

# Efficiently Finding Shortest Path on 3D Weighted Terrain Surfaces

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**—Nowadays, the rapid development of computer graphics technology and geo-spatial positioning technology promotes the growth of using digital terrain data. Studying the shortest path query on terrain data has aroused widespread concern in industry and academia. In this paper, we propose an efficient method for the *weighted region problem* on a 3D weighted terrain surface. The problem involves finding the shortest path between two points passing different regions on the terrain surface and different regions are assigned different weights. Since it has been proved that, even in a 2D environment, there is no exact algorithm for finding the exact solution of the weighted region problem efficiently when the number of faces in the terrain is greater than two, we propose a  $(1 + \epsilon)$ -approximate efficient method to solve it on the terrain surface. In both the theoretical and practical analysis, our algorithm gives a shorter running time and less memory usage compared with the best-known algorithm.

## I. INTRODUCTION

In recent years, digital terrain data becomes increasingly widespread in industry and academia [39]. In industry, many commercial companies/applications, such as Metaverse [7], [30], [31], Cyberpunk 2077 (a popular 3D computer game) [2] and Google Earth [6], are using terrain data of objects such as mountains, valleys, and hills with different features (e.g., water and grassland) to help users reach the destination faster. In academia, researchers paid considerable attention to studying shortest path queries on terrain datasets [18], [24], [25], [33], [36], [37], [41], [40]. A terrain surface is represented by a set of *faces* each of which corresponds to a triangle. Each face (or triangle) has three line segments called *edges* connected with each other at three *vertices*. Figure 1 shows an example of a terrain surface. The *weighted shortest path* (resp. *unweighted shortest path*) on a terrain refers to the shortest path between a source  $s$  and a destination  $t$  that passes the face on the terrain where each face is assigned with a *weight* (resp. each face weight is set to a fixed value, e.g., 1). In Figure 1, the blue (resp. purple dashed) line denotes the weighted (resp. unweighted) shortest path from  $s$  to  $t$  on this terrain surface.

### A. Motivation

Given a source  $s$  and a destination  $t$ , computing the weighted shortest path on the terrain surface between  $s$  and  $t$  with different meanings of the face weights is involved in numerous applications, including autonomous vehicles' obstacle avoidance path planning, human's route-recommendation systems and laying electrical cables [14], [22], [23], [27], [41],

[42]. In Figure 1, a robot wants to move on a 3D terrain surface from  $s$  to  $t$  which consists of water (the faces with a blue color) and grassland (the faces with a green color), and avoid passing through the water. We can set the terrain faces corresponding to water (resp. grassland) with a larger (resp. smaller) weight. So, the weighted length of the path that passes water is larger, and the robot will choose the path that does not pass water (i.e., the weighted length is smaller). In a real-life example for placement of undersea optical fiber cable on the seabed, i.e., a terrain, we aim to minimize the weighted length of the cable for cost saving (over  $1.35 \times 10^5$  km of undersea cables have been constructed nowadays [26]). For the regions with a deeper sea level, the hydraulic pressure is higher, and the cable's lifespan is reduced, so it is more expensive to repair and maintain the cable [15]. We set the terrain faces for this type of regions with a larger weight. So, we can avoid placing the cable on these regions, and reduce the cost. Our motivation study shows that the total estimated cost of the cable for following the weighted shortest path and the unweighted shortest path are USD \$45.8M and \$54.8M, respectively, which shows the usefulness of the weighted shortest path. Motivated by these, we aim to solve the problem of finding the shortest path on a 3D terrain surface between two points passing different regions on the terrain surface and different regions are assigned with different weights, which is called the *weighted region problem*. The weight on the terrain surface is usually set according to the problem.

### B. Limitations in existing work

Consider a terrain  $T$  with  $n$  vertices. Let  $V$ ,  $E$ , and  $F$  be the set of vertices, edges, and faces of the terrain, respectively. There is no *exact* solution for solving the weighted region problem when the number of faces in the terrain is larger than two [17]. There are three categories of algorithms for solving the weighted region problem approximately: (1) *Continuous Wavefront (ConWave)* approach [34], (2) *heuristics* approach [27], [38], and (3) *Steiner point* approach [24], [29].

1) **ConWave approach:** It uses Snell's law (one widely known fact from physics) [9] and continuous Dijkstra's algorithm [34] to calculate an approximate weighted shortest path in  $O(n^8 \log c_1)$  time, which is very large, since it involves many redundant checks, where  $c_1$  is a constant depending on the error and geometry information of  $T$ .

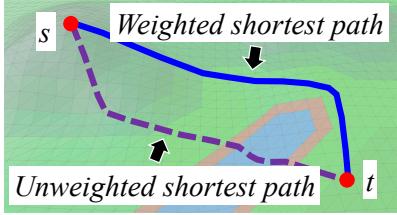


Fig. 1. A terrain surface and paths

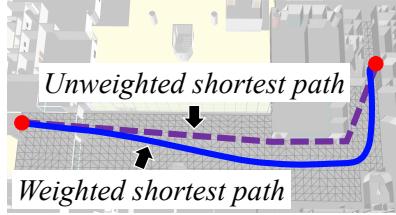


Fig. 2. Paths in Path Advisor

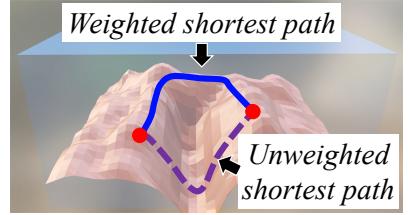


Fig. 3. Paths on seabed

2) **Heuristics approach:** It uses simulated annealing [27] and genetic algorithm [38] to calculate the weighted shortest path, but it cannot guarantee on the quality of the result path returned (in terms of the length of the path) with a given time limit and a given maximum memory.

3) **Steiner point approach:** It places discrete points (i.e., Steiner points) on edges in  $E$ , and then uses Dijkstra's algorithm [20] on a weighted graph constructed using these Steiner points and  $V$ , to calculate the weighted shortest path. The best-known existing studies [24], [29] run in  $O(n^3 \log n)$ , which are still very slow, since it does not utilize any geometry information on  $T$  and thus, need to place many Steiner points on each edge. Our experimental result shows that for a terrain with 50k faces, the best-known algorithm [24], [29] runs in 119,000s ( $\approx 1.5$  days), which is not acceptable.

### C. Our Efficient Algorithm

We propose an efficient two-step algorithm to calculate the weighted shortest path in the 3D weighted region problem using algorithm *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  $t$  without unfolding any face in  $T$ , where  $\epsilon$  is a non-negative real user parameter for controlling the error ratio, called the *error parameter*. Algorithm *Roug-Ref* is a step-by-step algorithm involving algorithm *Roug* and then algorithm *Ref*. (1) In algorithm *Roug*, given  $T$ ,  $s$ ,  $t$ , and  $\epsilon$ , we efficiently find a rough path between  $s$  and  $t$  with error guarantee  $(1 + \eta\epsilon)$ , where  $\eta > 1$  is a constant, and it is calculated based on  $T$  and  $\epsilon$ . (2) In algorithm *Ref*, given the rough path, we efficiently refine this path to be a  $(1 + \epsilon)$ -approximate weighted shortest path.

*Snell's law* [9] is an important geometry information of  $T$  between two adjacent faces in  $F$  that share one edge. When a light ray passes the boundary of two different media (e.g., air and water), it bends at the boundary due to the fact that light seeks the path with minimum time, as described by *Fermat's Principle* [5]. The angles of incidence and refraction for the light satisfy Snell's law. The blue line in Figure 4 is a weighted shortest path that satisfies Snell's law from  $s$  to  $t$  that passes edges  $e_1, \dots, e_l$  in order.

Our algorithm employs Snell's law to efficiently calculate the weighted shortest path, the feature that is absent in the best-known existing works [24], [29]. In order to (1) enable the rapid reduction of the search area for the weighted shortest path from the entire  $T$  to certain faces, edges, or vertices on  $T$  when Snell's law is applied, and (2) provide an error guarantee

for the refined path after Snell's law is utilized, we first need to find an approximate path using algorithm *Roug*. Algorithm *Roug-Ref* does not fall into any category in Section I-B.

### D. Contribution and Organization

We summarize our major contributions.

(1) To the best of our knowledge, we are the first to propose the novel and efficient algorithm *Roug-Ref* that calculates a  $(1 + \epsilon)$ -approximate weighted shortest path, since there is no existing work that uses the rough-refine idea for calculating the weighted shortest path within the error bound. We also have four additional novel techniques to efficiently reduce the algorithm running time and memory usage, including: (a) an *efficient* Steiner point placement scheme in algorithm *Roug*, which considers geometry information on  $T$  for each face in  $F$ , such as the weight of a face, the internal angle of a face, the length of an edge, etc., that is not considered in the best-known algorithm [24], [29], (b) the use of a *progressive* idea in algorithm *Ref* to further reduce the searching region, (c) the consideration of additional geometry information in algorithm *Ref*, called *effective weight*, for pruning, and (d) the development of a *novel* approach in algorithm *Ref* to handle cases when we are unable to use Snell's law to refine a rough path to a  $(1 + \epsilon)$ -approximate weighted shortest path (although this only occurs in less than 1% of cases), so that even if an additional step is required for error bound guarantee, the total running time of algorithm *Roug-Ref* remains comparable to the adapted best-known algorithm.

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

(3) Our algorithm outperforms the best-known algorithm [24], [29] in terms of running time and memory usage, making it highly suitable for real-life applications that require real-time responses. Our experimental results show that our algorithm runs up to 1630 times faster than the best-known algorithm [24], [29] on benchmark real datasets with the same error ratio. For instance, for a terrain 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 algorithm [24], [29] runs in 119,000s ( $\approx 1.5$  days) and uses 2.9GB of memory. Furthermore, in our user study in Section V-B6, our algorithm takes only 0.1s to calculate the result for a real-time map application.

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.

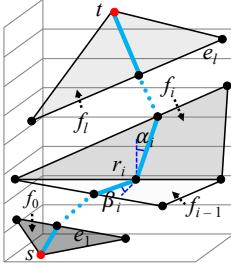


Fig. 4. An example of  $\Pi^*(s, t)$

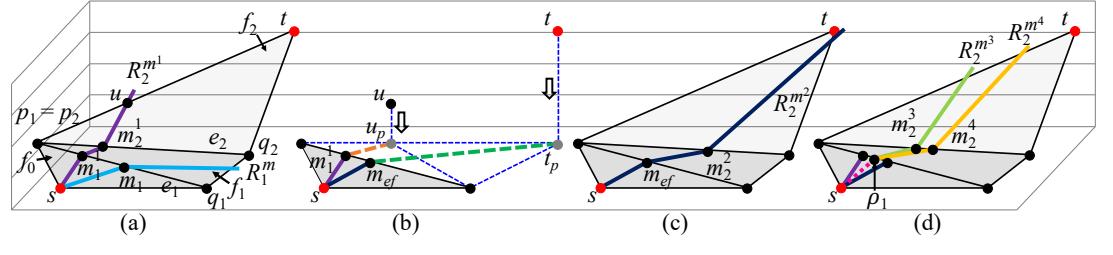


Fig. 5. Snell's law path refinement step in Ref. (a) with initial ray for calculating effective weight on the effective face  $\Delta u_p p_1 q_1$ , (b) for calculating  $m_{ef}$  using the weight of  $f_0$  and the effective weight of  $\Delta u_p p_1 q_1$ , (c) with final ray passing through  $m_{ef}$ , and (d) processing on the remaining edges

## II. PRELIMINARY

### A. Notation and Definition

1) **Terrain and path:** Consider a terrain  $T$ . Let  $V$ ,  $E$ , and  $F$  be the set of vertices, edges, and faces of  $T$ , respectively. Let  $n$  be the number of vertices of  $T$ . Each vertex  $v \in V$  has three coordinate values, denoted by  $x_v$ ,  $y_v$  and  $z_v$ . If two faces share a common edge, they are said to be adjacent. Each face  $f_i \in F$  is assigned a weight  $w_i$ , which is a positive real number, and the weight of an edge is the smaller weight of the two faces containing the edge. Given a face  $f_i$ , and two points  $p$  and  $q$  on  $f_i$ , we define  $d(p, q)$  to be the Euclidean distance between  $p$  and  $q$  on  $f_i$ , and  $D(p, q) = w_i \cdot d(p, q)$  to be the weighted surface distance from  $p$  to  $q$  on  $f_i$ . Given  $s$  and  $t$ , the weighted region problem aims to find the optimal weighted shortest path  $\Pi^*(s, t) = (s, r_1, \dots, r_l, t)$  on  $T$  (with  $l \geq 0$ ) such that the weighted distance  $\sum_{i=0}^l D(r_i, r_{i+1})$  is minimum, where  $r_0 = s$ ,  $r_{l+1} = t$ , each  $r_i$  for  $i \in \{1, \dots, l\}$  is named as a *intersection point* in  $\Pi^*(s, t)$ , and it is a point on an edge in  $E$ . The blue line in Figure 4 shows an example of  $\Pi^*(s, t)$  on a terrain surface. We define  $|\cdot|$  to be the weighted distance of a path (e.g.,  $|\Pi^*(s, t)|$  is the weighted distance of  $\Pi^*(s, t)$ ). Given a face  $f_i$ , and two points  $p$  and  $q$  on  $f_i$ , we define  $\bar{pq}$  to be a line segment on  $f_i$ . Let  $\Pi(s, t)$  be the final calculated weighted shortest path of our algorithm, algorithm *Roug-Ref* guarantees  $|\Pi(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ .

2) **Snell's Law:** In Figure 4, let  $S = ((v_1, v'_1), \dots, (v_l, v'_l)) = (e_1, \dots, e_l)$  be a sequence of edges that  $\Pi^*(s, t)$  connects from  $s$  to  $t$  in order based on  $T$ , and  $S$  is said to be *passed by*  $\Pi^*(s, t)$ . Let  $F(S) = (f_0, f_1, \dots, f_{l-1}, f_l)$  be a sequence of adjacent faces with respect to  $S$ , such that for every  $f_i$  with  $i \in \{1, \dots, l-1\}$ ,  $f_i$  is the face containing  $e_i$  and  $e_{i+1}$  in  $S$ , while  $f_0$  is the adjacent face of  $f_1$  at  $e_1$  and  $f_l$  is the adjacent face of  $f_{l-1}$  at  $e_l$ . Let  $W(S) = (w_0, w_1, \dots, w_{l-1}, w_l)$  be a weight list with respect to  $F(S)$  such that for every  $w_i$  with  $i \in \{0, \dots, l\}$ ,  $w_i$  is the face weight of  $f_i$  in  $F(S)$ . We define  $\alpha_i$  and  $\beta_i$  to be the angles of incidence and refraction of  $\Pi^*(s, t)$  on  $e_i$  for  $i \in \{1, \dots, l\}$ , respectively. Proposition 1 states Snell's law with  $i \in \{1, \dots, l\}$ .

**Proposition 1.**  $\Pi^*(s, t)$  has  $w_i \cdot \sin \alpha_i = w_{i-1} \cdot \sin \beta_i$ .

3) **Two types of queries:** There are two types of queries, (1) *vertex-to-vertex (V2V) path query*, i.e., both  $s$  and  $t$  are in

$V$ , and (2) *arbitrary point-to-arbitrary point (A2A) path query*, i.e.,  $s$  and  $t$  are two arbitrary points on the surface of  $T$ . If  $s$  or  $t$  is not in  $V$ , we can add new triangles between them and their neighbour vertices in  $T$  to make them as vertices, and then the A2A path query can be regarded as one form of the V2V path query. We focus on the V2V path query in this paper and study the A2A path query in the appendix. A notation table appears in the appendix of Table III.

### B. Challenges

1) **Different from unweighted case:** Solving the 3D weighted region problem differs significantly from solving the unweighted case. In the unweighted case, the state-of-the-art solution for finding the *exact* shortest path is to unfold the 3D terrain surface into a 2D terrain, and connect the source  $s$  and destination  $t$  using a straight line segment [16]. However, this approach cannot be used for the weighted case. Even in 2D, when finding the *exact* weighted shortest path (i.e., the path with the minimum total distance that satisfies Snell's law), if the path passes through the boundary between two faces with different weights, it will bend when it crosses an edge in  $E$  every time [34]. Therefore, we cannot connect  $s$  and  $t$  with a straight line segment in the unfolded 2D terrain.

2) **No exact solution:** There is no known *exact* solution for solving the weighted region problem when the number of faces in the terrain is larger than two [17]. In Figure 4, given two points  $s$  and  $t$ , on the first edge  $e_1$  opposite to  $s$ , if we can find a point  $c$  on  $e_1$ , such that there is a path starting from  $s$ , passing  $c$ , and then following Snell's law when it crosses an edge, and finally going through  $t$ , with the minimum distance, then we can find the *exact* weighted shortest path, where  $c$  is called *optimal point*. One may assume that we can set the position of  $c$  to be unknown, then solve it using a polynomial equation. But, even if the weighted shortest path crosses only three faces, the equation will contain unknown with a degree of six, which cannot be solved using Algebraic Computation Model over the Rational Numbers (ACM $\mathbb{Q}$ ) [17].

## III. RELATED WORK

### A. Algorithms on the weighted terrain surface

1) **ConWave approach:** It aims to calculate the weighted shortest path which follows Snell's law on a continuous

surface by exploiting Snell's law using continuous Dijkstra [34] algorithm. The algorithm [34] returns a path with  $(1 + \epsilon)$ -approximation weighted shortest distance in  $O(n^8 \log(\frac{LNWL}{w\epsilon}))$  time, which is computationally expensive. This is because when we calculate a path that hits a vertex in  $V$ , if we want to utilize Snell's law for the path after it exits this vertex, we no longer have complete information about where it goes next. But, the work [34] uses a continuous Dijkstra's algorithm to check all the combinations of the cases when a path hits a vertex that may happen on  $T$ , and thus, their running time is very large. As we will discuss later, if we can find a sequence of edges  $S$  that the optimal weighted shortest path passes, then uses Snell's law on  $S$  to find the result path and thus, the running time can be reduced (this is the technique used by us).

2) **Heuristics approach:** It uses simulated annealing [27] and genetic algorithm [38] to calculate the weighted shortest path, but it cannot guarantee on the quality of the result path returned (in terms of the length of the path) with a given time limit and a given maximum memory.

3) **Steiner point approach:** It uses Dijkstra's algorithm on the weighted graph constructed using Steiner points on edges in  $E$  and  $V$  to calculate the weighted shortest path. Even though the work [24], [29], i.e., algorithm *Fixed Steiner Point placement scheme (FixSP)*, is regarded as the best-known algorithm for solving the weighted region problem and runs in  $O(n^3 \log n)$  time, it is still very slow. Because algorithm *FixSP* (3a) does not utilize any geometry information on  $T$  between two adjacent faces in  $F$  that share one edge, i.e., Snell's law, so they need to place many Steiner points (and each two Steiner points on the same edge are very close to each other) on edges in  $E$ , and (3b) does not utilize any geometry information on  $T$  for each face in  $F$ , e.g., the weight of a face, the internal angle of a face, the length of an edge, etc., so they always place the same numbers of Steiner points on each edge in  $E$  (i.e., in order to bound the error, no matter how  $T$  looks like, they always need to place  $O(n^2)$  Steiner points per edge to bound the error [29]), so their running time and memory usage are still very large. Our experimental result shows that for a terrain with 50k faces and  $\epsilon = 0.1$ , our algorithm just needs to place about 10 Steiner points per edge to find a rough path in 71s ( $\approx 1.2$  min), then we can use Snell's law on the edge sequence passed by the rough path to calculate a refined path in 2s with total memory usage 43MB, but the best-known algorithm [24], [29] needs to place more than 600 Steiner points per edge to find the result path in 119,000s ( $\approx 1.5$  days) with total memory usage 2.9GB.

Algorithm [24] is originally designed for finding the unweighted shortest path, we could adapt it for solving the weighted region problem. Specifically, given a face  $f_i$ , and two points  $p$  and  $q$  on  $f_i$ , in the unweighted case, on the graph constructed by the Steiner point and  $V$ , the weight of an edge in the graph between  $p$  to  $q$  on  $f_i$  is  $d(p, q)$ , i.e., the Euclidean distance between  $p$  and  $q$ . After the adaption to the weighted case, we assign each face a weight, so the weight of an edge in the weighted graph is  $w_i \cdot d(p, q)$ , where  $w_i$  is the weight of  $f_i$ .

Algorithm [24] is the most recent and most efficient *on-the-fly* algorithm. Furthermore, the reported running time of algorithm [24] in the unweighted case is  $O((n + n') \log(n + n'))$ , where  $n'$  is the total number of Steiner points. After adaption to the weighted case, we have  $n' = n^3$  [29], so we obtain the adapted running time of algorithm [24], i.e.,  $O(n^3 \log n)$ , and applying algorithm [24] on a weighted terrain surface is essentially equivalent to that of algorithm [29].

## B. Other related work

There are some other work related to our problem although they are not exactly our problem. (1) *Unweighted shortest path algorithm* [12], [16]: As mentioned in Section II-B1, algorithm [16] is recognized as the best-known algorithm for the unweighted shortest path calculation, but it cannot be adapted to weighted case. Algorithm [12] uses finite element for unweighted shortest path calculation, but it cannot guarantee on the quality of the result path returned (in terms of the length of the path) with a given time limit and a given maximum memory. There is no known algorithm to adopt both algorithms [12], [16] into the weighted case. (2) *Oracle-based algorithm* [36], [37]: The work [36], [37] uses the best-known algorithm [24] (before adaption to the weighted case) as *on-the-fly* algorithm to build an *oracle* on an unweighted terrain surface. After we adapt algorithm [24] to the weighted case, the oracle [36], [37] can be adapted on a weighted terrain surface. But, the fundamental part of the adaption is about the *on-the-fly* algorithm, not the oracle. With different *on-the-fly* algorithms on other data structures, e.g., a graph or a road network, they can use the same oracle [36], [37] for indexing. We focus on *on-the-fly* algorithm for the weighted region problem in this paper.

## IV. METHODOLOGY

### A. Overview

1) **Four concepts:** We give four concepts, (a) *weighted graph*, (b) *removing value*, (c) *node information*, and (d) *full or non-full edge sequence*.

(a) **The weighted graph:** It is used for calculating the shortest path by Dijkstra's algorithm. Let  $G_A = (G_A.V, G_A.E)$  be a weighted graph used in algorithm *Roug* or *Ref*, where  $G_A.V$  and  $G_A.E$  are the sets of nodes and weighted edges of  $G_A$ ,  $A$  is a placeholder and can be *Roug* or *Ref*. To construct  $G_A$ , we first define a set of Steiner points on each edge of  $E$  as *SPA*, where  $A$  can be *Roug* or *Ref*. We let  $G_A.V = SPA \cup V$ . As we will introduce later,  $G_{Roug}.V \subseteq G_{Ref}.V$ . 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  $\overline{pq}$ , and let  $G_A.E$  be a set of weight edges. The weight for edge  $\overline{pq}$  is  $w_{pq} \cdot d(p, q)$ , where  $w_{pq}$  means the weight of the face or edge that both  $p$  and  $q$  lie on. In Figure 6, (i)  $SP_{Roug}$  is denoted as three blue nodes, (ii)  $SP_{Ref}$  is denoted as three blue nodes and six green nodes, (iii)  $G_{Roug}.V$  is denoted as three blue nodes and three black nodes, (iv)  $G_{Ref}.V$  is denoted as three blue nodes, six green nodes and three black nodes, and (v) the purple line is a weighted edge  $\overline{bc}$  with weight 3 in  $G_{Ref}.E$ .

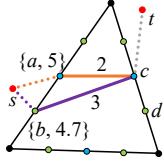


Fig. 6. Pruned Dijkstra's algorithm

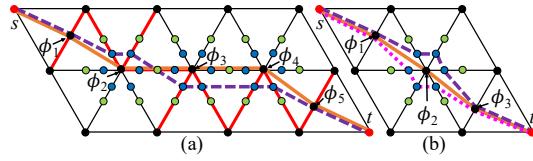


Fig. 7. (a) Successive endpoint case, and (b) single endpoint case

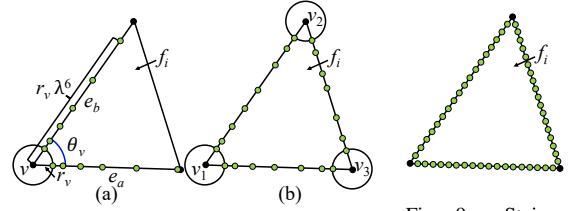


Fig. 8. Steiner points on (a)  $e_a$  and  $e_b$ , and (b)  $f_i$  in our algorithm

Fig. 9. Steiner points on  $f_i$  in FixSP

(b) **The removing value:** It is a constant (denoted by  $k$ ) used for calculating  $SP_{Roug}$ . In Figure 10 (b), we place Steiner points for  $SP_{Ref}$  and then move from two endpoints  $v_1$  and  $v_2$  on each edge. Whenever we encounter a point, we keep one and iteratively remove the next  $k = 1$  points to obtain a set of remaining Steiner points,  $SP_{Roug}$ . In our experiments, we usually set  $k$  to 2.

(c) **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 let  $dist_A(u)$  be the current shortest distance from  $s$  to node  $u$ , let  $prev_A(u)$  be the previous node on the shortest path from  $s$  to node  $u$ . So, after algorithm *Roug*, *node information* stores the shortest distance from  $s$  to each node  $G_{Roug}.V$  and each node's previous node along the shortest path starting from  $s$ , i.e.,  $dist_{Roug}(u)$  and  $prev_{Roug}(u)$  for each  $u \in G_{Roug}.V$ . In Figure 6, after algorithm *Roug*, the shortest path from  $s$  to  $c$  is  $s \rightarrow a \rightarrow c$  with distance  $5 + 2 = 7$  and previous node  $a$  (where the number next to  $a$  and  $b$  are the shortest distance from  $s$  to  $a$  and  $b$ , respectively), so we have  $dist_{Roug}(c) = 5 + 2 = 7$  and  $prev_{Roug}(c) = a$  as node information of  $c$ .

(d) **The full or non-full edge sequence:** Given an edge sequence  $S = ((v_1, v'_1), \dots, (v_l, v'_l)) = (e_1, \dots, e_l)$ ,  $S$  is said to be a *full edge sequence* if the length of each edge in  $S$  is larger than 0. In Figure 7 (a), given the path in the purple dashed line between  $s$  and  $t$ , the edge sequence  $S_a$  (in red) passed by this path is a full edge sequence since the length of each edge in  $S_a$  is larger than 0. Similarly, given an edge sequence  $S$ ,  $S$  is said to be a *non-full edge sequence* if there exists at least one edge whose length is 0. In Figure 7 (a), given the path in the orange line between  $s$  and  $t$ , the edge sequence  $S_b$  passed by this path is a non-full edge sequence since the edge length at  $(\phi_2, \phi_2)$ ,  $(\phi_3, \phi_3)$  and  $(\phi_4, \phi_4)$  are 0. As we will discuss later, a full (resp. non-full) edge sequence can reduce (resp. increase) the running time in algorithm *Ref*.

2) **Overview of algorithm *Roug*:** Given a terrain  $T$ , two vertices  $s$  and  $t$  in  $V$ , the error parameter  $\epsilon$ , and the remaining value  $k$ , we find a rough path between  $s$  and  $t$  with error guarantee  $(1+\eta\epsilon)$ , and store the shortest distance and previous node as node information. There are two steps:

- **$\eta$  calculation:** In Figure 10 (b), given  $T$ ,  $\epsilon$ , and  $k$ , we place Steiner points using  $\epsilon$  with an efficient Steiner point placement scheme to get  $SP_{Ref}$ , remove some points using  $k$  to get a set of remaining Steiner points  $SP_{Roug}$ , and then

calculate  $\eta\epsilon$  using  $SP_{Roug}$ .

- **Rough path calculation:** In Figure 10 (c), given  $T$ ,  $s$ ,  $t$ , and  $SP_{Roug}$ , we use  $SP_{Roug}$  and  $V$ , i.e.,  $G_{Roug}.V$ , to construct a weighted graph  $G_{Roug}$ , apply Dijkstra's algorithm on  $G_{Roug}$  to calculate a  $(1+\eta\epsilon)$ -approximate rough path between  $s$  and  $t$ , and store  $dist_{Roug}(u)$  and  $prev_{Roug}(u)$  for each  $u \in G_{Roug}.V$  as node information. We denote the rough path as  $\Pi_{Roug}(s, t)$ , i.e., the orange dashed line.

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

- **Full edge sequence conversion:** In Figure 10 (d), given  $\Pi_{Roug}(s, t)$  whose corresponding edge sequence is a non-full edge sequence (since the corresponding edge sequence of  $\Pi_{Roug}(s, t)$  in Figure 10 (c) has an edge with a length of 0 at  $(v, v)$ ), we efficiently modify it and obtain a modified rough path whose corresponding edge sequence is converted to a full edge sequence  $S$  progressively. In Figure 10 (d), we denote the modified rough path as  $\Pi_{Ref-1}(s, t)$ , i.e., the orange dashed line, which corresponding edge sequence is converted from a non-full edge sequence to a full edge sequence  $S = (e_1, e_2, e_3, e_4)$  (edges in red).
- **Snell's law path refinement:** In Figure 10 (e), given  $T$ ,  $s$ ,  $t$ , and the edge sequence  $S$  (edges in red) of  $\Pi_{Ref-1}(s, t)$  based on  $T$ , we use Snell's law to efficiently refine  $\Pi_{Ref-1}(s, t)$  using  $S$ . We denote the refined path on  $S$  as  $\Pi_{Ref-2}(s, t)$ , i.e., the green line.

• **Path checking:** In Figure 10 (f), given  $\Pi_{Roug}(s, t)$ ,  $\Pi_{Ref-2}(s, t)$ ,  $\epsilon$ , and  $\eta$ , we guarantee  $|\Pi_{Roug}(s, t)| \leq (1+\eta\epsilon)|\Pi^*(s, t)|$  due to the error bound of Dijkstra's algorithm, so if  $|\Pi_{Ref-2}(s, t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s, t)|$ , we have  $|\Pi_{Ref-2}(s, t)| \leq (1+\epsilon)|\Pi^*(s, t)|$  and return  $\Pi_{Ref-2}(s, t)$  as output. Otherwise, we need the next step.

• **Error guaranteed path refinement:** In Figure 10 (g), given  $T$ ,  $s$ ,  $t$ ,  $SP_{Ref}$ , and the node information, we apply Dijkstra's algorithm on a weighted graph  $G_{Ref}$  constructed by  $SP_{Ref}$  and  $V$ , to efficiently calculate a  $(1+\epsilon)$ -approximate weighted shortest path between  $s$  and  $t$ . We denote the refined path as  $\Pi_{Ref-3}(s, t)$ , i.e., the purple line. We guarantee that  $|\Pi_{Ref-3}(s, t)| \leq (1+\epsilon)|\Pi^*(s, t)|$  (due to the error bound of Dijkstra's algorithm, see Theorem 3.1 of work [28]).

4) **Necessity of using algorithm *Roug* before algorithm *Ref*:** (1) *Reduce the searching region when using Snell's law:* Recall that the main reason why our algorithm is efficient is

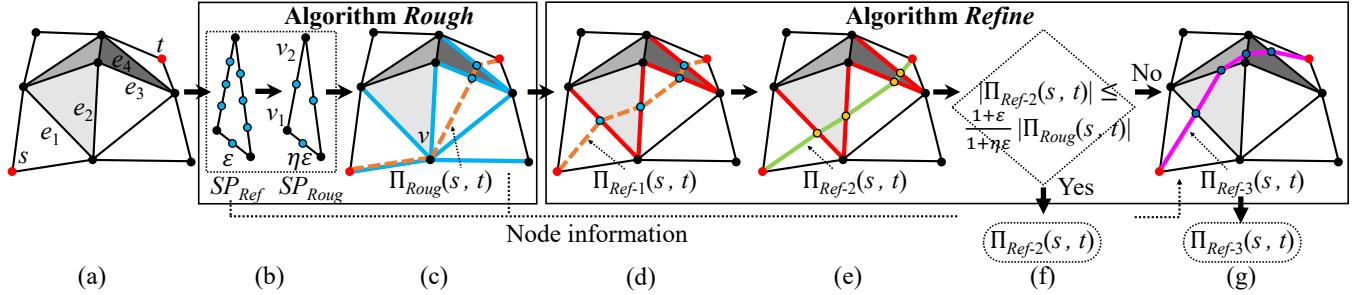


Fig. 10. Framework overview

that we use Snell's law in algorithm *Ref*. Given two vertices  $s$  and  $t$ , we first obtain edge sequence  $S$  passed by the rough path  $\Pi_{Roug}(s, t)$  based on  $T$ , then apply Snell's law on  $S$ . If we do not know  $S$ , we need to try Snell's law on different combinations of edges in  $E$  and select the result path with the minimum length, which is time-consuming. (2) *Provide an error guarantee of the refined path after utilizing Snell's law*: After utilizing Snell's law, we do not know whether the refined path  $\Pi_{Ref-2}(s, t)$  on  $S$  is a  $(1+\epsilon)$ -approximate weighted shortest path, so we need  $\Pi_{Roug}(s, t)$  for assistance. Since we can guarantee that  $|\Pi_{Roug}(s, t)| \leq (1 + \eta\epsilon)|\Pi^*(s, t)|$ , in the path checking step, if  $|\Pi_{Ref-2}(s, t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s, t)|$ , we can guarantee that  $|\Pi_{Ref-2}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ , return  $\Pi_{Ref-2}(s, t)$  as output, and terminate our algorithm.

5) *Efficiency of Snell's law in algorithm Ref*: After algorithm *Roug*, Snell's law can be used on  $S$  to refine the rough path to  $\Pi_{Ref-2}(s, t)$ , which is very close to the optimal weighted shortest path  $\Pi^*(s, t)$ , i.e., the distance between the intersection point of  $\Pi_{Ref-2}(s, t)$  on each edge of  $S$  and the intersection point of  $\Pi^*(s, t)$  on each edge of  $S$  is smaller than a user parameter  $\delta$ , where  $\delta$  is used for controlling the error and proportional depending on  $\epsilon$ , so there is no need to place many Steiner points on edges in  $E$  to calculate the result path as in the best-known algorithm [24], [29].

#### B. Key Idea in Algorithm *Roug*

1) *Efficient Steiner point placement scheme*: It is involved in the  $\eta$  calculation step. The reason why our Steiner point placement scheme is efficient is that we consider the additional geometry information of  $T$ , e.g., the weight of a face, the internal angle of a face, the length of an edge, etc., so that we can place *different* numbers of Steiner points on different edges of  $E$ , and the interval between two adjacent Steiner points on the same edge can be *different*. This allows us to minimize the number of Steiner points per edge (and can also guarantee the error bound). Figure 8 (b) shows an example of our placement of Steiner points, where there are 8, 7, and 7 Steiner points on edge of  $f_i$ .

In contrast, the Steiner point placement scheme used in the best-known work [24], [29], i.e., algorithm *FixSP*, results in a large number of Steiner points per edge, and makes their algorithm very slow. Recall that no matter how  $T$  looks like, algorithm *FixSP* always needs to place  $O(n^2)$  Steiner points

per edge to bound the error. But our Steiner point placement scheme only needs  $O(n \log c_2)$  Steiner points per edge, where  $c_2$  is a constant depending on the error and the geometry information of  $T$ . Figure 9 shows an example of the placement of Steiner points in algorithm *FixSP* (with the same error ratio as of Figure 8 (b)). Our experiment shows that applying Dijkstra's algorithm on the weighted graph constructed using our Steiner point placement scheme just needs 219s ( $\approx 3.6$  min) on a terrain with 50k faces with  $\epsilon = 0.1$ , but algorithm *FixSP* needs 119,000s ( $\approx 1.5$  days) under the same setting.

#### C. Key Idea in Algorithm *Ref*

1) *Novel full edge sequence conversion technique using progressive idea*: It is involved in the full edge sequence conversion step. We provide the reason why we need this step. Recall that we use Snell's law on the edge sequence  $S$  passed by the rough path  $\Pi_{Roug}(s, t)$ , for efficient refinement. In Figure 10 (c), if  $\Pi_{Roug}(s, t)$  passes a vertex  $v$  (the edge sequence  $S$  passed by  $\Pi_{Roug}(s, t)$  has an edge with a length of 0 at  $(v, v)$ , i.e., a non-full edge sequence), we need to add *all* the edges with  $v$  as one endpoint in  $S$  (edges in blue) for error guarantees, in addition to the edges with lengths larger than 0 that passed by the rough path. However, we do not need *all* such edges. Instead, we only need *some* of them. Thus, we need to try Snell's law on different combinations of edge sequences to select the result path with the minimum length, which significantly increases the running time. To solve it, we use the full edge sequence conversion step to modify the rough path efficiently and ensure that all the edges passed by the modified rough path have lengths larger than 0, as shown in Figure 10 (d).

**Illustration:** In Figure 7 (a), given a rough path  $(s, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, t)$  (i.e., the orange line between  $s$  and  $t$ ) whose corresponding edge sequence is a non-full edge sequence, we divide it into a smaller segment  $(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5)$  such that all the edges passed by this segment have length equal to 0, i.e., at  $(\phi_2, \phi_2)$ ,  $(\phi_3, \phi_3)$  and  $(\phi_4, \phi_4)$ . We then add more Steiner points and use this step *progressively* to find a new path segment, i.e., the purple dashed line between  $\phi_1$  and  $\phi_5$ , until the edge sequence (in red) passes by this path segment is a full edge sequence. If the distance of the new path segment is smaller than the distance of the original path segment, we replace the new one with the original one.

In the end, we can obtain a modified rough path (i.e., the purple dashed line between  $s$  and  $t$ ) whose corresponding edge sequence (in red) is a full edge sequence. Since we do not want the rough path to pass the vertex, it seems that in algorithm *Roug*, we can use Dijkstra's algorithm on the weighted graph constructed only using the Steiner points (i.e., not using the Steiner points and  $V$ ). But, this cannot guarantee the rough path to be a  $(1 + \eta\epsilon)$ -approximate weighted shortest path.

**2) Novel effective weight pruning technique:** It is involved in the Snell's law path refinement step. In this step, the basic idea is to use binary search to find the optimal point of the weighted shortest path on each edge in  $S$  by utilizing Snell's law, then connect these points to form the refined path. However, we can efficiently prune out some checking in the binary search and reduce the algorithm's running time by utilizing one useful information on  $T$  called *effective weight*.

**Illustration:** In Figure 5 (a), for the first edge  $e_1$  in  $S$  that is opposite to  $s$ , we select the midpoint  $m_1$  on  $e_1$ , and trace a blue light ray that follows Snell's law from  $s$  to  $m_1$ . This light ray bends at each edge in  $S$ . We check whether  $t$  is on the left or right of this ray and adjust the position of  $m_1$  to the left or right accordingly. This procedure is repeated until the light ray passes the entire  $S$ . Now, we have the purple light ray from  $s$  to  $m_1^1$  that passes the whole  $S$  for the first time, then we can use effective weight to prune out unnecessary checking. In Figure 5 (a) and (b), we regard all the faces except the first face (i.e.,  $f_1$  and  $f_2$ ) in  $F(S)$  as one *effective face* (i.e.,  $\Delta u_{pp_1q_1}$ ), and use the ray in the purple line for calculating effective weight of  $\Delta u_{pp_1q_1}$ . Then, we can calculate the position of effective point  $m_{ef}$  on  $e_1$  in one simple quartic equation using the weight of  $f_0$  and the effective weight of  $\Delta u_{pp_1q_1}$ . In Figure 5 (c), we find the ray starting from  $s$  and passing  $m_{ef}$  (i.e., the dark blue line), which is very close to  $t$ , implies that  $m_{ef}$  is very close to the optimal point. We iterate the midpoint selection step until (1) the light ray hits  $t$  or (2) the distance between new  $m_1$  and previous  $m_1$  is smaller than  $\delta$ . In Figure 5 (d), after processing  $e_1$ , we continue to process other edges in  $S$ , and we have the ray starting from  $\rho_1$  and passing  $m_2^3$  (the green line) or  $m_2^4$  (the yellow line).

**3) Novel error guaranteed path refinement using pruning technique:** It is involved in the error guaranteed path refinement step, with the help of algorithm *Roug*. We provide the reason why we need this step. In the path checking step, if the checking is positive (which occurs 99% of the time theoretically and 100% in our experiments), we do not need the error guaranteed path refinement step. If not (this only happens when the edge sequence  $S$  passed by the rough path differs from the edge sequence passed by the optimal weighted shortest path), we need the error guaranteed path refinement step. We hope that even with this extra step, the total running time of algorithm *Roug-Ref* will not increase a lot.

**Illustration:** (1) In the  $\eta$  calculation step of algorithm *Roug*, recall that we have calculated  $SP_{Ref}$  with error  $\epsilon$  and  $SP_{Roug}$  with error  $\eta\epsilon$ . In Figure 10 (b), we calculate  $\eta$  by first placing Steiner points on each edge in  $E$  with input error  $\epsilon$  to obtain  $SP_{Ref}$ , then iteratively keeping one Steiner

point and removing the next one when encountering a Steiner point while going from the two endpoints of each edge to the center of the edge. This calculation results in  $SP_{Roug}$ , which is used to calculate  $\eta\epsilon$ . As such,  $SP_{Roug} \subseteq SP_{Ref}$ , which means  $G_{Roug}.V \subseteq G_{Ref}.V$ . In Figure 6,  $G_{Roug}.V$  is denoted as three blue nodes and three black nodes,  $G_{Ref}.V$  is denoted as three blue nodes, six green nodes and three black nodes. (2) In the rough path calculation step of algorithm *Roug*, we have  $dist_{Roug}(c) = 5 + 2 = 7$  and  $prev_{Roug}(c) = a$  as node information of  $c$ . (3) In the error guaranteed path refinement step in algorithm *Ref*, for the priority queue of Dijkstra's algorithm on the weighted graph  $G_{Ref}$ , when we follow the path  $s \rightarrow b \rightarrow c$  and are ready to dequeue the edge containing  $b$  (since  $dist_{Roug}(b) = 4.7 < 5 = dist_{Roug}(a)$ ), we find that the distance from  $s$  to  $c$  is  $4.7 + 3 = 7.7 > 7$ . So, the shortest path from  $s$  to  $c$  will not pass  $b$ , and we do not need to insert  $c$  and  $dist_{Roug}(c) = 7.7$  into the queue, which saves the enqueue and dequeue time. It is easy to verify that the case where we do not insert a Steiner point into the queue in the error guaranteed path refinement step in algorithm *Ref* is exactly the same as the case where we perform Dijkstra's algorithm on the weighted graph  $G_{Roug}$  in the rough path calculation step in algorithm *Roug*. Thus, the total running time of the rough path calculation step in algorithm *Roug* and the error guaranteed path refinement step in algorithm *Ref*, is the same as the running time when we perform Dijkstra's algorithm on  $G_{Ref}$  without the node information.

Although our experimental result shows that the error guaranteed path refinement step is not needed in 100% of cases, it does not mean the useless of the efficient technique in this step. When the remaining value  $k$  is larger, more Steiner points are removed,  $\eta$  is larger, so the chance that the checking result is false becomes larger. In this case, the error guaranteed path refinement step is necessary to ensure error guarantee. Our experimental result verifies that the optimal value of  $k$  is 2, as it minimizes the algorithm's total running time. By changing  $k$  from 2 to a value larger than 2, the chance of using this step increases from 0% to 100%.

#### D. Implementation Details of Algorithm *Roug*

We define  $\epsilon'$  to be a different error parameter corresponding to  $\epsilon$ , where  $(2 + \frac{2W}{(1-2\epsilon') \cdot w})\epsilon' = \epsilon$ ,  $W$  and  $w$  are the maximum and minimum weight of all faces in  $F$ , respectively. 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  $V$ . Given a vertex  $v$  in  $V$ , we define  $h_v$  to be the minimum distance from  $v$  to the boundary of one of its incident faces. Given a vertex  $v$  in  $V$ , we define a polygonal cap around  $v$ , denoted as  $C_v$ , to be a *sphere* with center at  $v$ . Let  $r_v = \epsilon' h_v$  be the radius of  $C_v$  with  $0 < \epsilon' < \frac{1}{2}$ . Let  $\theta_v$  be the angle between any two edges of  $T$  that are incident to  $v$ . Let  $h$ ,  $r$  and  $\theta$  be the minimum  $h_v$ ,  $r_v$  and  $\theta_v$  for all  $v \in V$ , respectively. Figure 8 (a) shows an example of the polygonal cap  $C_v$  around  $v$ , with radius  $r_v$ , and the angle  $\theta_v$  of  $v$ .

**Detail:** We introduce how to place Steiner points based on  $\epsilon$ . Let  $\lambda = (1 + \epsilon' \cdot \sin \theta_v)$  if  $\theta_v < \frac{\pi}{2}$ , and  $\lambda = (1 + \epsilon')$  otherwise.

For each vertex  $v$  of face  $f_i$ , let  $e_a$  and  $e_b$  be the edges of  $f_i$  incident to  $v$ . We place Steiner points  $a_1, a_2, \dots, a_{\tau_a-1}$  (resp.  $b_1, b_2, \dots, b_{\tau_b-1}$ ) along  $e_a$  (resp.  $e_b$ ) such that  $|\overline{va_j}| = r_v \lambda^{j-1}$  (resp.  $|\overline{vb_k}| = r_v \lambda^{k-1}$ ) where  $\tau_a = \log_\lambda \frac{|e_a|}{r_v}$  (resp.  $\tau_b = \log_\lambda \frac{|e_b|}{r_v}$ ) for every integer  $2 \leq j \leq \tau_a - 1$  (resp.  $2 \leq k \leq \tau_b - 1$ ). We repeat it on each edge of  $f_i$ . Figure 8 (a) and Figure 8 (b) show an example of Steiner points on  $e_a$  and  $e_b$ , and  $f_i$  based on  $\epsilon$ , respectively. Knowing this, in Figure 10 (b), we use the remaining value  $k$  to remove some Steiner points, and we can derive  $\eta\epsilon$  using the remaining Steiner points  $SP_{Roug}$ . Then, we can use Dijkstra's algorithm on  $G_{Roug}$  (constructed using  $SP_{Roug}$  and  $V$ ) to calculate  $\Pi_{Roug}(s, t)$ .

#### E. Implementation Details of Algorithm Ref

1) **Full edge sequence conversion:** Given a point  $v$ ,  $v$  is on the edge means that it lies in the internal of an edge in  $E$ ,  $v$  is on the vertex means that it lies on the vertex in  $V$ . In Figure 7 (a),  $\phi_1$  is on the edge,  $\phi_2$  is on the vertex.

**Detail and example:** Algorithm 1 shows this step. After we get  $\Pi_{Ref-1}(s, t)$ , we retrieve its edge sequence  $S$  based on  $T$ . The following illustrates it with an example.

---

#### Algorithm 1 EdgSeqConv ( $\Pi_{Roug}(s, t)$ , $\zeta$ )

---

**Input:** the rough path  $\Pi_{Roug}(s, t)$  and  $\zeta$  (a constant and normally set as 10)  
**Output:** the modified rough path  $\Pi_{Ref-1}(s, t)$

- 1:  $v_s \leftarrow \text{NULL}$ ,  $v_e \leftarrow \text{NULL}$ ,  $E' \leftarrow \emptyset$ ,  $\Pi_{Ref-1}(s, t) \leftarrow \Pi_{Roug}(s, t)$
- 2: **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**
- 3:    $v_c \leftarrow v$ ,  $v_n \leftarrow v_c.\text{next}$ ,  $v_p \leftarrow v_c.\text{prev}$
- 4:   **if**  $v_n$  is on the vertex and  $v_p$  is on the edge **then**
- 5:      $v_s \leftarrow v_c$ ,  $E' \leftarrow E' \cup$  edges with  $v_c$  as one endpoint
- 6:   **else if** both  $v_n$  and  $v_p$  are on the edge **then**
- 7:      $E' \leftarrow E' \cup$  edges with  $v_c$  as one endpoint
- 8:   **else if**  $v_n$  is on the edge and  $v_n$  is on the vertex **then**
- 9:      $v_e \leftarrow v_n$ ,  $E' \leftarrow E' \cup$  edges with  $v_c$  as one endpoint
- 10:    add new Steiner points at the midpoints between the vertices and original Steiner points on  $E'$
- 11:     $\Pi'_{Roug}(v_s, v_e) \leftarrow$  path calculated using Dijkstra's algorithm on the weighted graph constructed by new Steiner points and  $V$
- 12:     $\Pi'_{Ref-1}(v_s, v_e) \leftarrow \text{EdgSeqConv}(\Pi'_{Roug}(v_s, v_e), \zeta)$
- 13:   **if**  $|\Pi'_{Ref-1}(v_s, v_e)| < |\Pi_{Roug}(v_s, v_e)|$  **then**
- 14:      $\Pi_{Ref-1}(s, t) \leftarrow \Pi_{Ref-1}(s, v_s) \cup \Pi'_{Ref-1}(v_s, v_e) \cup \Pi_{Ref-1}(v_e, t)$
- 15:   **else if** both  $v_n$  and  $v_n$  are on the edge **then**
- 16:     **for**  $i \leftarrow 1$  to  $\zeta$  **do**
- 17:       add new Steiner points at the midpoints between  $v_c$  and the nearest Steiner points of  $v_c$  on the edges that adjacent to  $v_c$
- 18:        $\Pi_l(v_p, v_n)$  (resp.  $\Pi_r(v_p, v_n)$ )  $\leftarrow$  path passes the set of newly added Steiner points on the left (resp. right) side of the path  $(v_p, v_c, v_n)$
- 19:       **if**  $\Pi_l(v_p, v_n) < \Pi_{Roug}(v_p, v_n)$  **then**
- 20:          $\Pi_{Ref-1}(s, t) \leftarrow \Pi_{Ref-1}(s, v_p) \cup \Pi_l(v_p, v_n) \cup \Pi_{Ref-1}(v_n, t)$
- 21:         **break**
- 22:       **else if**  $\Pi_r(v_p, v_n) < \Pi_{Roug}(v_p, v_n)$  **then**
- 23:          $\Pi_{Ref-1}(s, t) \leftarrow \Pi_{Ref-1}(s, v_p) \cup \Pi_r(v_p, v_n) \cup \Pi_{Ref-1}(v_n, t)$
- 24:         **break**
- 25:      $v_c \leftarrow v_n$ ,  $v_n \leftarrow v_c.\text{next}$ ,  $v_p \leftarrow v_c.\text{prev}$
- 26: **return**  $\Pi_{Ref-1}(s, t)$

---

(1.1) **Successive point:** Line 4-14, see Figure 7 (a) with the orange path as input. (1.1.1) Line 4-5:  $v_c = \phi_2$ ,  $v_n = \phi_3$ ,  $v_p = \phi_1$ , there are at least two successive points in  $\Pi_{Roug}(s, t)$  that is on the vertex, we store  $v_s = \phi_1$ , i.e., the start vertex. (1.1.2) Line 6-7:  $v_c = \phi_3$ ,  $v_n = \phi_4$ ,  $v_p = \phi_2$ . (1.1.3) Line 8-14:  $v_c = \phi_4$ ,  $v_n = \phi_5$ ,  $v_p = \phi_3$ , we have found all the successive points,

we store  $v_n = \phi_5$ , i.e., the end vertex. The blue points are the new Steiner points on  $E'$ . The purple dashed line between  $\phi_1$  and  $\phi_5$  represents  $\Pi'_{Ref-1}(v_s, v_e)$ , and the orange line between  $\phi_1$  and  $\phi_5$  represents  $\Pi_{Roug}(v_s, v_e)$ .

(1.2) **Single point:** Line 15-24, see Figure 7 (b) with the orange path as input.  $v_c = \phi_2$ ,  $v_n = \phi_3$ ,  $v_p = \phi_1$ , then only  $v_c$  is on the vertex. The blue points are new Steiner points. The orange, purple line-dashed, pink dot-dashed lines between  $\phi_1$  and  $\phi_3$  represent  $\Pi_{Roug}(v_p, v_n)$ ,  $\Pi_l(v_p, v_n)$ , and  $\Pi_r(v_p, v_n)$ .

2) **Snell's law path refinement:** We let  $\Pi_{Ref-2}(s, t) = (s, \rho_1, \dots, \rho_l, t)$ , where  $\rho_i$  for  $i \in \{1, \dots, l\}$  is a point on an edge in  $E$ . Given an edge sequence  $S$ , a source  $s$ , and a point  $c_1$  on  $e_1 \in S$ , we can obtain a 3D surface Snell's ray  $\Pi_c = (s, c_1, c_2, \dots, c_g, R_g^c)$  starting from  $s$ , hitting  $c$  and following Snell's law on other edges in  $S$ , where  $1 \leq g \leq l$ , each  $c_i$  for  $i \in \{1, \dots, g\}$  is an intersection point in  $\Pi_c$ , and  $R_g^c$  is the last out-ray at  $e_g \in S$ . Figure 5 (a) shows an example of  $\Pi_m = (s, m_1, R_1^m)$  (i.e., the blue line) that does not pass the whole  $S = (e_1, e_2)$ , and  $\Pi_{m1} = (s, m_1^1, m_2^1, R_2^{m1})$  (i.e., the purple line) that passes the whole  $S$ . If we can find  $\rho_1$  on  $e_1$  such that the 3D surface Snell's ray  $\Pi_\rho = (s, \rho_1, \dots, \rho_l, R_l^\rho)$  which hits  $t$ , then  $\Pi_\rho$  is the result of  $\Pi_{Ref-2}(s, t)$ . Recall that  $\delta$  is a user parameter used for controlling the error, we let  $\delta = \frac{h_{ew}}{6IW}$ , where  $h$  is the minimum height of any face in  $F$ ,  $W$  and  $w$  are the maximum and minimum weights of face in  $F$ , and  $l$  is the number of edges in  $S$ , respectively.

**Detail and example:** Algorithm 2 shows this step, and the following illustrates it with an example.

(2.1) **Binary search initial path finding:** Line 6-20, see Figure 5 (a). (2.1.1) Line 6-12: We calculate  $m_1$ , the blue line  $\Pi_m = (s, m_1, R_1^m)$  does not pass the whole  $S$ , and  $e_2$  is on the left side of  $R_1^m$ , so we set  $[a_1, b_1] = [p_1, m_1]$ . (2.1.2) Line 13-20: We then calculate  $m_1^1$ , the purple line  $\Pi_{m1} = (s, m_1^1, m_2^1, R_2^{m1})$  passes the whole  $S$ , and  $t$  is on the right side of  $R_2^{m1}$ , so we set  $[a_1, b_1] = [m_1^1, m_1]$  and  $[a_2, b_2] = [m_2^1, q_2]$ .

(2.2) **Effective weight pruning:** Line 21-29. The purple line  $\Pi_{m1} = (s, m_1^1, m_2^1, R_2^{m1})$  that passes the whole  $S$  based on  $T$  for the first time, we can use effective weight pruning on  $e_i$ . (2.2.1) Line 22, in Figure 5 (a):  $u$  is the intersection point between the purple line  $R_2^{m1}$  and the left edge  $p_1\bar{t}$  of  $f_2$  that adjacent to  $t$ . (2.2.2) Line 23-24, in Figure 5 (b):  $u_p$  is the projected point of  $u$ , and  $f_{ef} = \Delta u_p p_1 q_1$  is an effective face for  $f_1$  and  $f_2$ . (2.2.3) Line 25, in Figure 5 (b): By using the purple line  $sm_1^1$ , the orange dashed line  $m_1^1 u_p$ ,  $f_0$ ,  $f_{ef}$ ,  $w_0$ , and Snell's law, we calculate  $w_{ef}$ . (2.2.4) Line 26-27, in Figure 5 (b):  $t_p$  is the projected point of  $t$ , we set  $m_{ef}$  to be unknown and use Snell's law in vector form [10], then build a quartic equation using  $w_0$ ,  $w_{ef}$ , the dark blue line  $\overline{sm_{ef}}$ , and the green dashed line  $\overline{m_{ef}t_p}$ . Then, we use root formula [8] to solve  $m_{ef}$ . Note that only  $f_0$  and  $f_{ef}$  are involved, so the equation will have the unknown at the power of four. (2.2.5) Line 28-29, in Figure 5 (c): We compute the dark blue line  $\Pi_{m2} = (s, m_{ef}, m_2^2, R_2^{m2})$ , and  $t$  is on the left side of  $R_2^{m2}$ , so we have  $[a_1, b_1] = [m_1^1, m_{ef}]$  and  $[a_2, b_2] = [m_2^1, m_2^2]$ .

(2.3) **Binary search refined path finding:** Line 2, 6-20, 30-31, see Figure 5 (d). (2.3.1) Line 6-20: We perform binary

---

**Algorithm 2** *SneLawRef* ( $s, t, \delta, S$ )

---

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

- 1:  $\Pi_{\text{Ref-2}}(s, t) \leftarrow \{s\}$ ,  $\text{root} \leftarrow s$
- 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  $[a_i, b_i]$
- 7:     compute the 3D surface Snell's ray with  $\Pi_m = (\text{root}, m_i, m_{i+1}, \dots, m_g, R_g^m)$  with  $g \leq l$
- 8:     **if**  $\Pi_m$  does not pass the whole  $S$ , i.e.,  $g < l$  **then**
- 9:       **if**  $e_{g+1}$  is on the left side of  $R_g^m$  **then**
- 10:          $[a_i, b_i] \leftarrow [a_i, m_i], \dots, [a_g, b_g] \leftarrow [a_g, m_g]$
- 11:       **else if**  $e_{g+1}$  is on the right side of  $R_g^m$  **then**
- 12:          $[a_i, b_i] \leftarrow [m_i, b_i], \dots, [a_g, b_g] \leftarrow [m_g, b_g]$
- 13:       **else if**  $\Pi_m$  passes the whole  $S$ , i.e.,  $g = l$  **then**
- 14:         **if**  $t$  is on  $R_g^m$  **then**
- 15:            $\Pi_{\text{Ref-2}}(s, t) \leftarrow \Pi_{\text{Ref-2}}(s, t) \cup \{m_i, m_{i+1}, \dots, m_g, t\}$
- 16:         **return**  $\Pi_{\text{Ref-2}}(s, t)$
- 17:       **else if**  $t$  is on the left side of  $R_g^m$  **then**
- 18:          $[a_i, b_i] \leftarrow [a_i, m_i], \dots, [a_g, b_g] \leftarrow [a_g, m_g]$
- 19:       **else if**  $t$  is on the right side of  $R_g^m$  **then**
- 20:          $[a_i, b_i] \leftarrow [m_i, b_i], \dots, [a_g, b_g] \leftarrow [m_g, b_g]$
- 21:       **if** have not used effective weight pruning on  $e_i$  **then**
- 22:          $u \leftarrow$  the intersection point between  $R_g^m$  and one of the two edges that are adjacent to  $t$  in the last face  $f_l$  in  $F(S)$
- 23:          $u_p \leftarrow$  projected point of  $u$  on the first face  $f_0$  in  $F(S)$
- 24:          $f_{ef} \leftarrow$  effective face contains all faces in  $F(S) \setminus \{f_0\}$
- 25:          $w_{ef} \leftarrow$  effective weight for  $f_{ef}$ , calculated using  $\overline{sm_i, m_i u_p}$ ,  $f_0, f_{ef}, w_0$  (the weight for  $f_0$ ), and Snell's law
- 26:          $t_p \leftarrow$  the projected point of  $t$  on  $f_0$
- 27:          $m_{ef} \leftarrow$  effective intersection point  $m_{ef}$  on  $e_1$ , calculated using  $w_0, w_{ef}, s, t_p$ , and Snell's law in a quartic equation
- 28:          $m_i \leftarrow m_{ef}$ , compute  $\Pi_m = (\text{root}, m_i, \dots, m_g, R_g^m)$
- 29:         update  $[a_i, b_i], \dots, [a_g, b_g]$  same as in line 14-20
- 30:          $\rho_1 \leftarrow [a_i, b_i]$  midpoint,  $\Pi_{\text{Ref-2}}(s, t) \leftarrow \Pi_{\text{Ref-2}}(s, t) \cup \{\rho_1\}$ ,  $\text{root} \leftarrow \rho_1$
- 31:          $\Pi_{\text{Ref-2}}(s, t) \leftarrow \Pi_{\text{Ref-2}}(s, t) \cup \{t\}$
- 32:       **return**  $\Pi_{\text{Ref-2}}(s, t)$

---

search until  $|a_1 b_1| = |m_1^1 m_{ef}| < \delta$ . (2.3.2) Line 30:  $\rho_1$  is the midpoint of  $[a_1, b_1] = [m_1^1, m_{ef}]$ , and we have the pink dashed line  $\Pi_{\text{Ref-2}}(s, t) = (s, \rho_1)$ , and  $\text{root} = \rho_1$ . (2.3.3) Line 2: We iterate the procedure and have the green line  $\Pi_{m^3} = (\rho_1, m_2^3, R_2^{m^3})$  and the yellow line  $\Pi_{m^4} = (\rho_1, m_2^4, R_2^{m^4})$ . Until we process all the edges in  $S = (e_1, e_2)$ , we get result path  $\Pi_{\text{Ref-2}}(s, t) = (s, \rho_1, \rho_2, t)$ .

3) **Error guaranteed path refinement:** We define  $Q$  to be a priority queue that stores nodes  $u \in G_{\text{Ref}}.V$  that are being preprocessed, as  $\{u, \text{dist}_{\text{Ref}}(u)\}$ . In Figure 6,  $Q$  stores  $\{\{a, 5\}, \{b, 4.7\}\}$ .

**Detail and example:** Algorithm 3 shows this step, and the following illustrates it with an example (see Figure 6).

(3.1) **Distance and previous node initialization:** Line 2-6. For green node  $d$ , we initialize  $\text{dist}_{\text{Ref}}(d) = \infty$  and  $\text{prev}_{\text{Ref}}(d) = \text{NULL}$ ; for blue node  $c$ , we initialize  $\text{dist}_{\text{Ref}}(c) = \text{dist}_{\text{Roug}}(c) = 5 + 2 = 7$  and  $\text{prev}_{\text{Ref}}(c) = \text{prev}_{\text{Roug}}(c) = a$ .

(3.2) **Priority queue looping:** Line 7-15. (3.2.1) Line 8-9: Suppose  $Q$  stores  $\{\{a, 5\}, \{b, 4.7\}\}$ , we dequeue  $b$  with shortest distance value 4.7. (3.2.2) Line 12-15: One adjacent node of  $b$  is  $c$ , with the node information, we know  $\text{dist}_{\text{Ref}}(c) = 7$ , so  $7 < \text{dist}_{\text{Ref}}(b) + \text{wleng}(b, c) = 4.7 + 3 = 7.7$ , there is no need to enqueue  $\{c, \text{dist}_{\text{Ref}}(c) = 7.7\}$  into  $Q$ , which saves the time. But, without the node information, we just know

---

**Algorithm 3** *ErrGuarRef* ( $s, t, \epsilon, \text{NodeInfo}$ )

---

**Input:** source  $s$ , destination  $t$ , error parameter  $\epsilon$ , node information  $\text{dist}_{\text{Roug}}(u)$  and  $\text{prev}_{\text{Roug}}(u)$  for each  $u \in G_{\text{Roug}}.V$   
**Output:** the refined path  $\Pi_{\text{Ref-3}}(s, t)$

- 1: place Steiner points  $SP_{\text{Roug}}$  using  $\epsilon$ , construct a weighted graph  $G_{\text{Ref}}$
- 2: **for** each  $u \in G_{\text{Ref}}.V$  **do**
- 3:   **if**  $u \in G_{\text{Ref}}.V \setminus G_{\text{Roug}}.V$  **then**
- 4:      $\text{dist}_{\text{Ref}}(u) \leftarrow \infty$ ,  $\text{prev}_{\text{Ref}}(u) \leftarrow \text{NULL}$
- 5:   **else if**  $u \in G_{\text{Roug}}.V$  **then**
- 6:      $\text{dist}_{\text{Ref}}(u) \leftarrow \text{dist}_{\text{Roug}}(u)$ ,  $\text{prev}_{\text{Ref}}(u) \leftarrow \text{prev}_{\text{Roug}}(u)$
- 7: **while**  $Q$  is not empty **do**
- 8:   dequeue  $Q$  with smallest distance value
- 9:    $v \leftarrow$  dequeued node,  $\text{dist}_{\text{Ref}}(v) \leftarrow$  dequeued shortest distance value
- 10:   **if**  $t = v$  **then**
- 11:     **break**
- 12:   **for** each adjacent vertex  $v'$  of  $v$ , such that  $\overrightarrow{vv'} \in G_{\text{Ref}}.E$  **do**
- 13:     **if**  $\text{dist}_{\text{Ref}}(v') > \text{dist}_{\text{Ref}}(v) + w_{vv'} \cdot d(v, v')$  **then**
- 14:        $\text{dist}_{\text{Ref}}(v') \leftarrow \text{dist}_{\text{Ref}}(v) + w_{vv'} \cdot d(v, v')$ ,  $\text{prev}_{\text{Ref}}(v') \leftarrow v$
- 15:       enqueue  $\{v', \text{dist}_{\text{Ref}}(v')\}$  into  $Q$
- 16:    $u \leftarrow \text{prev}_{\text{Ref}}(t)$ ,  $\Pi_{\text{Ref-3}}(s, t) \leftarrow \{t\}$
- 17: **while**  $u \neq s$  **do**
- 18:    $\Pi_{\text{Ref-3}}(s, t) \leftarrow \Pi_{\text{Ref-3}}(s, t) \cup \{u\}$ ,  $u \leftarrow \text{prev}_{\text{Ref}}(u)$
- 19:  $\Pi_{\text{Ref-3}}(s, t) \leftarrow \{s\}$ , reverse  $\Pi_{\text{Ref-3}}(s, t)$
- 20: **return**  $\Pi_{\text{Ref-3}}(s, t)$

---

$\text{dist}_{\text{Ref}}(c) = \infty$ , and we need to enqueue  $\{c, \text{dist}_{\text{Ref}}(c) = 7.7\}$  into  $Q$ , which increases the time.

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

#### F. Theoretical Analysis

The running time, memory usage, and error of algorithm *Roug-Ref* are in Theorem 1.

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

**Proof Sketch:** The running time contains (1)  $O(n \log n)$  for *Roug* due to Dijkstra's algorithm on  $G_{\text{Roug}}$  with  $n$  nodes, (2)  $O(n \log n)$  for the full edge sequence conversion step in *Ref* due to Dijkstra's algorithm, (3)  $O(l)$  for the Snell's law path refinement step in *Ref* due to  $l$  edges in  $S$  and  $O(1)$  time in finding the optimal intersection point on each edge, and (4)  $O(n \log n)$  for the error guaranteed path refinement step in *Ref* due to Dijkstra's algorithm on  $G_{\text{Ref}}$  with  $n$  nodes and the node information. The memory usage contains  $O(n)$  for *Roug* due to Dijkstra's algorithm on  $G_{\text{Roug}}$  with  $n$  nodes, (2)  $O(n)$  for the full edge sequence conversion step in *Ref* due to Dijkstra's algorithm, (3)  $O(l)$  for the Snell's law path refinement step in *Ref* due to  $l$  edges in  $S$ , and (4)  $O(n)$  for the error guaranteed path refinement step in *Ref* due to Dijkstra's algorithm on  $G_{\text{Ref}}$  with  $n$  nodes.

For the error bound, in the rough path calculation step of *Roug*, we guarantee  $|\Pi_{\text{Roug}}(s, t)| \leq (1 + \eta\epsilon)|\Pi^*(s, t)|$  (due to the error bound of Dijkstra's algorithm, see Theorem 3.1 of work [28]). So in the path checking step of *Ref*, (1) if  $|\Pi_{\text{Ref-2}}(s, t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{\text{Roug}}(s, t)|$ , we guarantee  $|\Pi_{\text{Ref-2}}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ ; (2) if not, we have the error guaranteed path refinement step, which guarantees  $|\Pi_{\text{Ref-3}}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$  (due to the error bound of

TABLE I  
ALGORITHM COMPARISONS,  $A = \text{NoPrunDijk}$ ,  $B = \text{FixSP}$ ,  $C = \text{NoEdgSeqConv}$ ,  $D = \text{NoEffWeig}$

Algorithm	Time	Size	Error
<i>ConWave</i> [34]	$O(n^8 \log(\frac{LNWL}{w\epsilon}))$	Gigantic	$O(n^4)$ $(1 + \epsilon)$
<i>FixSP</i> [24], [29]	$O(n^3 \log n)$	Gigantic	$O(n^3)$ $(1 + \epsilon)$
<i>LogSP</i>	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	Large	$O(n \log \frac{LN}{\epsilon})$ $(1 + \epsilon)$
<i>Roug-Ref(A, B, C, D)</i>	$O(n^3 \log n + n^{1/2} \log(\frac{LNWL}{w\epsilon}))$	Large	$O(n^3 + nl)$ $(1 + \epsilon)$
<i>Roug-Ref(A, •, •, •)</i>	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$	Large	$O(n \log \frac{LN}{\epsilon} + l)$ $(1 + \epsilon)$
<i>Roug-Ref(•, B, •, •)</i>	$O(n^2 \log n + l)$	Large	$O(n^2 + l)$ $(1 + \epsilon)$
<i>Roug-Ref(•, •, C, •)</i>	$O(n \log n + nl)$	Medium	$O(nl)$ $(1 + \epsilon)$
<i>Roug-Ref(•, •, •, D)</i>	$O(n \log n + l^2 \log(\frac{LNWL}{w\epsilon}))$	Medium	$O(n + l)$ $(1 + \epsilon)$
<i>Roug-Ref (ours)</i>	$O(n \log n + l)$	Small	$O(n + l)$ $(1 + \epsilon)$

Dijkstra's algorithm, see Theorem 3.1 of work [28]). The detailed proof appears in the appendix. ■

### G. Baseline

Algorithm *FixSP* [24], [29] in the *Steiner point* approach is regarded as the best-known baseline algorithm. The Dijkstra's algorithm using our efficient Steiner point placement scheme (without the rough-refine idea), called algorithm *Logarithmic Steiner Point placement scheme (LogSP)*, is included as a baseline. Furthermore, in our algorithm *Roug-Ref*, (1) we do not use the node information for pruning in the error guaranteed path refinement step of algorithm *Ref*, (2) we use *FixSP* to substitute *LogSP* as Steiner points placement scheme in algorithm *Roug* and the error guaranteed path refinement in *Ref*, (3) we remove the full edge sequence conversion step in algorithm *Ref*, and (4) we remove the effective weight pruning out sub-step in the Snell's law path refinement step of algorithm *Ref* (after removal, it becomes a naive method [34]), for **ablation study**. We use (1) *Roug-Ref(A, •, •, •)*, (2) *Roug-Ref(•, B, •, •)*, (3) *Roug-Ref(•, •, C, •)*, and (4) *Roug-Ref(•, •, •, D)*, to denote these variations, and we also add (5) *Roug-Ref(A, B, C, D)*, (6) *Roug-Ref(A, B, C, •)*, (7) *Roug-Ref(A, B, •, D)*, (8) *Roug-Ref(A, •, C, D)*, (9) *Roug-Ref(•, B, C, D)*, (10) *Roug-Ref(A, B, •, •)*, (11) *Roug-Ref(A, •, C, •)*, (12) *Roug-Ref(A, •, •, D)*, (13) *Roug-Ref(•, B, C, •)*, (14) *Roug-Ref(•, B, •, D)*, (15) and *Roug-Ref(•, •, C, D)*, for completeness, where  $\{A = \text{NoPrunDijk}, B = \text{FixSP}, C = \text{NoEdgSeqConv}, D = \text{NoEffWeig}\}$ , and  $\bullet$  means keeping the same component as in algorithm *Roug-Ref*.

In general, we have 18 algorithms. We do not use the *ConWave* approach as the baseline, due to its large running time. There is also no implementation of this approach so far [28]. We do not use the *heuristics* approach as the baseline, because it cannot guarantee on the quality of the result path returned with a given time limit and a given maximum memory. We do not use *unweighted shortest path* algorithms [12], [16] as the baseline since they cannot be adapted to the weighted case. We do not use *oracle-based* algorithms [36], [37] as the baseline since we focus on *on-the-fly* algorithm in this paper, and we have included the *on-the-fly* algorithm in work [36], [37], i.e., algorithm *FixSP* [24], as the baseline.

**Comparisons:** We compare 9 algorithms (8 baselines and the *ConWave* approach) in Table I. The comparisons of all algorithms appear in the appendix. *Roug-Ref* is the best algorithm since it has the smallest running time and memory

TABLE II  
DATASETS

Name	F
<i>BearHead (BH)</i> [4], [36], [37]	280k
<i>EaglePeak (EP)</i> [4], [36], [37]	300k
<i>Seabed (SB)</i> [13]	2k
<i>CyberPunk (CP)</i> [3]	2k
<i>PathAdvisor (PA)</i> [41]	1k
<i>BH small-version (BH-small)</i>	3k
<i>EP small-version (EP-small)</i>	3k
Multi-resolution of <i>EP</i>	1M, 2M, 3M, 4M, 5M
Multi-resolution of <i>EP-small</i>	10k, 20k, 30k, 40k, 50k

usage. The proofs of theoretical analysis for *FixSP*, *LogSP*, and the first 4 variations of *Roug-Ref* appear in the appendix, the proofs for other variations of *Roug-Ref* are similar to them.

## V. EMPIRICAL STUDIES

### A. Experimental Setup

We conducted our 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 works [24], [25], [33], [36], [37].

1) **Datasets:** Following existing studies on terrain data [19], [33], [35], we conducted our experiment on 17 real terrain datasets in Table II. For *SB* and *CP* dataset, we first obtained the height / satellite map, then used Blender [1] to generate the 3D terrain model. For *BH-small* and *EP-small* datasets, we generate them using *BH* and *EP* following the procedure in [33], [36], [37] (which creates a terrain with different resolutions and the procedure appears in the appendix), since algorithm *FixSP* [24], [29] is not feasible on any of the datasets with more than 250k faces due to its expensive running time. Following the work [21], we set the weight of a triangle in terrain datasets to be the slope of that face. In order to test the scalability of our algorithm, we also generate multi-resolution of *EP* (with 1M, 2M, 3M, 4M, 5M faces) and *EP-small* (with 10k, 20k, 30k, 40k, 50k faces) datasets.

2) **Algorithms:** Our algorithm *Roug-Ref*, and the baseline algorithms, i.e., *FixSP* [24], [29], *LogSP*, and 15 variations of *Roug-Ref* are studied. Since *FixSP* is not feasible on large datasets due to its expensive running time, we (1) compared all algorithms on datasets with less than 250k faces, and (2) compared algorithms not involving *FixSP* (i.e., both *LogSP* and variations of *Roug-Ref* not involving *FixSP*) on datasets with more than 250k faces.

3) **Query Generation:** We randomly chose pairs of vertices in  $V$ , as source and destination, and the average, minimum, and maximum results of 100 queries are reported.

4) **Factors & Measurements:** We studied three factors, namely (1)  $\epsilon$ , (2) removing value (i.e., in the  $\eta$  calculation step of algorithm *Roug*, the number of removed Steiner points whenever we encounter a new Steiner point), and (3) dataset size (i.e., the number of faces in a terrain surface). In addition, we used four measurements to evaluate the algorithm performance, namely (1) *preprocessing time* (i.e., the time for constructing the weighted graph using Steiner points), (2a) *query time for the first algorithm* (i.e., the time in *Roug*), (2b)

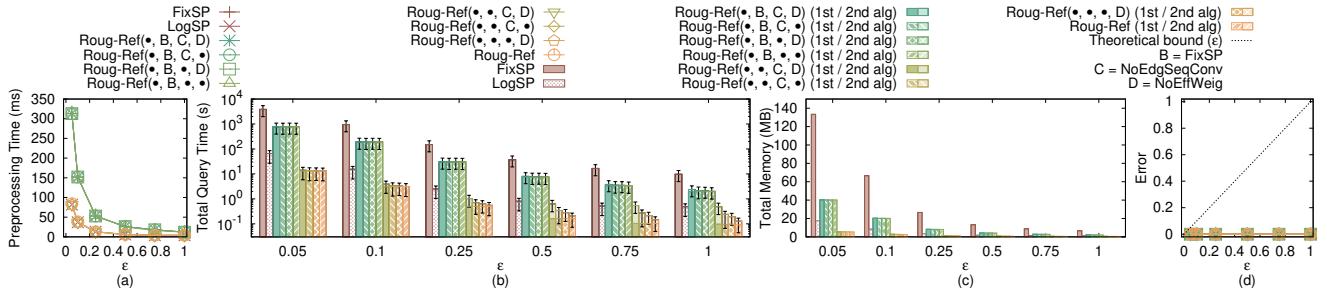


Fig. 11. Effect of  $\epsilon$  on *BH-small* dataset

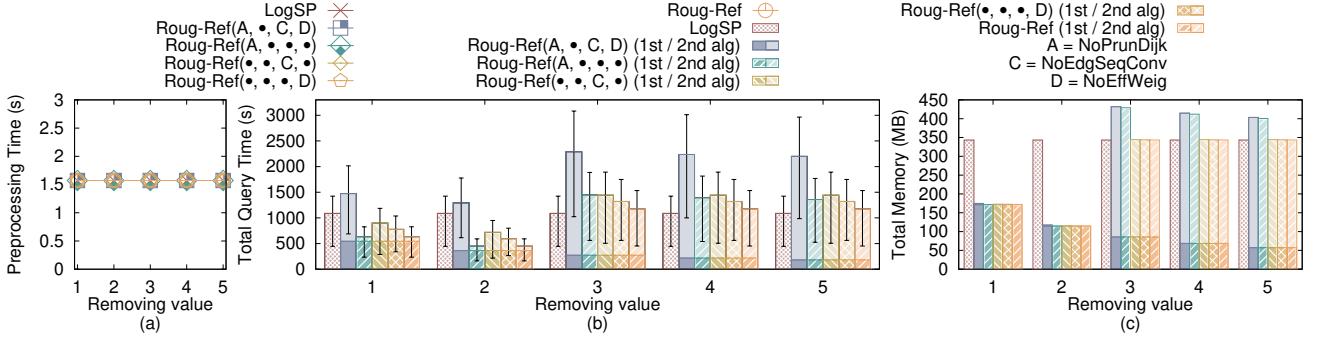


Fig. 12. Effect of removing value on *EP* dataset

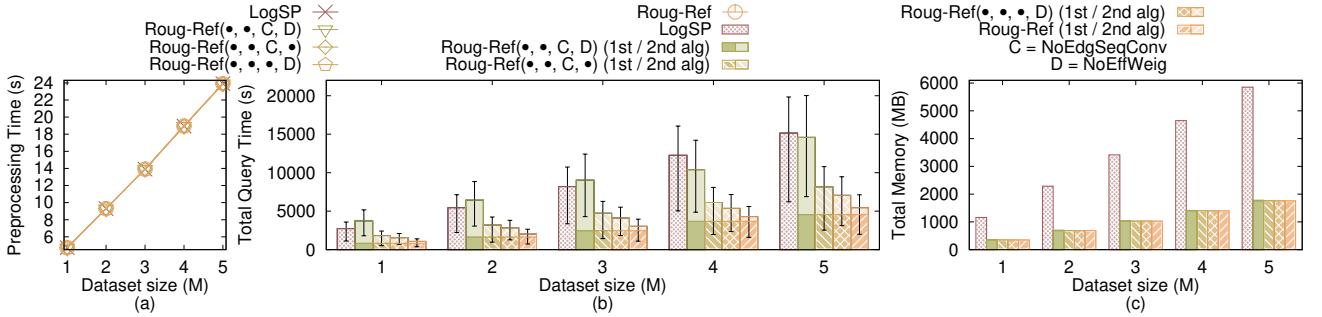


Fig. 13. Effect of dataset size on multi-resolution of *EP* datasets

query time for the second algorithm (i.e., the time in *Ref*), (2c) total query time (i.e., the time in *Roug-Ref*), (3a) memory usage for the first algorithm (i.e., the space consumption in *Roug*), (3b) memory usage for the second algorithm (i.e., the space consumption in *Ref*), (3c) total memory usage (i.e., the space consumption in *Roug-Ref*), and (4) distance error (i.e., the error of the distance returned by the algorithm compared with the exact weighted shortest path).

## B. Experimental Results

**1) Overview:** Figure 11, Figure 12, Figure 13 show the V2V path query result on the *BH-small* dataset, *EP* dataset, and multi-resolution of *EP* datasets when varying  $\epsilon$ , removing value, and dataset size, respectively. For the (1) total query time, (2) query time for algorithm *Roug* and (3) query time for algorithm *Ref*, the vertical bar means the minimum and maximum (1) total query time, (2) query time for algorithm *Roug* and (3) query time for algorithm *Ref*. The results on other combinations of dataset and the variation of  $\epsilon$ , removing value, dataset size, the separation of two steps in query time and memory usage calculated using all algorithms, and A2A path query appear in the appendix. Since no algorithm can

solve the weighted region problem exactly so far, we use *Roug-Ref*(*NoPrunDijk*, *FixSP*, *NoEdgSeqConv*, *NoEffWeig*) and set  $\epsilon = 0.05$  and the removing value to be 2 to simulate the exact weighted shortest path on datasets with less than 250k faces for measuring distance error. Since *FixSP* is not feasible on datasets with more than 250k faces, the distance error is omitted on these datasets.

**2) Effect of  $\epsilon$  for V2V path query:** We tested 6 values of  $\epsilon$  from  $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$  by setting removing value to be 2, and the results on *BH-small* dataset calculated using default algorithms, i.e., *FixSP*, *LogSP*, and variations of *Roug-Ref* without *NoPrunDijk*, are in Figure 11. When varying  $\epsilon$ , we already selected the optimal removing value ( $= 2$ ), our experimental result shows that there is no need to use the error guaranteed path refinement step in algorithm *Ref*, and all the measurements for 15 variations of *Roug-Ref* with or without *NoPrunDijk* are same, so we compared variations of *Roug-Ref* without *NoPrunDijk* for clarity (this also applies on varying the dataset size). *Roug-Ref* performs better than all the remaining algorithms in terms of query time and memory usage, and it is clearer to observe the superior

performance of *Roug-Ref* when  $\epsilon = 1$ . The overlapping lines of the preprocessing time mean the result is the same, since the variation of algorithms will not affect the preprocessing time. The error of all algorithms is close to 0%.

**3) Effect of removing value for V2V path query:** We tested 5 values of removing value from  $\{1, 2, 3, 4, 5\}$  on datasets with less than 250k faces by setting  $\epsilon$  to be 0.1, on datasets with more than 250k faces by setting  $\epsilon$  to be 0.25, and the results on *EP* dataset calculated using *LogSP*, and some important variations of *Roug-Ref* without *FixSP* (i.e., with only one component change), are in Figure 12 (the result calculated using all the variations of *Roug-Ref* appear in the appendix). When varying the removing value, it may happen that the error guaranteed path refinement step in algorithm *Ref* is used, the variations of *Roug-Ref* with or without *NoPrunDijk* will have different effects on the measurements, so we compared variations of *Roug-Ref* involves both with and without *NoPrunDijk*) for completeness. By setting the removing value to be 2, the total running time and total memory usage of the algorithms using *Roug-Ref* framework are the smallest. This is because when the removing value is larger, *Roug* can run faster, but it has a higher chance that *Ref* needs to perform the error guaranteed path refinement step. Thus, we select the optimal removing value, i.e., 2. Although the removing value is larger than 2, the total running time of *Roug-Ref* is slightly larger than *LogSP*, which shows the usefulness of the node information for pruning out in Dijkstra's algorithm in the error guaranteed path refinement step of *Ref*.

**4) Effect of dataset size (scalability test) for V2V path query:** We tested 5 values of dataset size from  $\{11M, 2M, 3M, 4M, 5M\}$  on multi-resolution of *EP* datasets by setting  $\epsilon$  to be 0.25 and removing value to be 2, and the results calculated using default algorithms, i.e., *FixSP*, *LogSP*, and variations of *Roug-Ref* without *NoPrunDijk* and *FixSP*, are in Figure 13. *Roug-Ref* can still beat other algorithms in terms of query time and memory usage. We also tested 5 values of dataset size from  $\{10k, 20k, 30k, 40k, 50k\}$  on multi-resolution of *EP-small* datasets by setting  $\epsilon$  to be 0.1 and removing value to be 2. The figure appears in the appendix.

**5) Arbitrary point-to-arbitrary point (A2A) path query:** We tested the A2A path query by varying  $\epsilon$  from  $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$  and setting removing value to be 2 on *EP-small* and *EP* datasets. The result appears in the appendix. *Roug-Ref* still performs better than all the remaining algorithms in terms of query time and memory usage.

**6) User study:** We conducted a user study on a campus map weighted shortest path finding tool, which allows users to find the shortest path between any two rooms in a university campus, namely *Path Advisor* [41]. The floor of the building is represented in a terrain surface. It is expected that the path should not be too close to the obstacle (e.g., the distance between the path to the obstacle should be at least 0.2 meter). Based on this, when a face on the floor is closer to the boundary of aisle in a building (resp. the aisle center), the face is assigned with a larger (resp. smaller) weight. We adopted our 18 algorithms to their tool by using *PA* dataset [41]. We

chose two places in *Path Advisor* as source and destination, respectively, and repeated it for 100 times to calculate the path (with  $\epsilon = 0.5$ ). Figure 2 shows an example of different paths in *Path Advisor*. The blue and purple dashed paths are the weighted shortest path (with distance 105.8m) and the unweighted shortest path (with distance 98.4m), respectively. We presented Figure 2 and the path distance result to 30 users (i.e., university students), and 96.7% of users think the blue path is the most realistic one since it is not close to the obstacle and it does not have sudden direction changes. In addition, the average query time for the state-of-the-art algorithm *FixSP* and our algorithm *Roug-Ref* are 16.62s and 0.1s, respectively. The result on other measurements appears in the appendix.

**7) Motivation study:** We also conducted a motivation study on the placement of undersea optical fiber cable on the seabed as mentioned in Section I-A by using *SB* dataset [13]. For a face with a deeper sea level, the hydraulic pressure is higher, the cable's lifespan is reduced, and it is more expensive to repair and maintain the cable, so the face will have a larger weight. The average life expectancy of the cable is 25 years [32], 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) [32]. We randomly selected two points as source and destination, respectively, and repeated it 100 times to calculate the path (with  $\epsilon = 0.5$ ). Figure 3 shows an example of different paths on seabed. The blue and purple dashed paths are the weighted shortest path (with a distance 457.9km) and the unweighted shortest path (with a distance 438.3km), respectively. Given the cost of undersea cable, i.e., USD \$25,000/km [26], when constructing a cable that will be used for 100 years [32], the total estimated cost for the green and red paths are USD \$45.8M ( $= \frac{100\text{years}}{25\text{years}} \times 457.9\text{km} \times \$25,000/\text{km}$ ) 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 state-of-the-art algorithm *FixSP* and our algorithm *Roug-Ref* are 22.50s and 0.2s, respectively. The result on other measurements appears in the appendix.

**8) Summary:** Our algorithm *Roug-Ref* consistently outperforms the state-of-the-art algorithm, i.e., *FixSP*, in terms of all measurements. Specifically, our algorithm runs up to 1630 times faster than the state-of-the-art algorithm. When the dataset size is 50k with  $\epsilon = 0.1$ , our algorithm's total query time is 73s ( $\approx 1.2$  min), and total memory usage is 43MB, but the state-of-the-art algorithm's total query time is 119,000s ( $\approx 1.5$  days), and total memory usage is 2.9GB. The case study also shows that *Roug-Ref* is the best algorithm since it is the fast algorithm that supports real-time responses.

## VI. CONCLUSION

In our paper, we propose a two-step approximation algorithm for calculating the weighted shortest path in the 3D weighted region problem using algorithm *Roug-Ref*. Our algorithm can bound the error ratio, and the experimental results show that it runs up to 1630 times faster than the state-of-the-art algorithm. Future work can be proposing a new pruning step to reduce the algorithm's running time further.

## REFERENCES

- [1] “Blender,” 2023. [Online]. Available: <https://www.blender.org>
- [2] “Cyberpunk 2077,” 2023. [Online]. Available: <https://www.cyberpunk.net>
- [3] “Cyberpunk 2077 map,” 2023. [Online]. Available: <https://www.videogamecartography.com/2019/08/cyberpunk-2077-map.html>
- [4] “Data geocomm,” 2023. [Online]. Available: <http://data.geocomm.com/>
- [5] “Fermat’s principle,” 2023. [Online]. Available: [https://en.wikipedia.org/wiki/Fermat's\\_principle](https://en.wikipedia.org/wiki/Fermat's_principle)
- [6] “Google earth,” 2023. [Online]. Available: <https://earth.google.com/web>
- [7] “Metaverse,” 2023. [Online]. Available: <https://about.facebook.com/meta>
- [8] “Quartic function,” 2023. [Online]. Available: [https://en.wikipedia.org/wiki/Quartic\\_function](https://en.wikipedia.org/wiki/Quartic_function)
- [9] “Snell’s law,” 2023. [Online]. Available: [https://en.wikipedia.org/wiki/Snell's\\_law](https://en.wikipedia.org/wiki/Snell's_law)
- [10] “Snell’s law in vector form,” 2023. [Online]. Available: <https://physics.stackexchange.com/questions/435512/snells-law-in-vector-form>
- [11] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack, “An  $\varepsilon$ -approximation algorithm for weighted shortest paths on polyhedral surfaces,” in *Scandinavian Workshop on Algorithm Theory*. Springer, 1998, pp. 11–22.
- [12] G. Altintas, “Finding shortest path on a terrain surface by using finite element method,” *arXiv preprint arXiv:2201.06957*, 2022.
- [13] C. Amante and B. W. Eakins, “Etopo1 arc-minute global relief model: procedures, data sources and analysis,” 2009.
- [14] H. Antikainen, “Comparison of different strategies for determining raster-based least-cost paths with a minimum amount of distortion,” *Transactions in GIS*, vol. 17, no. 1, pp. 96–108, 2013.
- [15] 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.
- [16] J. Chen and Y. Han, “Shortest paths on a polyhedron,” in *SOCG*, New York, NY, USA, 1990, p. 360–369.
- [17] 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.
- [18] K. Deng, H. T. Shen, K. Xu, and X. Lin, “Surface k-nn query processing,” in *22nd International Conference on Data Engineering (ICDE’06)*. IEEE, 2006, pp. 78–78.
- [19] K. Deng and X. Zhou, “Expansion-based algorithms for finding single pair shortest path on surface,” in *International Workshop on Web and Wireless Geographical Information Systems*. Springer, 2004, pp. 151–166.
- [20] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
- [21] A. Gheibi, A. Maheshwari, J.-R. Sack, and C. Scheffer, “Path refinement in weighted regions,” *Algorithmica*, vol. 80, no. 12, pp. 3766–3802, 2018.
- [22] C. Goerzen, Z. Kong, and B. Mettler, “A survey of motion planning algorithms from the perspective of autonomous uav guidance,” *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1, pp. 65–100, 2010.
- [23] N. Jaklin, M. Tibboel, and R. Geraerts, “Computing high-quality paths in weighted regions,” in *Proceedings of the Seventh International Conference on Motion in Games*, 2014, pp. 77–86.
- [24] M. Kaul, R. C.-W. Wong, and C. S. Jensen, “New lower and upper bounds for shortest distance queries on terrains,” *Proceedings of the VLDB Endowment*, vol. 9, no. 3, pp. 168–179, 2015.
- [25] M. Kaul, R. C.-W. Wong, B. Yang, and C. S. Jensen, “Finding shortest paths on terrains by killing two birds with one stone,” *Proceedings of the VLDB Endowment*, vol. 7, no. 1, pp. 73–84, 2013.
- [26] J. Kim, “Submarine cables: the invisible fiber link enabling the internet,” 2022. [Online]. Available: <https://dgtlinfra.com/submarine-cables-fiber-link-internet/>
- [27] M. R. Kindl and N. C. Rowe, “Evaluating simulated annealing for the weighted-region path-planning problem,” in *2012 26th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2012, pp. 926–931.
- [28] M. Lanthier, “Shortest path problems on polyhedral surfaces.” Ph.D. dissertation, Carleton University, 2000.
- [29] M. Lanthier, A. Maheshwari, and J.-R. Sack, “Approximating shortest paths on weighted polyhedral surfaces,” *Algorithmica*, vol. 30, no. 4, pp. 527–562, 2001.
- [30] L.-H. Lee, T. Braud, P. Zhou, L. Wang, D. Xu, Z. Lin, A. Kumar, C. Bermejo, and P. Hui, “All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda,” *arXiv preprint arXiv:2110.05352*, 2021.
- [31] L.-H. Lee, Z. Lin, R. Hu, Z. Gong, A. Kumar, T. Li, S. Li, and P. Hui, “When creators meet the metaverse: A survey on computational arts,” *arXiv preprint arXiv:2111.13486*, 2021.
- [32] J.-F. Libert and G. Waterworth, “13 - cable technology,” in *Undersea Fiber Communication Systems (Second Edition)*. Academic Press, 2016, pp. 465–508.
- [33] L. Liu and R. C.-W. Wong, “Finding shortest path on land surface,” in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011, pp. 433–444.
- [34] 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.
- [35] C. Shahabi, L.-A. Tang, and S. Xing, “Indexing land surface for efficient knn query,” *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1020–1031, 2008.
- [36] V. J. Wei, R. C.-W. Wong, C. Long, and D. M. Mount, “Distance oracle on terrain surface,” in *SIGMOD/PODS’17*, New York, NY, USA, 2017, p. 1211–1226.
- [37] V. J. Wei, R. C.-W. Wong, C. Long, D. M. Mount, and H. Samet, “Proximity queries on terrain surface,” *ACM Transactions on Database Systems (TODS)*, 2022.
- [38] E. K. Xidas, “On designing near-optimum paths on weighted regions for an intelligent vehicle,” *International Journal of Intelligent Transportation Systems Research*, vol. 17, no. 2, pp. 89–101, 2019.
- [39] S. Xing, C. Shahabi, and B. Pan, “Continuous monitoring of nearest neighbors on land surface,” *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1114–1125, 2009.
- [40] D. Yan, Z. Zhao, and W. Ng, “Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces,” in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 942–951.
- [41] Y. Yan and R. C.-W. Wong, “Path advisor: a multi-functional campus map tool for shortest path,” *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2683–2686, 2021.
- [42] X. Zheng, S. Koenig, D. Kempe, and S. Jain, “Multirobot forest coverage for weighted and unweighted terrain,” *IEEE Transactions on Robotics*, vol. 26, no. 6, pp. 1018–1031, 2010.

## APPENDIX A SUMMARY OF FREQUENT USED NOTATIONS

Table III shows a summary of frequent used notations.

## APPENDIX B COMPARISON OF ALL ALGORITHMS

Table IV shows a comparison of all algorithms.

## APPENDIX C EXAMPLE ON THE GOOD PERFORMANCE OF THE EFFECTIVE WEIGHT PRUNING SUB-STEP IN THE SNELL’S LAW PATH REFINEMENT 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 point  $r_1$  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  $[50, 100]$ ,  $[75, 100]$ ,  $[75, 87.5]$ ,  $[81.25, 87.5]$ ,  $[84.375, 87.5]$ ,  $[85.9375, 87.5]$ ,  $[86.71875, 87.5]$ ,  $[87.109375, 87.5] \dots$ . But, with the effective weight pruning sub-step in the Snell’s law path refinement step of algorithm *Ref*, assume that we still

TABLE III  
SUMMARY OF FREQUENT USED NOTATIONS

Notation	Meaning
$T$	The terrain surface
$V/E/F$	The set of vertices / edges / faces of $T$
$n$	The number of vertices of $T$
$N$	The smallest integer value which is larger than or equal to the coordinate value of any vertex
$W/w$	The maximum / minimum weights of $T$
$L$	The length of the longest edge in $T$
$h$	The minimum height of any face in $F$
$\epsilon$	The error parameter
$\eta$	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
$ \Pi(s, t) $	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 <i>Roug</i>
$\Pi_{Ref-1}(s, t)$	The modified rough path calculated using the full edge sequence conversion step of algorithm <i>Ref</i>
$\Pi_{Ref-2}(s, t)$	The refined path calculated using the Snell's law path refinement step of algorithm <i>Ref</i>
$\Pi_{Ref-3}(s, t)$	The refined path calculated using the error guaranteed path refinement step of algorithm <i>Ref</i>
$\rho_i$	The intersection point of $\Pi_{Ref-2}(s, t)$ and an edge in $E$
$\xi$	The iteration counts of single endpoint cases in the full edge sequence conversion step of algorithm <i>Ref</i>
$S$	The edge sequence that $\Pi_{Ref-1}(s, t)$ connects from $s$ to $t$ in order based on $T$
$l$	The number of edges in $S$
$\Pi_c$	A 3D surface Snell's ray
$SP_{Roug} / SP_{Ref}$	The set of Steiner points on each edge of $E$ used in algorithm <i>Roug</i> / <i>Ref</i>
$G_{Roug} / G_{Ref}$	The connected weighted graph used in algorithm <i>Roug</i> / <i>Ref</i>
$G_{Roug}.V / G_{Ref}.V$	The set of vertices in the connected weighted graph used in algorithm <i>Roug</i> / <i>Ref</i>
$G_{Roug}.E / G_{Ref}.E$	The set of edges in the connected weighted graph used in algorithm <i>Roug</i> / <i>Ref</i>

need to use several iterations of the original binary search to let  $\Pi_m$  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  $\Pi_m$  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 A2A PATH QUERY

Apart from the V2V path query that we discussed in the main body of this paper, we also present a method to answer the *arbitrary point-to-arbitrary point (A2A) path query* in the weighted region problem based on our method. We give a definition first. Given a vertex  $v$ , a face  $f$  with three vertices

$v_1, v_2, v_3$  and weight  $w$ , we define  $v$  belongs to  $f$  if the area of  $f = \Delta v_1 v_2 v_3$  is equal to the sum of the area of  $\Delta v v_1 v_2$ ,  $\Delta v v_1 v_3$  and  $\Delta v v_2 v_3$ . With the definition, we are ready to introduce the adapted method to answer the A2A path query in the weighted region. This adapted method is similar to the one presented in Section IV, the only difference is that if  $s$  or  $t$  is not in  $V$ , we can simply make them as vertices by adding new triangles between them and the three vertices of the face  $f$  that  $s$  or  $t$  belongs in  $T$ , and the newly added triangle face's weight is the same as the weight of  $f$ . We need to remove  $f$  from  $F$ , and add the newly added triangle faces into  $F$ . We also add  $s$  or  $t$  in  $V$ , and add the edges of the newly added triangle faces into  $E$ . In the worst case, both  $s$  and  $t$  are not in  $V$ , we need to create six new faces, and add two new vertices, so the total number of vertices is still  $n$ . Thus, the running time, memory usage and error bound of the adapted method for answering the A2A path query in the weighted region problem is still the same as the method in the main body of the paper, that is, the same as the values in the Theorem 1.

#### APPENDIX E EMPIRICAL STUDIES

##### A. Experimental Results on the V2V Path Query

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

**Effect of  $\epsilon$ .** In Figure 11, Figure 17, Figure 21 and Figure 25, we tested 6 values of  $\epsilon$  from  $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$  on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting removing value to be 2. Figure 14, Figure 18, Figure 22 and Figure 26 are the separated query time and memory usage in two steps for these results. *Roug-Ref* performs better than all the remaining algorithms in terms of query time and memory usage. The error of all algorithms is close to 0%.

**Effect of removing value.** In Figure 15, Figure 19, Figure 23 and Figure 27, we tested 5 values of removing value from  $\{1, 2, 3, 4, 5\}$  on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting  $\epsilon$  to be 0.1 on *BH-small* and *EP-small* datasets, and 0.25 on *BH* and *EP* datasets. Figure 16, Figure 20, Figure 24 and Figure 28 are the separated query time and memory usage in two steps for these results. By setting the removing value to be 2, the total running time and total memory usage of using *Roug-Ref* framework are the smallest. This is because when the removing value is larger, *Roug* can

TABLE IV  
COMPARISON OF ALL ALGORITHMS, WHERE  $A = \text{NoPrunDijk}$ ,  $B = \text{FixSP}$ ,  $C = \text{NoEdgSeqConv}$ ,  $D = \text{NoEffWeig}$

Algorithm	Time	Size	Error
<i>ConWave</i> [34]	$O(n^8 \log(\frac{LNWL}{w\epsilon}))$	Gigantic	$O(n^4)$
<i>FixSP</i> [24], [29]	$O(n^3 \log n)$	Gigantic	$O(n^3)$
<i>LogSP</i>	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	Large	$O(n \log \frac{LN}{\epsilon})$
<i>Roug-Ref</i> ( $A, B, C, D$ )	$O(n^3 \log n + nl^2 \log(\frac{LNWL}{w\epsilon}))$	Large	$O(n^3 + nl)$
<i>Roug-Ref</i> ( $A, B, \bullet, \bullet$ )	$O(n^3 \log n + nl)$	Large	$O(n^3 + nl)$
<i>Roug-Ref</i> ( $A, B, \bullet, D$ )	$O(n^3 \log n + l^2 \log(\frac{LNWL}{w\epsilon}))$	Large	$O(n^3 + l)$
<i>Roug-Ref</i> ( $A, B, \bullet, \bullet$ )	$O(n^3 \log n + l)$	Large	$O(n^3 + l)$
<i>Roug-Ref</i> ( $A, \bullet, C, D$ )	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + nl^2 \log(\frac{LNWL}{w\epsilon}))$	Large	$O(n \log \frac{LN}{\epsilon} + nl)$
<i>Roug-Ref</i> ( $A, \bullet, C, \bullet$ )	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + nl)$	Large	$O(n \log \frac{LN}{\epsilon} + nl)$
<i>Roug-Ref</i> ( $A, \bullet, \bullet, D$ )	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l^2 \log(\frac{LNWL}{w\epsilon}))$	Large	$O(n \log \frac{LN}{\epsilon} + l)$
<i>Roug-Ref</i> ( $A, \bullet, \bullet, \bullet$ )	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$	Large	$O(n \log \frac{LN}{\epsilon} + l)$
<i>Roug-Ref</i> ( $\bullet, B, C, D$ )	$O(n^2 \log n + nl^2 \log(\frac{LNWL}{w\epsilon}))$	Large	$O(n^2 + nl)$
<i>Roug-Ref</i> ( $\bullet, B, C, \bullet$ )	$O(n^2 \log n + nl)$	Large	$O(n^2 + nl)$
<i>Roug-Ref</i> ( $\bullet, B, \bullet, D$ )	$O(n^2 \log n + l^2 \log(\frac{LNWL}{w\epsilon}))$	Large	$O(n^2 + l)$
<i>Roug-Ref</i> ( $\bullet, B, \bullet, \bullet$ )	$O(n^2 \log n + l)$	Large	$O(n^2 + l)$
<i>Roug-Ref</i> ( $\bullet, \bullet, C, D$ )	$O(n \log n + nl^2 \log(\frac{LNWL}{w\epsilon}))$	Medium	$O(nl)$
<i>Roug-Ref</i> ( $\bullet, \bullet, C, \bullet$ )	$O(n \log n + nl)$	Medium	$O(nl)$
<i>Roug-Ref</i> ( $\bullet, \bullet, \bullet, D$ )	$O(n \log n + l^2 \log(\frac{LNWL}{w\epsilon}))$	Medium	$O(n + l)$
<b><i>Roug-Ref</i> (ours)</b>	$O(n \log n + l)$	<b>Small</b>	<b><math>O(n + l)</math></b>

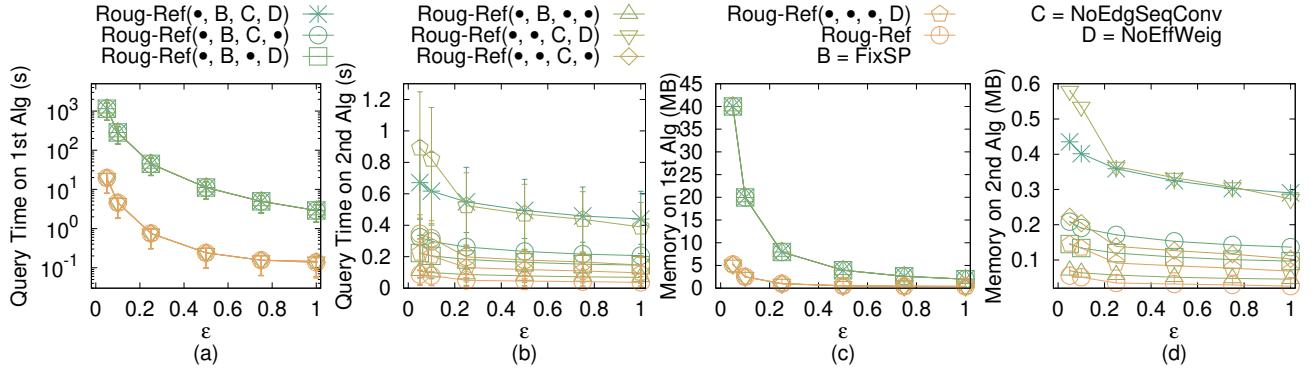


Fig. 14. Effect of  $\epsilon$  on *BH-small* dataset with separated query time and memory usage in two steps (V2V path query)

run faster, but it has a higher chance that *Ref* needs to perform the error guaranteed path refinement step. Thus, we select the optimal removing value, i.e., 2.

**Effect of dataset size (scalability test).** In Figure 29 and Figure 13, we tested 5 values of dataset size from  $\{10k, 20k, 30k, 40k, 50k\}$  on multi-resolution of *EP-small* datasets by setting  $\epsilon$  to be 0.1, and  $\{1M, 2M, 3M, 4M, 5M\}$  on multi-resolution of *EP* datasets by setting  $\epsilon$  to be 0.25 for scalability test. Figure 30 and Figure 31 are the separated query time and memory usage in two steps for these results. On multi-resolution of *EP-small* datasets, *Roug-Ref* can still beat other algorithms in terms of query time and memory usage. When the dataset size is 50k with  $\epsilon = 0.1$ , the state-of-the-art algorithm's (i.e., *FixSP*) total query time is 119,000s ( $\approx 1.5$  days), and total memory usage is 2.9GB, while our algorithm's (i.e., *Roug-Ref*) total query time is 73s ( $\approx 1.2$  min), and total memory usage is 43MB.

### B. Experimental Results on the A2A Path Query

In Figure 32 and Figure 34, we tested the A2A path query by varying  $\epsilon$  from  $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$  and setting removing value to be 2 on *EP-small* and *EP* datasets. Figure 33

and Figure 35 are the separated query time and memory usage in two steps for these results. Similar to the V2V path query, for the A2A path query, *Roug-Ref* still performs better than all the remaining algorithms in terms of query time and memory usage. This is because A2A path query is very similar to V2V path query.

### C. Experimental Results on User Study (Path Advisor)

Figure 36 and Figure 37 show the result for Path Advisor user study when varying  $\epsilon$ . Our user study in Section V-B6 has already shown that most users think the blue path (i.e., the weighted shortest path) is the most realistic one. Our user study in Section V-B6 has also already shown that 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.

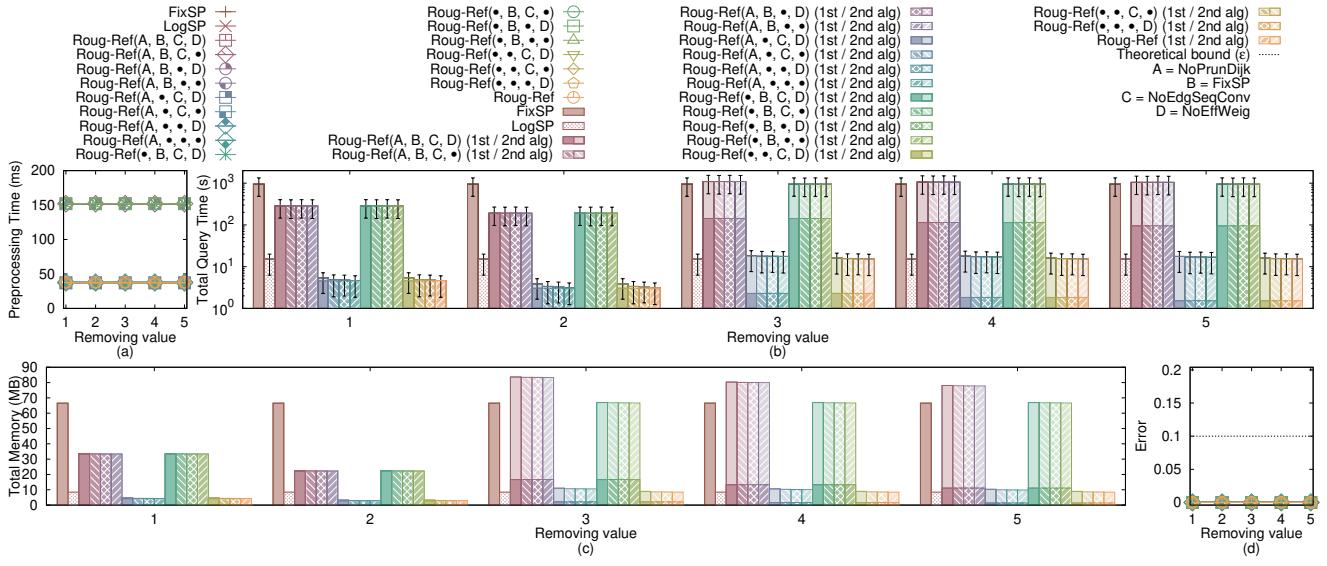


Fig. 15. Effect of removing value on *BH-small* dataset (V2V path query)

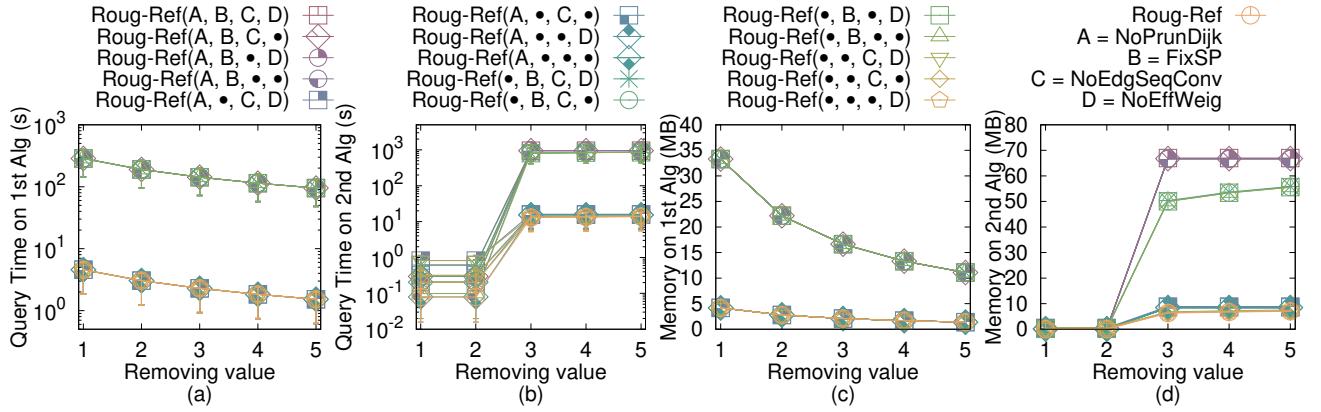


Fig. 16. Effect of removing value on *BH-small* dataset with separated query time and memory usage in two steps (V2V path query)

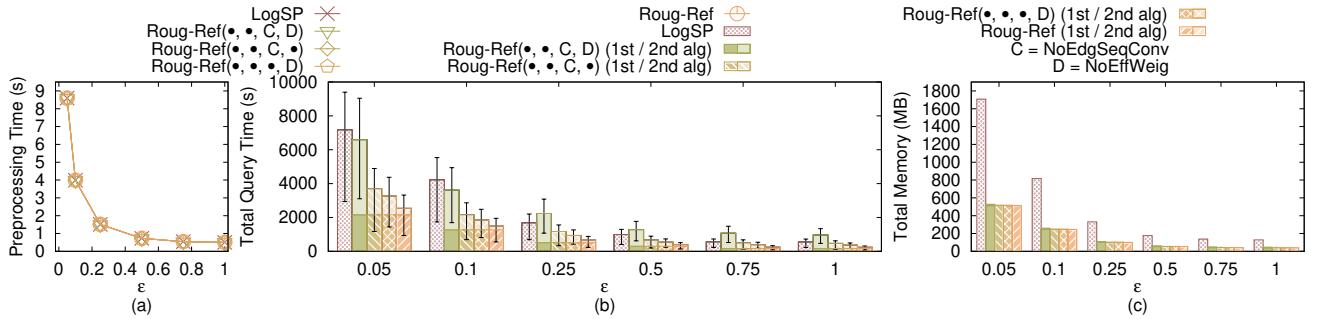


Fig. 17. Effect of  $\epsilon$  on *BH* dataset (V2V path query)

#### D. Experimental Results on User Study (Cyberpunk 2077)

We conducted another user study on Cyberpunk 2077 [2], a popular 3D computer game. The dataset is *CP* dataset [3] used in our experiment. We set the weight of a triangle in terrain to be the slope of that face. We randomly selected two points as source and destination, respectively, and repeated it 100 times to calculate the path. Figure 38 and Figure 39 show the result for Cyberpunk 2077 user study when varying  $\epsilon$ . When  $\epsilon = 0.5$ , the average query time for the state-of-the-art algorithm *FixSP* and our algorithm *Roug-Ref* are 21.61s and 0.2s, respectively. It is important to get real-time responses in computer games.

Thus, *Roug-Ref* is the most suitable algorithm for Cyberpunk 2077.

#### E. Experimental Results on Motivation Study

Figure 40 and Figure 41 show the result for seabed motivation study when varying  $\epsilon$ . Our motivation study in Section V-B7 has already shown that the blue path (i.e., the weighted shortest path) is the most realistic one since it can avoid the regions with higher hydraulic pressure, and thus, can reduce the construction cost of undersea optical fiber cable. *Roug-Ref* still has the smallest query time and memory usage.

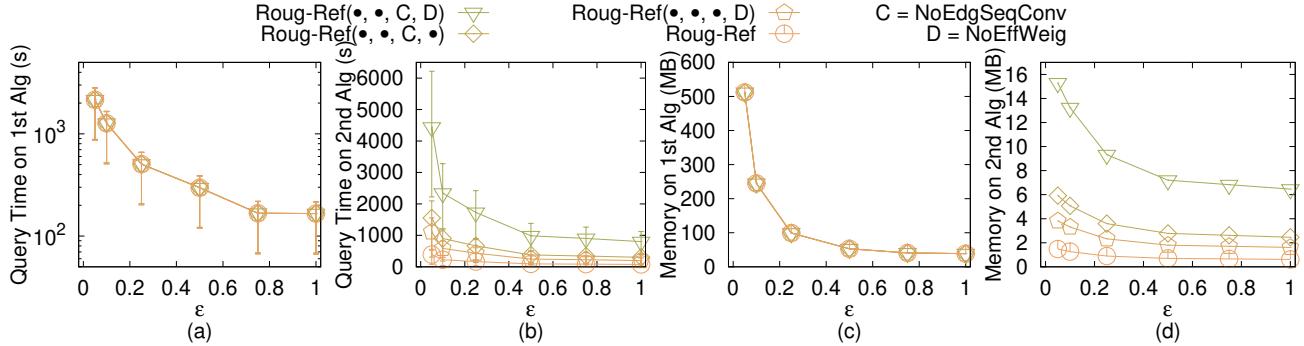


Fig. 18. Effect of  $\epsilon$  on BH dataset with separated query time and memory usage in two steps (V2V path query)

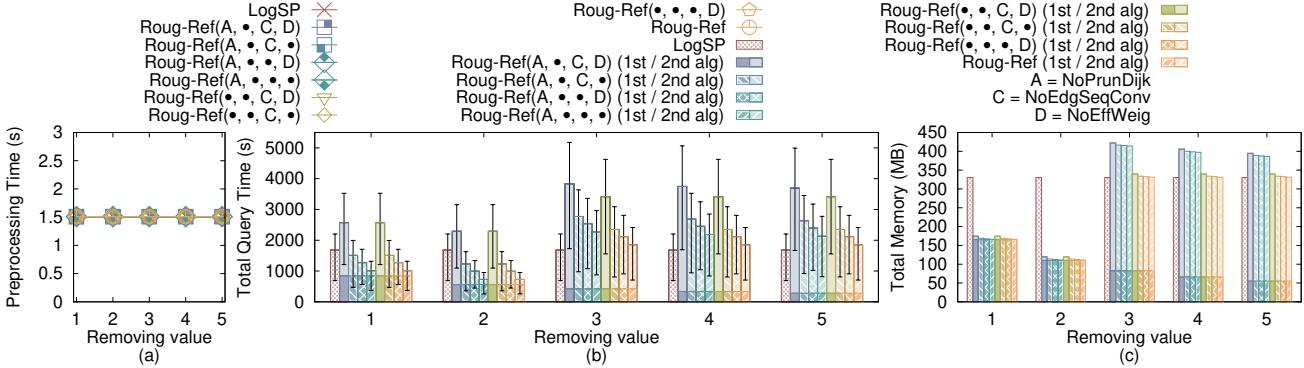


Fig. 19. Effect of removing value on BH dataset (V2V path query)

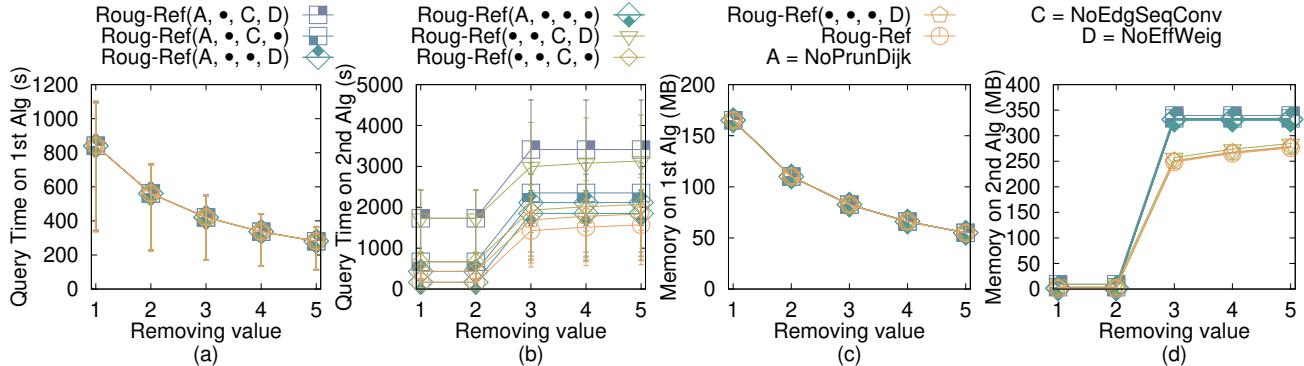


Fig. 20. Effect of removing value on BH dataset with separated query time and memory usage in two steps (V2V path query)

## F. 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 work [33], [36], [37]. Let  $T_t = (V_t, E_t, F_t)$  be our target terrain that we want to generate with  $ext_t$  edges along  $x$ -coordinate,  $ey_t$  edges along  $y$ -coordinate and dataset size of  $DS_t$ , where  $DS_t = 2 \cdot ext_t \cdot ey_t$ . Let  $T_o = (V_o, E_o, F_o)$  be the original terrain that we currently have with  $ext_o$  edges along  $x$ -coordinate,  $ey_o$  edges along  $y$ -coordinate and dataset size of  $DS_o$ , where  $DS_o = 2 \cdot ext_o \cdot ey_o$ . We then generate  $(ext_t + 1) \cdot (ey_t + 1)$  2D points  $(x, y)$  based on a Normal distribution  $N(\mu_N, \sigma_N^2)$ , where  $\mu_N = (\bar{x} = \frac{\sum_{v_o \in V_o} x_{v_o}}{(ext_o+1) \cdot (ey_o+1)}, \bar{y} = \frac{\sum_{v_o \in V_o} y_{v_o}}{(ext_o+1) \cdot (ey_o+1)})$  and  $\sigma_N^2 = (\frac{\sum_{v_o \in V_o} (x_{v_o} - \bar{x})^2}{(ext_o+1) \cdot (ey_o+1)}, \frac{\sum_{v_o \in V_o} (y_{v_o} - \bar{y})^2}{(ext_o+1) \cdot (ey_o+1)})$ . 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 F PROOFS

**Lemma 1.** *There are at most  $k_{SP} \leq 2(1 + \log_{\lambda} \frac{L}{r})$  Steiner points on each edge in  $E$  when placing Steiner point based on  $\epsilon$  in algorithm Roug.*

*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  $\epsilon$  in algorithm Roug, each set of Steiner points contains at most  $(1 + \log_{\lambda} \frac{L}{r})$  Steiner points, where the 1 comes from the first Steiner point that

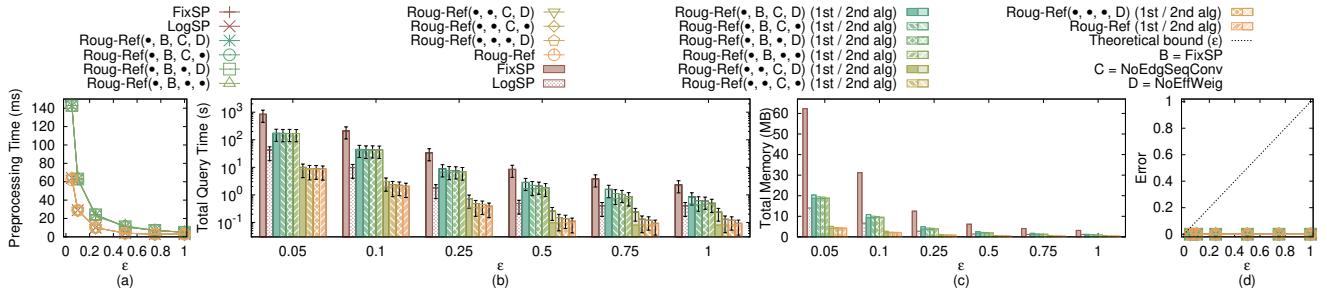


Fig. 21. Effect of  $\epsilon$  on EP-small dataset (V2V path query)

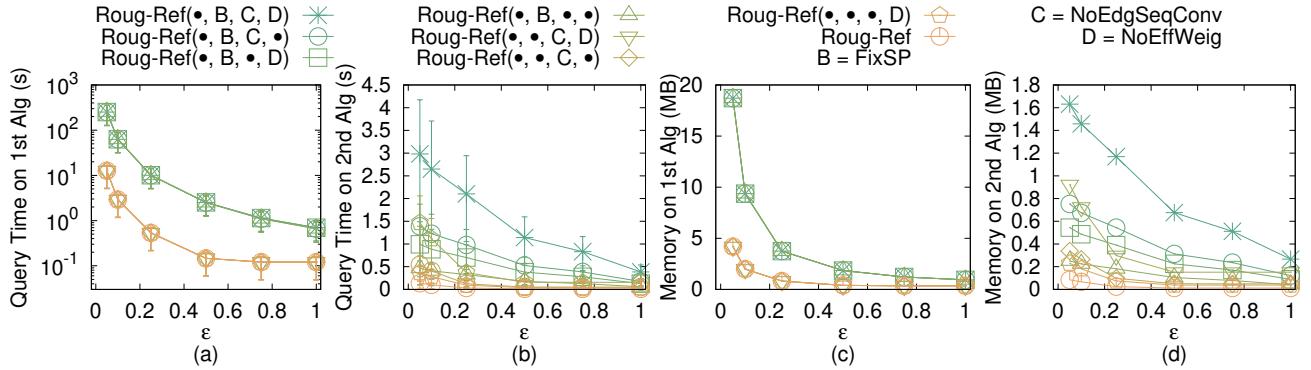


Fig. 22. Effect of  $\epsilon$  on EP-small dataset with separated query time and memory usage in two steps (V2V path query)

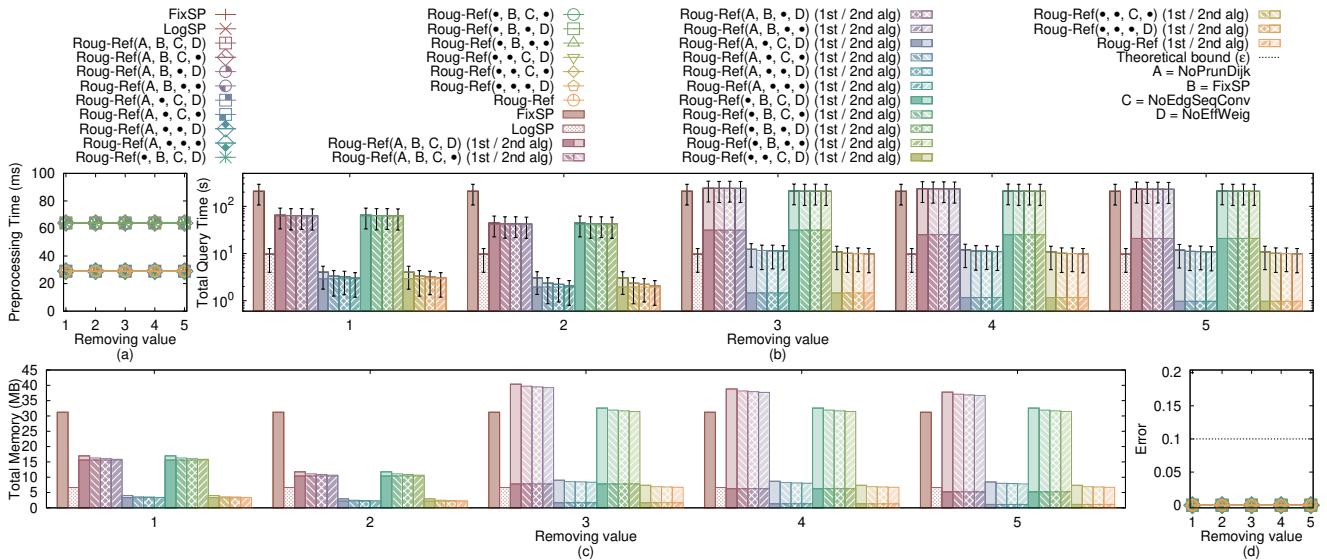


Fig. 23. Effect of removing value on EP-small dataset (V2V path query)

is the nearest one from the endpoint. Therefore, we have  $k_{SP} \leq 2(1 + \log_{\lambda} \frac{L}{r})$ . ■

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

*Proof:* The mathematical derivation is like we regard  $\epsilon'$  as an unknown and solve a quadratic equation. The derivation is as follows.

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

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

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

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

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

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

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

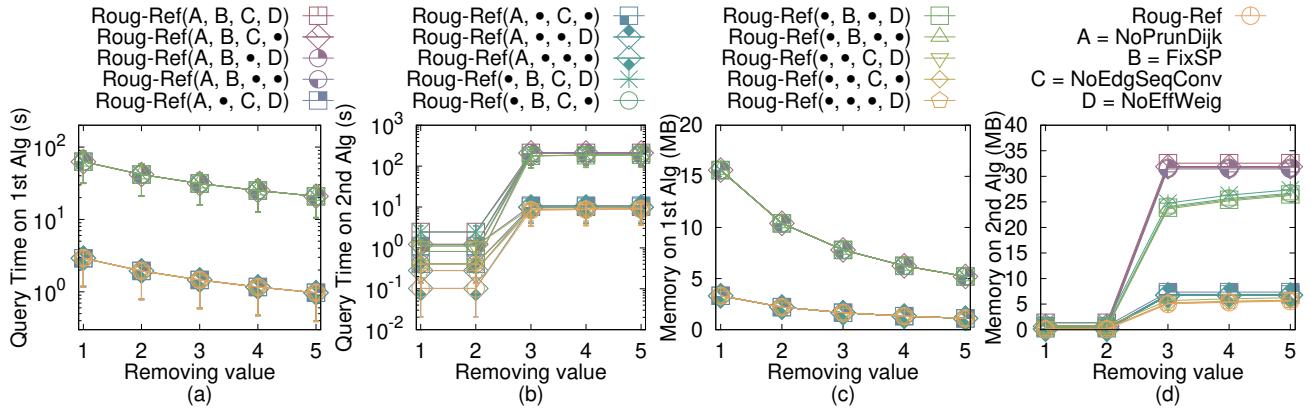


Fig. 24. Effect of removing value on EP-small dataset with separated query time and memory usage in two steps (V2V path query)

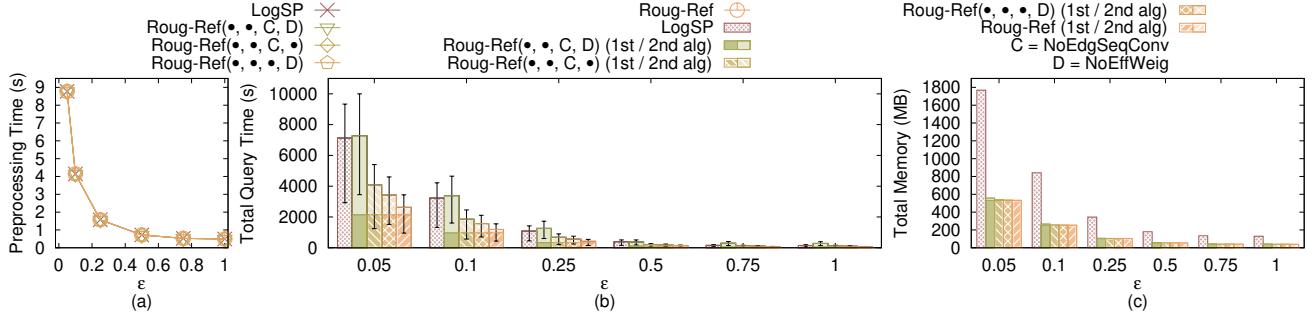


Fig. 25. Effect of  $\epsilon$  on EP dataset (V2V path query)

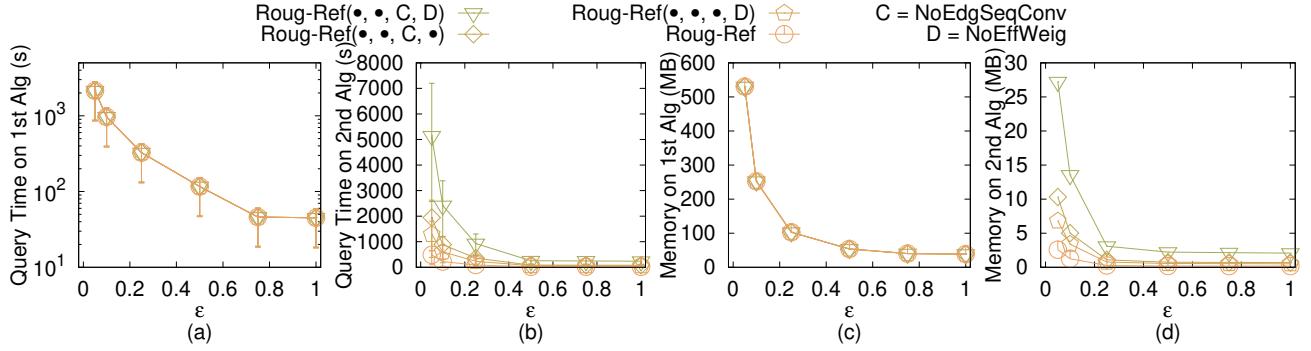


Fig. 26. Effect of  $\epsilon$  on EP dataset with separated query time and memory usage in two steps (V2V path query)

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 find 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  $\omega = (y_az_b - z_ay_b)^2 + (z_ax_b - x_ay_b)^2 + (x_az_b - y_ay_b)^2$ . ■

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

*Proof of Theorem 2:* Originally, if we do not remove Steiner points in algorithm Roug, i.e., we place Steiner point based on  $\epsilon$ , then following Lemma 1, the number of Steiner points  $k_{SP}$  on each edge is  $O(\log_{\lambda} \frac{L}{r})$ , where  $\lambda = (1 + \epsilon' \cdot \sin \theta)$  and  $r = \epsilon' h$ . 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  $\eta$  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  $O(n)$ .

*Proof of Theorem 3:* Firstly, we prove the running time.

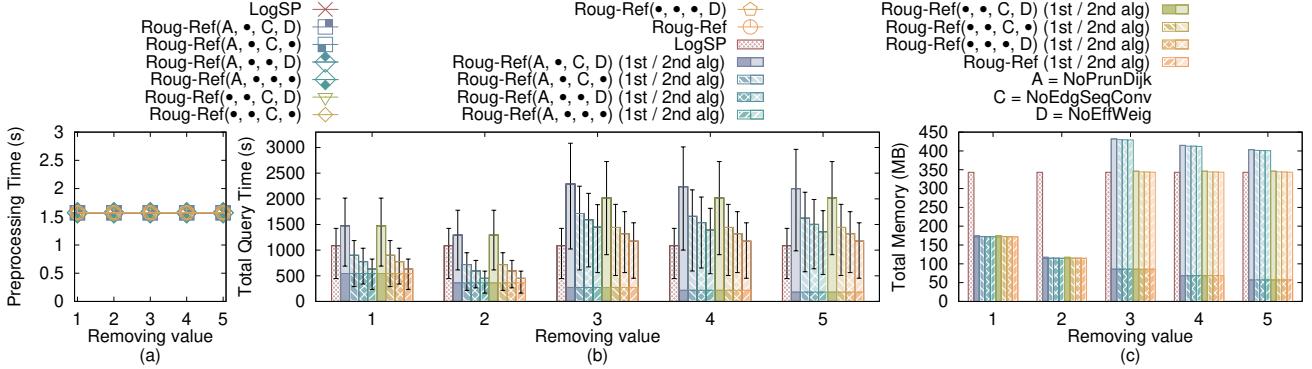


Fig. 27. Effect of removing value on EP dataset (V2V path query)

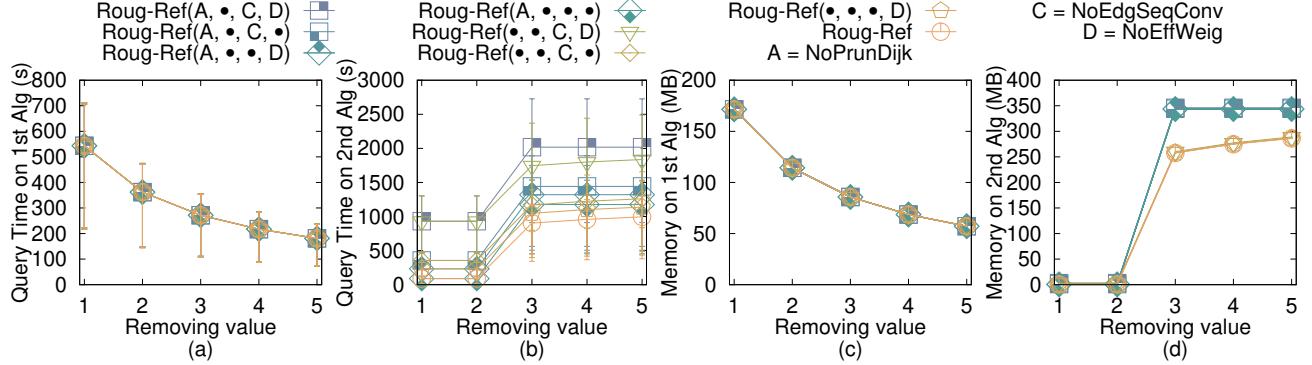


Fig. 28. Effect of removing value on EP dataset with separated query time and memory usage in two steps (V2V path query)

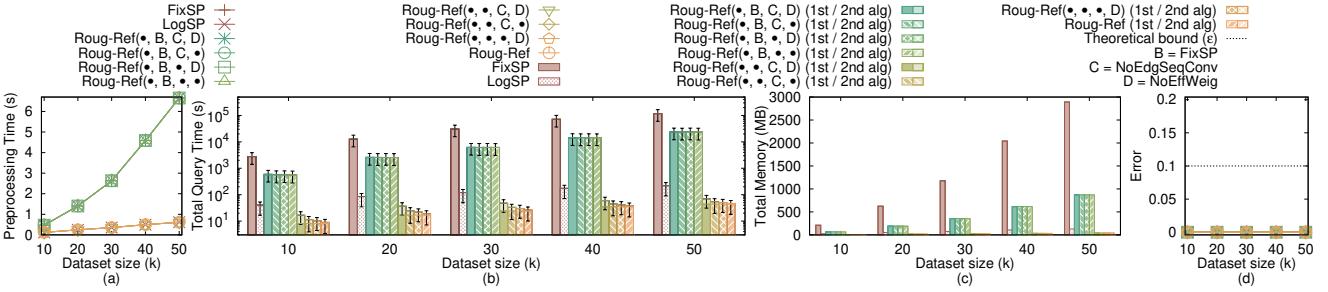


Fig. 29. Effect of dataset size on multi-resolution of EP-small datasets (V2V path query)

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 care 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  $\zeta$  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 of Theorem 3:* 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*

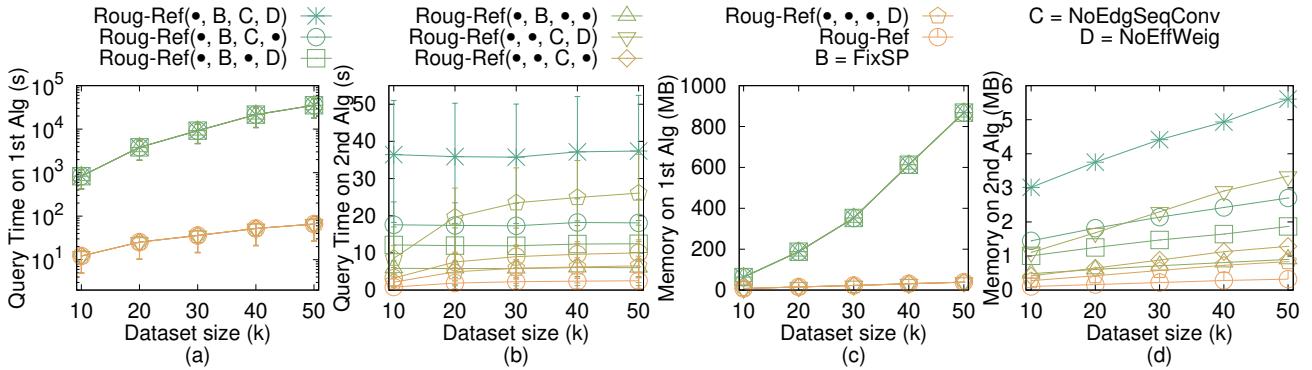


Fig. 30. Effect of dataset size on multi-resolution of EP-small datasets with separated query time and memory usage in two steps (V2V path query)

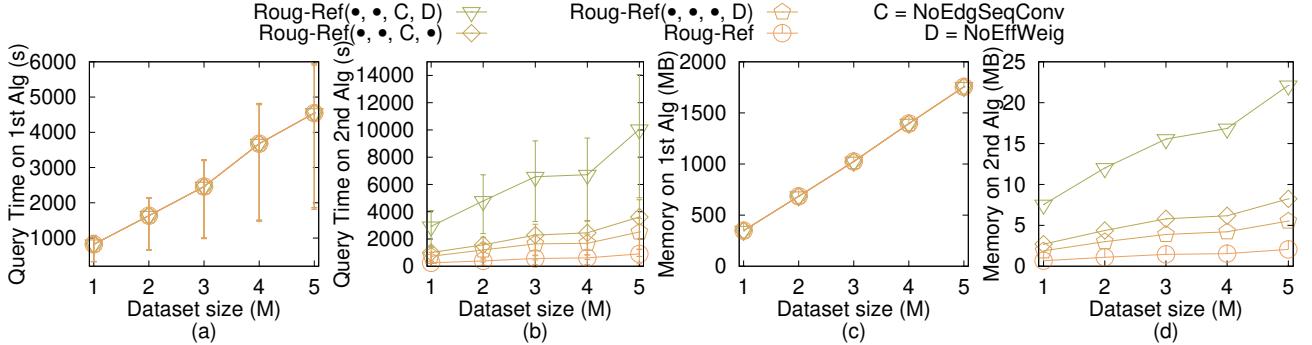


Fig. 31. Effect of dataset size on multi-resolution of EP datasets with separated query time and memory usage in two steps (V2V path query)

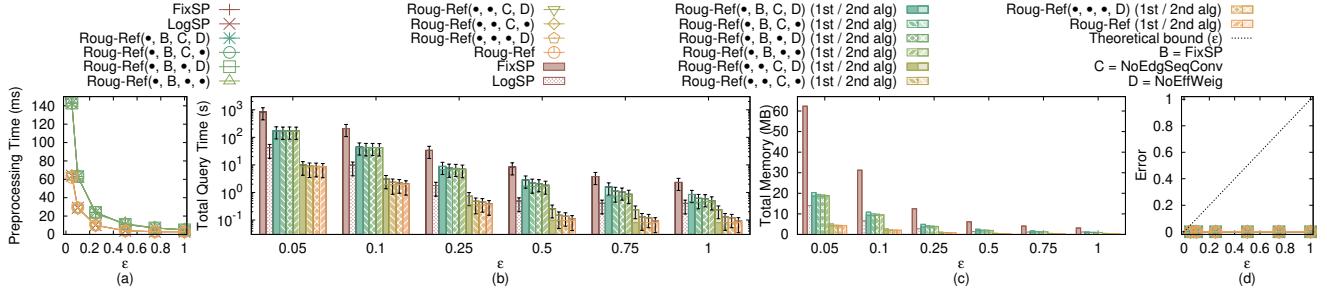


Fig. 32. A2A path query on EP-small dataset

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 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  $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$ . But the constant term  $O(\log \frac{LN}{\epsilon})$  is not important and can be omitted, so we obtain the total memory usage  $O(n + l)$ . (3) In general, the total memory usage is  $O(n + l)$ .

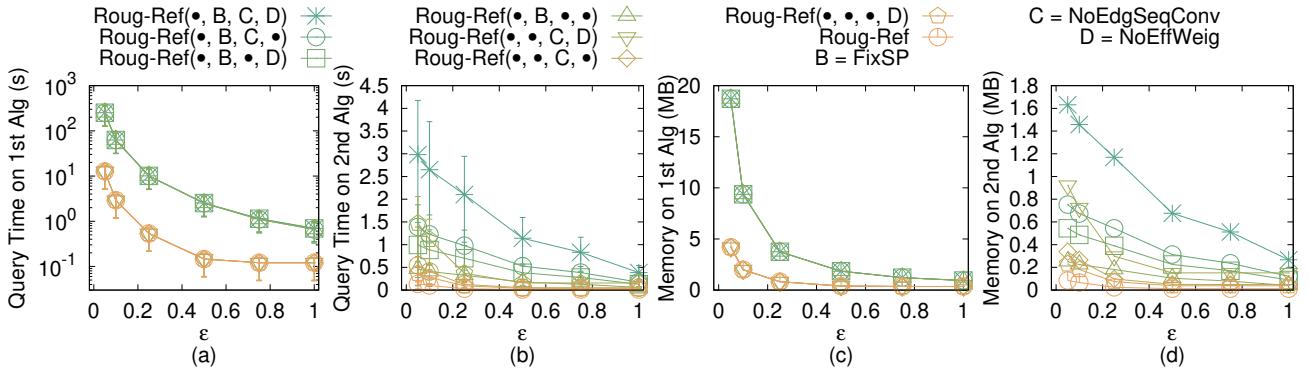


Fig. 33. A2A path query on EP-small dataset with separated query time and memory usage in two steps

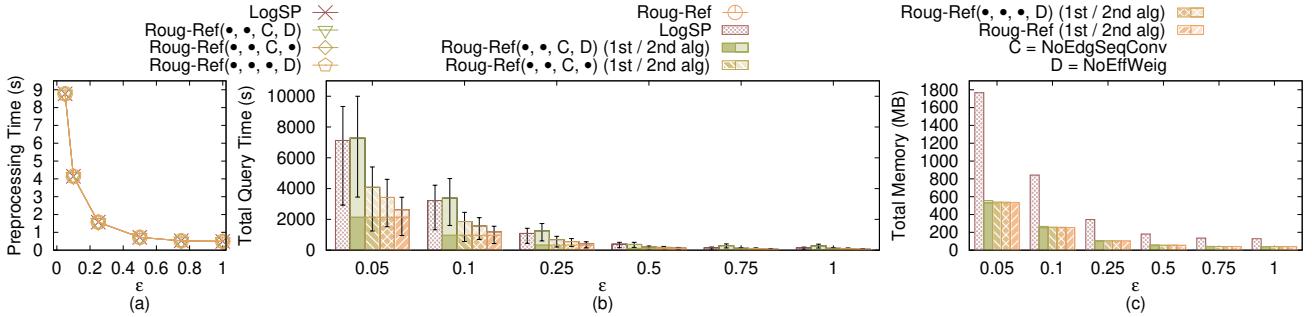


Fig. 34. A2A path query on EP dataset

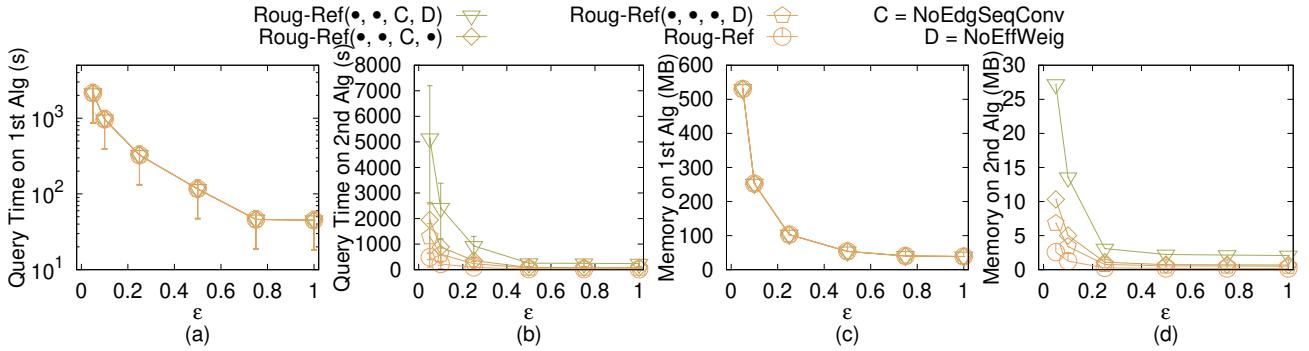


Fig. 35. A2A path query on EP dataset with separated query time and memory usage in two steps

Finally, we prove the error bound. Recall one baseline algorithm *LogSP*, the algorithm that uses Dijkstra's algorithm on the weighted graph constructed by  $SP_{Ref}$  and  $V$ , i.e., the rough path calculation step of our algorithm *Roug* (by changing the input error from  $\eta\epsilon$  to  $\epsilon$ ). We define the path calculated by algorithm *LogSP* between  $s$  and  $t$  to be  $\Pi_{LogSP}(s, t)$ . The work [11], [28] show that  $|\Pi_{LogSP}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ . A proof sketch appears in Theorem 1 of work [11] and a detailed proof appears in Theorem 3.1 of work [28]. But, the work [11], [28] have  $|\Pi_{LogSP}(s, t)| \leq (1 + (2 + \frac{2W}{(1-2\epsilon') \cdot w})\epsilon')|\Pi^*(s, t)|$  where  $0 < \epsilon' < \frac{1}{2}$ . After substituting  $(2 + \frac{2W}{(1-2\epsilon') \cdot w})\epsilon' = \epsilon$  with  $0 < \epsilon' < \frac{1}{2}$  and  $\epsilon > 0$ , we have  $|\Pi_{LogSP}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$  where  $\epsilon > 0$ . This error bound is always true no matter whether the edge sequence passed by  $\Pi_{LogSP}(s, t)$  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  $\Pi_{Roug}(s, t)$  in the rough path calculation step, so by adapt  $\eta\epsilon$  as the input error in algorithm *LogSP*, we have  $|\Pi_{Roug}(s, t)| \leq (1 + \eta\epsilon)|\Pi^*(s, t)|$ . Next, in the path checking step of algorithm *Ref*, if  $|\Pi_{Ref\_2}(s, t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s, t)|$ , we return  $\Pi_{Ref\_2}(s, t)$  as output  $\Pi(s, t)$ , which implies that  $|\Pi_{Ref\_2}(s, t)| = |\Pi(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ . Otherwise, we use the error guaranteed path refinement step in algorithm *Ref*, and we return  $\Pi_{Ref\_3}(s, t)$  as output  $\Pi(s, t)$ , where the error bound is the same as in algorithm *LogSP*, i.e.,  $|\Pi_{Ref\_3}(s, t)| = |\Pi(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ . Therefore, algorithm *Roug-Ref* guarantees that  $|\Pi(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ . ■

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

*Proof of Theorem 5:* 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  $E$  [29]. Since there

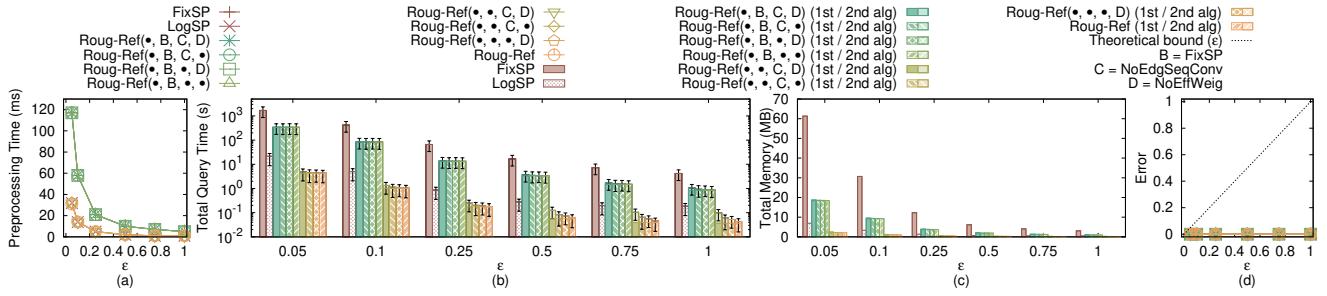


Fig. 36. Effect of  $\epsilon$  for Path Advisor user study

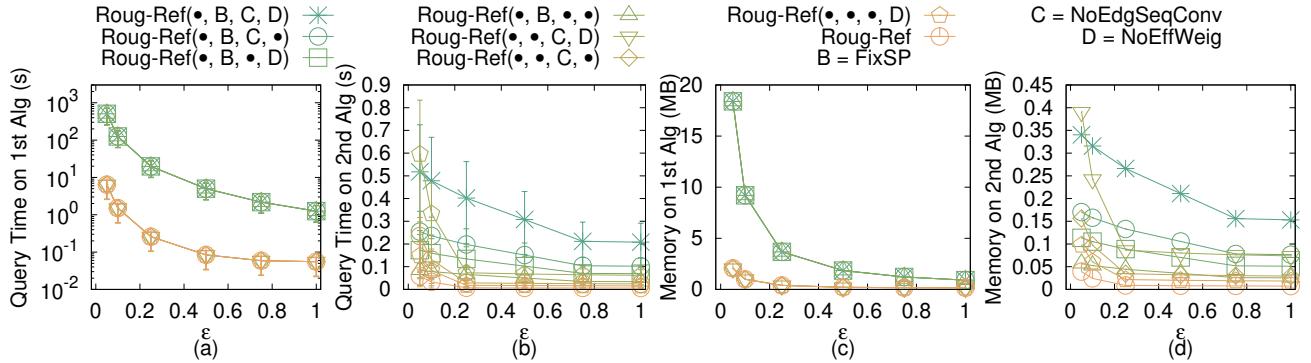


Fig. 37. Effect of  $\epsilon$  for Path Advisor user study with separated query time and memory usage in two steps

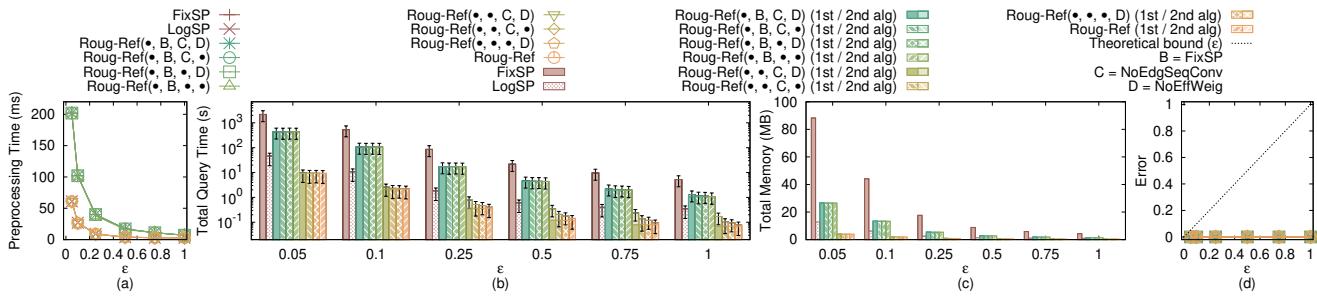


Fig. 38. Effect of  $\epsilon$  for Cyberpunk 2077 user study

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 work [28] shows that  $|\Pi_{\text{FixSP}}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ . ■

**Theorem 6.** 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  $|\Pi_{\text{LogSP}}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ , where  $\Pi_{\text{LogSP}}(s, t)$  is the path result returned by algorithm *LogSP*.

*Proof of Theorem 6:* 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 shows that  $|\Pi_{\text{LogSP}}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ . ■

**Theorem 7.** The total running time for algorithm *Roug-Ref(NoPrunDijk, •, •, •)* is  $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$ , the total memory usage is  $O(n \log \frac{LN}{\epsilon} + l)$ . It guarantees that  $|\Pi_{\text{Roug-Ref(NoPrunDijk,•,•,•)}}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ , where  $\Pi_{\text{Roug-Ref(NoPrunDijk,•,•,•)}}(s, t)$  is the path result returned by algorithm *Roug-Ref(NoPrunDijk, •, •, •)*.

*Proof of Theorem 7:* 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.

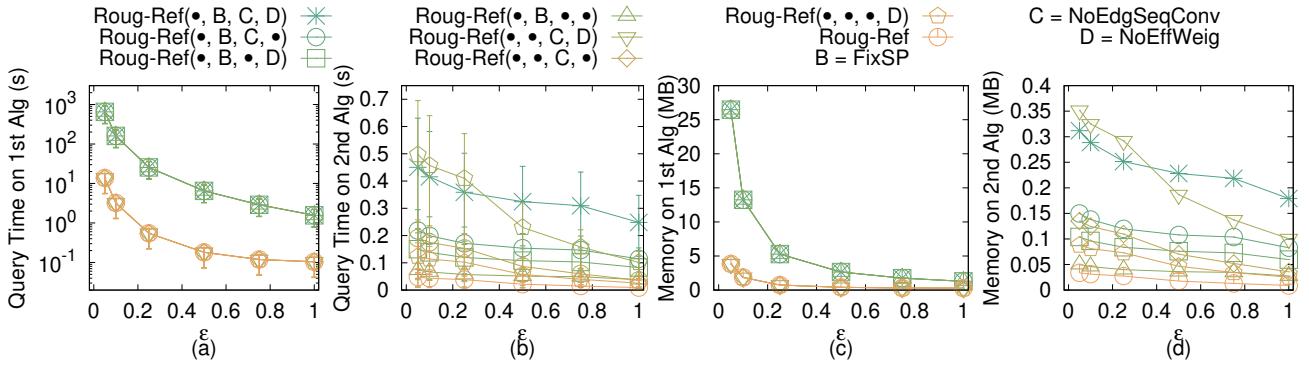


Fig. 39. Effect of  $\epsilon$  for Cyberpunk 2077 user study with separated query time and memory usage in two steps

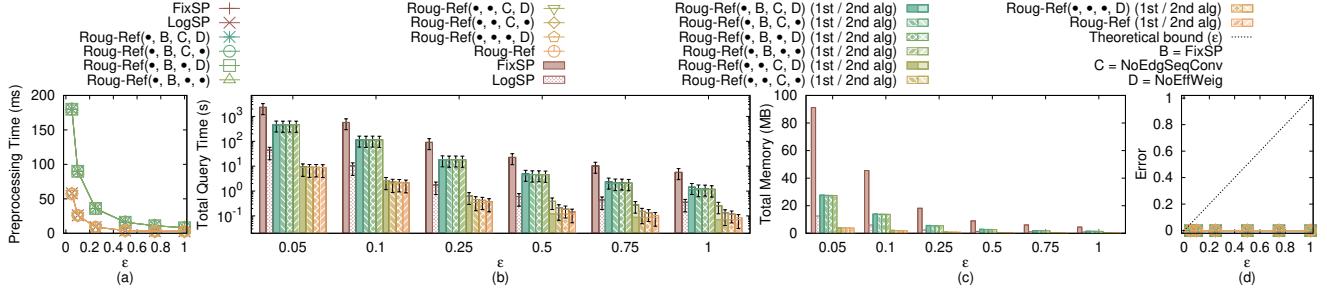


Fig. 40. Effect of  $\epsilon$  for seabed motivation study

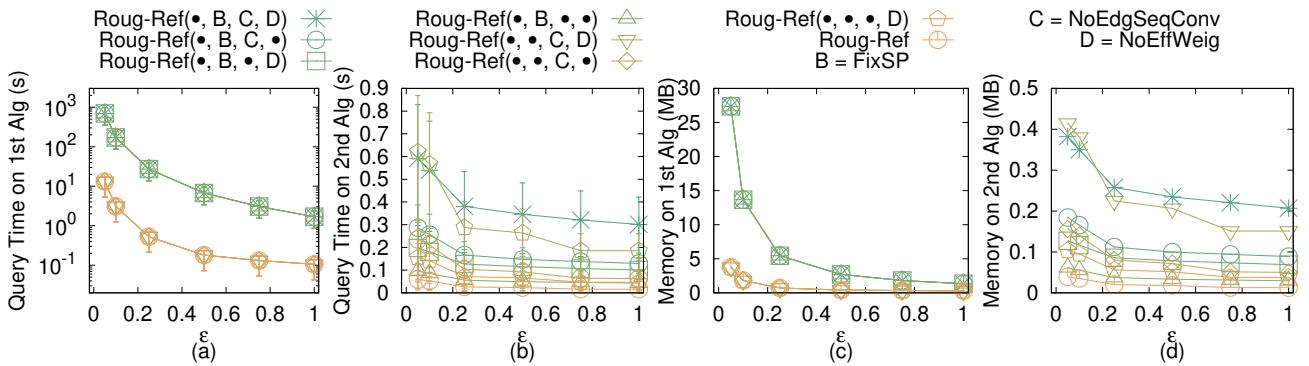


Fig. 41. Effect of  $\epsilon$  for seabed motivation study with separated query time and memory usage in two steps

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  $G_{Ref}$  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  $G_{Ref}$  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  $O(n \log \frac{LN}{\epsilon} + l)$ . ■

**Theorem 8.** *The total running time for algorithm Roug-Ref(●, FixSP, ●, ●) is  $O(n^2 \log n + l)$ , the total memory usage is  $O(n^2 + l)$ . It guarantees that  $|\Pi_{Roug-Ref(●, FixSP, ●, ●)}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ , where  $\Pi_{Roug-Ref(●, FixSP, ●, ●)}(s, t)$  is the path result returned by algorithm Roug-Ref(●, FixSP, ●, ●).*

*Proof of Theorem 8:* The difference between Roug-Ref(●, FixSP, ●, ●) 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  $E$  [29]. 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  $|\Pi_{FixSP}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ , where  $\Pi_{FixSP}(s, t)$  is the path result returned by algorithm *FixSP*. Since from Theorem 1, we also have  $|\Pi_{LogSP}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ , where  $\Pi_{LogSP}(s, t)$  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*(•, *FixSP*, •, •). ■

**Theorem 9.** *The total running time for algorithm *Roug-Ref*(•, •, *NoEdgSeqConv*, •) is  $O(n \log n + nl)$ , the total memory usage is  $O(nl)$ . It guarantees that  $|\Pi_{Roug-Ref(•, •, NoEdgSeqConv, •)}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ , where  $\Pi_{Roug-Ref(•, •, NoEdgSeqConv, •)}(s, t)$  is the path result returned by algorithm *Roug-Ref*(•, •, *NoEdgSeqConv*, •).*

*Proof of Theorem 9:* 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 10.** *The total running time for algorithm *Roug-Ref*(•, •, •, *NoEffWeig*) is  $O(n \log n + l^2 \log(\frac{LNWL}{w\epsilon}))$ , the total memory usage is  $O(n + l)$ . It guarantees that  $|\Pi_{Roug-Ref(•, •, •, NoEffWeig)}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ , where  $\Pi_{Roug-Ref(•, •, •, NoEffWeig)}(s, t)$  is the path result returned by algorithm *Roug-Ref*(•, •, •, *NoEffWeig*).*

*Proof of Theorem 10:* 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, they first take  $O(l)$  time for computing the 3D surface Snell's ray  $\Pi_m$  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, they take  $O(\log \frac{L_i}{\delta})$  time for deciding 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  $\delta = \frac{hew}{6lW}$  and  $L_i \leq L$  for  $\forall i \in \{1, \dots, l\}$ , the running time is  $O(\log \frac{L_i}{hew})$ . Since we run the above two nested loops  $l$  times, the total running time is  $O(l^2 \log \frac{L_i}{hew})$ . So the running time of the Snell's law path refinement step without the effective weight pruning sub-step in algorithm *Ref* is  $O(l^2 \log \frac{L_i}{hew})$ . 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  $O(n + l)$ . ■