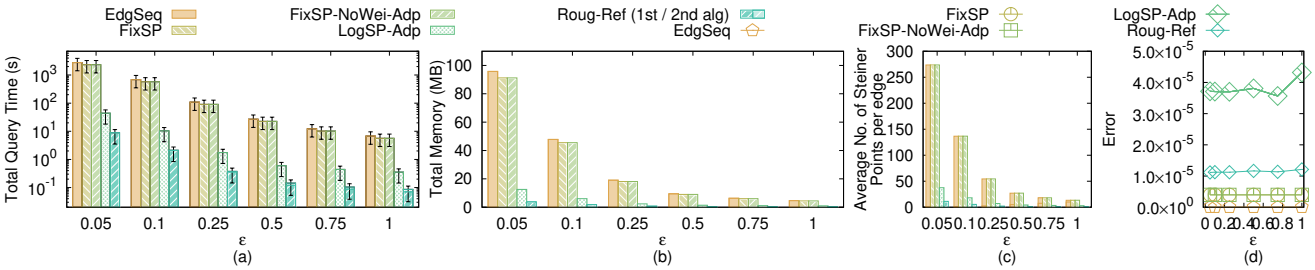

Fig. 42. Cable placement motivation study (effect of ϵ on *SB* dataset)

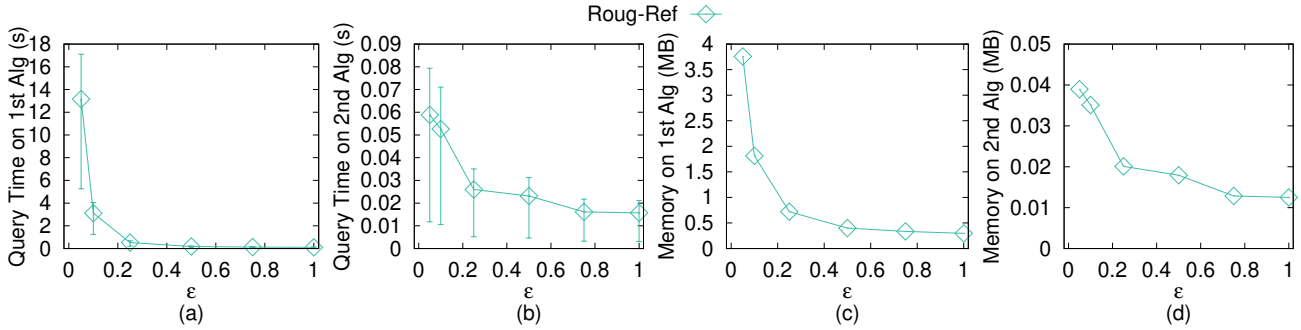

Fig. 43. Cable placement motivation study (effect of ϵ on *SB* dataset with separated query time and memory usage in two steps)

$\Pi_{FixSP\text{-}NoWei\text{-}Adp}(s,t)$ *is the path result returned by algorithm FixSP-NoWei-Adp.*

Proof. Since algorithm *FixSP-NoWei-Adp* is equivalent to algorithm *FixSP*, its running time, memory usage and error bound is the same as that of algorithm *FixSP*. □

Theorem 7. *The running time for algorithm LogSP is* $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$, *the memory usage is* $O(n \log \frac{LN}{\epsilon})$. *It guarantees that* $|\Pi_{LogSP}(s,t)| \leq (1 + (2 + \frac{2W}{(1-2\epsilon)\cdot w})\epsilon)|\Pi^*(s,t)|$, *where* $\Pi_{LogSP}(s,t)$ *is the path result returned by algorithm LogSP.*

Proof. Firstly, we prove both the *running time* and *memory usage*. By using algorithm *LogSP*, we need to place $k_{SP} = O(\log \frac{LN}{\epsilon})$ Steiner points per edge on E. Since there are total n edges, there are total $O(n^3)$ Steiner points in the weighted graph, so the running time and memory usage of using Dijkstra's algorithm, i.e., algorithm *LogSP*, on this graph are $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$ and $O(n \log \frac{LN}{\epsilon})$.

Secondly, we prove the *error bound*. Theorem 1 of study [4] shows that $|\Pi_{LogSP}(s,t)| \leq (1+(2+\frac{2W}{(1-2\epsilon)\cdot w})\epsilon)|\Pi^*(s,t)|$. □

Theorem 8. *The running time for algorithm LogSP-Adp is* $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$, *and the memory usage is* $O(n \log \frac{LN}{\epsilon})$. *It guarantees that* $|\Pi_{LogSP\text{-}Adp}(s,t)| \leq (1 + \epsilon)|\Pi^*(s,t)|$, *where* $\Pi_{LogSP\text{-}Adp}(s,t)$ *is the path result returned by algorithm LogSP-Adp.*

Proof. Firstly, we prove both the *running time* and *memory usage*. By using algorithm *LogSP-Adp*, we need to place $k_{SP} = O(\log \frac{LN}{\epsilon})$ Steiner points per edge on E. Since there are total n edges, there are total $O(n^3)$ Steiner points in the weighted graph, so the running time and memory usage of using Dijkstra's algorithm, i.e., algorithm *LogSP-Adp*, on this graph are $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$ and $O(n \log \frac{LN}{\epsilon})$.

Secondly, we prove the *error bound*. Theorem 1 of study [4] shows that $|\Pi_{LogSP}(s,t)| \leq (1 + (2 + \frac{2W}{(1-2\epsilon)\cdot w})\epsilon)|\Pi^*(s,t)|$. After substituting $(2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = \epsilon$ with $0 < \epsilon' < \frac{1}{2}$ and

TABLE IV
COMPARISON OF ALL ALGORITHMS

Algorithm	Time		Size		Error
ConWave [24]	$O(n^8 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n^4)$	Large	$1+\epsilon$
EdgSeq [29]	$O(n^3 \log n + n^4 \log(\frac{n^2 NWL}{w\epsilon}))$	Large	$O(n^3)$	Large	$1+\epsilon$
FixSP [13], [18]	$O(n^3 \log n)$	Large	$O(n^3)$	Large	$1+\epsilon$
FixSP-NoWei-Adp [14]	$O(n^3 \log n)$	Large	$O(n^3)$	Large	$1+\epsilon$
LogSP [4]	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	Medium	$O(n \log \frac{LN}{\epsilon})$	Medium	$1+(2+\frac{2W}{(1-2\epsilon)\cdot w})\epsilon$
LogSP-Adp [4]	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	Medium	$O(n \log \frac{LN}{\epsilon})$	Medium	$1+\epsilon$
Roug-Ref-Naive1	$O(n^3 \log n + nl^2 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n^3 + nl)$	Large	$1+\epsilon$
Roug-Ref-NoEffSP	$O(n^2 \log n + l)$	Large	$O(n^2 + l)$	Large	$1+\epsilon$
Roug-Ref-Naive2	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + nl^2 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n \log \frac{LN}{\epsilon} + nl)$	Medium	$1+\epsilon$
Roug-Ref-NoEdgSeqConv	$O(n \log n + nl)$	Medium	$O(nl)$	Small	$1+\epsilon$
Roug-Ref-NoEffWeig	$O(n \log n + l^2 \log(\frac{lNWL}{w\epsilon}))$	Medium	$O(n+l)$	Small	$1+\epsilon$
Roug-Ref-NoPrunDijk	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$	Medium	$O(n \log \frac{LN}{\epsilon} + l)$	Medium	$1+\epsilon$
***Roug-Ref* (ours)**	$\boldsymbol{O(n \log n + l)}$	**Small**	$\boldsymbol{O(n+l)}$	**Small**	$\boldsymbol{1+\epsilon}$

$\epsilon > 0$, we have $|\Pi_{\textit{LogSP-Adp}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$ where $\epsilon > 0$. □

Theorem 9. *The running time for algorithm EdgSeq is $O(n^3 \log n + n^4 \log(\frac{n^2 NWL}{w\epsilon}))$, and the memory usage is $O(n^3)$. It guarantees that $|\Pi_{\textit{EdgSeq}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{EdgSeq}}(s,t)$ is the path result returned by algorithm EdgSeq.*

Proof. Firstly, we prove both the *running time*. Algorithm *EdgSeq* first uses algorithm *FixSP* to calculate an edge sequence in $O(n^3 \log n)$ time, then uses Snell's law in a binary search manner to calculate the result path in $O(l'^2 \log \frac{L}{\delta})$ time according to Lemma 2.4 of study [29], where l' is the number of edges in the edge sequence calculated by algorithm *FixSP* and $\delta = \frac{h\epsilon w}{6l'W}$. According to Lemma 7.1 of study [24], $l' = O(n^2)$. Thus, the running time of algorithm *EdgSeq* is $O(n^3 \log n + n^4 \log(\frac{n^2 NWL}{w\epsilon}))$.

Secondly, we prove the *total memory usage*. Algorithm *EdgSeq* first uses algorithm *FixSP* to calculate an edge sequence with $O(n^3)$ memory usage, then uses Snell's law in a binary search manner to calculate the result path with $O(l')$ memory usage since the result path will pass l' edges. Since $l' = O(n^2)$, the memory usage of algorithm *EdgSeq* is $O(n^3)$.

Thirdly, we prove the *error bound*. Algorithm *EdgSeq* first uses algorithm *FixSP* to calculate a $(1+\epsilon)$-approximate weighted shortest path, then uses Snell's law to calculate the result path with a smaller error. Due to the error bound of algorithm *FixSP*, we have $|\Pi_{\textit{EdgSeq}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. □

Theorem 10. *The total running time for algorithm Roug-Ref-NoEffSP is $O(n^2 \log n + l)$, and the total memory usage is $O(n^2 + l)$. It guarantees that $|\Pi_{\textit{Roug-Ref-NoEffSP}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{Roug-Ref-NoEffSP}}(s,t)$ is the path result returned by algorithm Roug-Ref-NoEffSP.*

Proof. The difference between *Roug-Ref-NoEffSP* and *Roug-Ref* is that in the former one, we use *FixSP* to substitute *LogSP* as Steiner points placement scheme in algorithm *Roug* and the error guaranteed path refinement in *Ref*.

Firstly, we prove both the *running time* and *memory usage*. From Theorem 5, we know that by using *FixSP*, we need to place $O(n^2)$ Steiner points per edge on E [18]. Since there are total n edges, there are total $O(n^3)$ Steiner points in the weighted graph, so the running time and memory usage of using Dijkstra's algorithm on this graph are $O(n^3 \log n)$ and $O(n^3)$. By using the framework of *Roug-Ref*, we obtain the total running time and total memory usage, i.e., $O(n^2 \log n + l)$ and $O(n^2 + l)$.

Secondly, we prove the *error bound*. From Theorem 5, we know that $|\Pi_{\textit{FixSP}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{FixSP}}(s,t)$ is the path result returned by algorithm *FixSP*. Since from Theorem 1, we also have $|\Pi_{\textit{LogSP}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{LogSP}}(s,t)$ is the path result returned by algorithm *LogSP*, so by using the framework of *Roug-Ref*, we obtain the error bound of algorithm *Roug-Ref-NoEffSP*. □

Theorem 11. *The total running time for algorithm Roug-Ref-NoEdgSeqConv is $O(n \log n + nl)$, and the total memory usage is $O(nl)$. It guarantees that $|\Pi_{\textit{Roug-Ref-NoEdgSeqConv}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{Roug-Ref-NoEdgSeqConv}}(s,t)$ is the path result returned by algorithm Roug-Ref-NoEdgSeqConv.*

Proof. The difference between *Roug-Ref-NoEdgSeqConv* and *Roug-Ref* is that in the former one, we remove the full edge sequence conversion step in algorithm *Ref* (such that the input edge sequence of the Snell's law path refinement step in algorithm *Ref* may be a non-full edge sequence, then we need to try Snell's law on different combinations of edges and select the result path with the minimum length). This will only affect the total running time and the total memory usage. Thus, we only prove the total running time and the total memory usage.

For both the *total running time* and the *total memory usage*, there are total n different cases of the edge sequence that we need to perform Snell's law on, that is, we need to use the Snell's law path refinement step in algorithm *Ref* for n times, and select the path with the shortest distance. Therefore, the total running time and the total memory usage need to include the Snell's law path refinement step in algorithm *Ref* for n

times. By using the framework of *Roug-Ref*, we obtain the total running time and total memory usage, i.e., $O(n \log n + nl)$ and $O(nl)$. □

Theorem 12. *The total running time for algorithm Roug-Ref-NoEffWeig is* $O(n \log n + l^2 \log(\frac{lNWL}{w\epsilon}))$*, and the total memory usage is* $O(n + l)$*. It guarantees that* $|\Pi_{\textit{Roug-Ref-NoEffWeig}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$*, where* $\Pi_{\textit{Roug-Ref-NoEffWeig}}(s,t)$ *is the path result returned by algorithm Roug-Ref-NoEffWeig.*

Proof. The difference between *Roug-Ref-NoEffWeig* and *Roug-Ref* is that in the former one, we remove the effective weight pruning out sub-step in the Snell's law path refinement step of algorithm *Ref*. This will only affect the total running time and the total memory usage. Thus, we only prove the total running time and the total memory usage.

Firstly, we prove the *total running time*. Let l be the number of edges in S. In the binary search initial path and binary search refined path finding sub-step, it first takes $O(l)$ time to compute the 3D surface Snell's ray Π_m since there are l edges in S and we need to use Snell's law l times to calculate the intersection point on each edge. Then, it takes $O(\log \frac{L_i}{\delta})$ time to decide the position of m_i because we stop the iteration when $|a_i b_i| < \delta$, and they are binary search approach, where L_i is the length of e_i. Since $\delta = \frac{h\epsilon w}{6lW}$ and $L_i \leq L$ for $\forall i \in \{1, \ldots, l\}$, the running time is $O(\log \frac{lWL}{h\epsilon w})$. Since we run the above two nested loops l times, the total running time is $O(l^2 \log \frac{lWL}{h\epsilon w})$. So the running time of the Snell's law path refinement step without the effective weight pruning sub-step in algorithm *Ref* is $O(l^2 \log \frac{lWL}{h\epsilon w})$. By using the framework of *Roug-Ref*, we obtain the total running time is $O(n \log n + l^2 \log(\frac{lNWL}{w\epsilon}))$.

Secondly, we prove the *total memory usage*. Since the refined path will pass l edges, so the memory usage of the Snell's law path refinement step without the effective weight pruning sub-step in algorithm *Ref* is $O(l)$. By using the framework of *Roug-Ref*, we obtain the total memory usage is $O(n + l)$. □

Theorem 13. *The total running time for algorithm Roug-Ref-NoPrunDijk is* $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$*, and the total memory usage is* $O(n \log \frac{LN}{\epsilon} + l)$*. It guarantees that* $|\Pi_{\textit{Roug-Ref-NoPrunDijk}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$*, where* $\Pi_{\textit{Roug-Ref-NoPrunDijk}}(s,t)$ *is the path result returned by algorithm Roug-Ref-NoPrunDijk.*

Proof. The difference between *Roug-Ref-NoPrunDijk* and *Roug-Ref* is that in the former one, we do not use the node information for pruning out in Dijkstra's algorithm in the error guaranteed path refinement step of algorithm *Ref*. This will only affect the total running time and the total memory usage. Thus, we only prove the total running time and the total memory usage.

Firstly, we prove the *total running time*. (1) In most of the cases, there is no need to use the error guaranteed path refinement step in algorithm *Ref*. In this case, the total running time is the sum of the running time using algorithm *Roug* and the first three steps in algorithm *Ref*. From Theorem 2, Theorem 3 and Theorem 4, we obtain the total running time $O(n \log n + l)$. (2) In some special cases, we need to use the error guaranteed path refinement step in algorithm *Ref* for error guarantee. Since we do not use the node information for pruning out in Dijkstra's algorithm in the error guaranteed path refinement step of algorithm *Ref*, we need to perform Dijkstra's algorithm on the weighted graph $G_{\textit{Ref}}$ constructed by the original Steiner points (i.e., $k_{SP} = O(\log \frac{LN}{\epsilon})$ Steiner points per edge) and V in the error guaranteed path refinement step, and the running time is $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$, so we obtain the total running time $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$. (3) In general, the total running time is $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$.

Secondly, we prove the *total memory usage*. (1) In most of the cases, there is no need to use the error guaranteed path refinement step in algorithm *Ref*. In this case, the total memory usage is the sum of the memory usage using algorithm *Roug* and the first three steps in algorithm *Ref*. From Theorem 2, Theorem 3 and Theorem 4, we obtain the average case total memory usage $O(n + l)$. (2) In some cases, we need to use the error guaranteed path refinement step in algorithm *Ref* for error guarantee. Since we do not use the node information for pruning out in Dijkstra's algorithm in the error guaranteed path refinement step of algorithm *Ref*, we need to perform Dijkstra's algorithm on the weighted graph $G_{\textit{Ref}}$ constructed by the original Steiner points (i.e., $k_{SP} = O(\log \frac{LN}{\epsilon})$ Steiner points per edge) and V in the error guaranteed path refinement step, and the memory usage is $O(n \log \frac{LN}{\epsilon})$, so we obtain the total memory usage $O(n \log \frac{LN}{\epsilon} + l)$. (3) In general, the total memory usage is $O(n \log \frac{LN}{\epsilon} + l)$. □