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 a terrain surface has aroused widespread concern in industry and academia. In this paper, we propose an efficient method for the *weighted region problem* on 3D weighted terrain surfaces on-the-fly. 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 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. Our experimental study shows that in realistic settings, our algorithm is capable of improvements in running time and memory usage of up to 1630 times and 40 times compared with the best-known algorithm.**

I. Introduction

In recent years, terrain data becomes increasingly widespread in industry and academia [38]. In industry, many companies and applications, including Metaverse [5], [28], [29], Cyberpunk 2077 (a 3D computer game) [1] and Google Earth [4], are using terrain surfaces, such as mountains and valleys, with different features (e.g., water and grassland) to help users compute the shortest path to the destination. In academia, researchers paid considerable attention to studying shortest path queries on terrain datasets [17], [20], [22], [23], [32], [35], [36], [39]. 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 (a) 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 (a), 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 is involved in numerous applications with different meanings of the weights of faces on the terrain, including autonomous vehicles' obstacle avoidance path planning, human's route-recommendation systems and laying electrical cables [11], [19], [21], [25], [39], [40]. In Figure 1 (a), 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 (leading to a smaller weighted length). In a real-life example for the placement of undersea optical fiber cable on the seabed (which is a terrain), we aim to minimize the weighted length of the cable for cost saving. It is worth mentioning that over 1.35×10^5km of undersea cables have been constructed nowadays [24]. 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 [13]. 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 following the weighted shortest path and the unweighted shortest path are USD \$45.8M and \$54.8M, respectively, which shows the usefulness of finding 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 where different regions are assigned with different weights depending on the application nature. This problem is called the *weighted region 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 greater than two [16]. There are three categories of algorithms for solving the weighted region problem on-the-fly *approximately*: (1) *wavefront* approach [34], (2) *heuristics* approach [25], [37], and (3) *Steiner point* approach [22], [27].

1) ***Wavefront approach****:* It uses Snell's law (one widely known fact from physics) [33] and the continuous Dijkstra's algorithm [34] (like a wavefront) to calculate an approximate weighted shortest path in more than $O(n^8)$ time, which is very large, since it involves many redundant checking operations.

2) ***Heuristics approach****:* It uses simulated annealing [25] and the genetic algorithm [37] to calculate the weighted

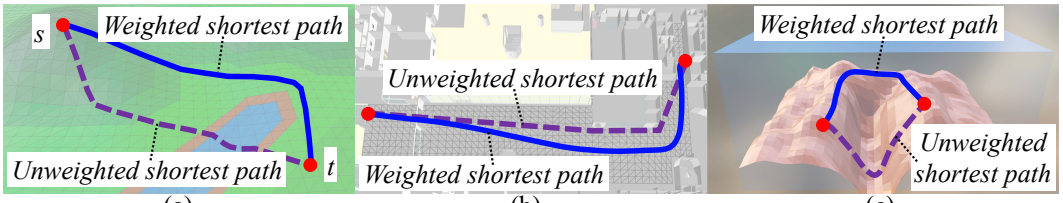

Fig. 1. (a) A terrain surface and paths, (b) paths in Path Advisor, and (c) paths on seabed

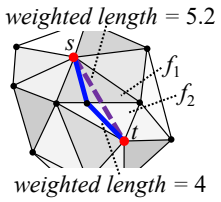

Fig. 2. A weighted terrain surface with the blue path satisfies Snell's law but the purple straight line does not

shortest path, but it cannot guarantee on the quality of the path returned (in terms of the length of the path) with a given time limit and a given memory capacity.

3) ***Steiner point approach****:* It (1) places discrete points (i.e., Steiner points) on edges in E or faces in F, and (2) uses Dijkstra's algorithm [18] on a weighted graph constructed using these Steiner points and V to calculate the weighted shortest path. The best-known existing studies [22], [27] 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. Our experimental results show that the best-known algorithm [22], [27] runs in 119,000s ($\approx$ 1.5 days) on a terrain with 50k faces, which is not acceptable.

C. Our Efficient Algorithm

We propose an efficient on-the-fly 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 ϵ is a non-negative real user parameter for controlling the error ratio and is called the *error parameter*. Algorithm *Roug-Ref* is a step-by-step algorithm involving algorithm *Roug* and algorithm *Ref*. (1) In algorithm *Roug*, given T, s, t, and ϵ, we efficiently find a *rough path* between s and t with error guarantee of the approximate ratio equal to $(1+\eta\epsilon)$, where $\eta > 1$ is a constant and is calculated based on T and ϵ (note that $\eta \in (1, 2]$ on average in our experiments). (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 [33] is an important geometry information of T between two adjacent faces in F that share one edge. This law, which originated from Physics, can be applied to calculating the weighted shortest path. In Physics, when a light ray passes the boundary of two different media (e.g., air and glass), it bends at the boundary due to the fact that light seeks the path with the minimum time as described by *Fermat's Principle* [12]. The angles of incidence and refraction for the light satisfy Snell's law. Our algorithm employs Snell's law to efficiently calculate the weighted shortest path, the feature that is absent in the best-known existing studies [22], [27]. Figure 2 shows how Snell's law is applied to calculating the weighted shortest path. In this figure, for the sake of illustration, all faces are on a plane. Consider a source s and a destination t. If all faces have the same weights (=1) (which is the setting of the unweighted terrain), it is obvious that the purple straight line is the (unweighted) shortest path. However, when the weight of face f_1 is 3 and the weight of face f_2 is 2, the weighted shortest path, satisfying Snell's law, is the blue path, which involves two line segments and has the weighted length = 4. We can see that the two line segments bend at the boundary between f_1 and f_2 (i.e., the edge shared by f_1 and f_2). Note that the weighted length of the purple line is $5.2 > 4$.

D. Contributions and Organization

We summarize our major contributions.

(1) We are the first to propose the efficient algorithm *Roug-Ref* that calculates a $(1+\epsilon)$-approximate weighted shortest path on-the-fly, since there is no existing work that uses the rough-refine idea for calculating the weighted shortest path within the error bound. We 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 the geometry information on T for each face in F (e.g., the weight of a face, the internal angle of a face, and the length of an edge) that is not considered in the best-known algorithm [22], [27], (b) the use of a *progressive* idea in algorithm *Ref* to further reduce the search area, (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 smaller 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 [22], [27] in terms of running time and memory usage. Our experimental results show that our algorithm runs up to 1630 times faster than the best-known algorithm [22], [27] 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 [22], [27] runs in 119,000s ($\approx$ 1.5 days) and

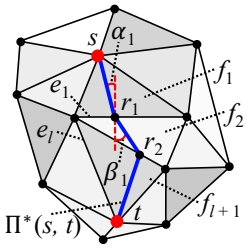

Fig. 3. An example of $\Pi^*(s,t)$

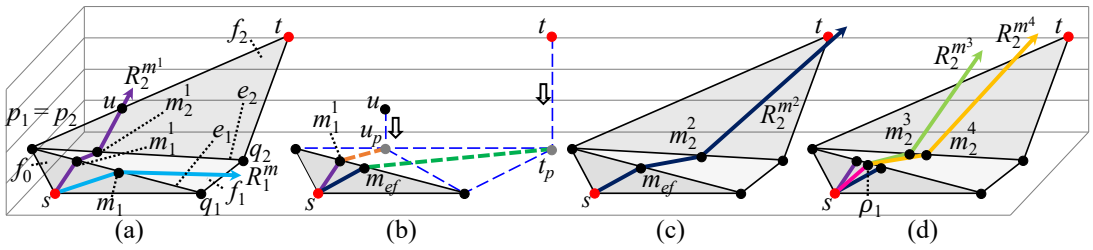

Fig. 4. Snell's law path refinement step in *Ref* (a) with initial ray for calculating effective weight on the effective face $\triangle u_p p_1 q_1$, (b) for calculating m_{ef} using the weight of f_0 and the effective weight of $\triangle u_p p_1 q_1$, (c) with final ray passing through m_{ef}, and (d) processing on the remaining edges

uses 2.9GB of memory. Furthermore, our user study shows that 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.

II. Preliminary

A. Notations and Definitions

1) ***Terrain surfaces and paths****:* Consider a terrain surface 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, x_v, y_v, and z_v. Let L be the length of the longest edge of T, and N be the smallest integer value that is larger than or equal to the coordinate value of any vertex in V. If two faces share a common edge, they are said to be *adjacent*. Each face $f_i \in F$ is assigned a weight w_i, which is a positive real number, and the weight of an edge is the smaller weight of the two faces containing the edge. The maximum and minimum weights of the face in F are denoted by W and w, respectively. The minimum height of a face in F is denoted by h. Given a face f_i, and two points p and q on f_i, 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* $\Pi^*(s,t) = (s, r_1, \ldots, r_l, t)$ on T (with $l \geq 0$) such that the weighted distance $\sum_{i=0}^{l} D(r_i, r_{i+1})$ is the minimum, where $r_0 = s$ and $r_{l+1} = t$. Here, for each $i \in \{1, \ldots, l\}$, r_i is a point on an edge in E, is named as an *intersection point* in $\Pi^*(s,t)$. The blue path in Figure 3 shows an example of $\Pi^*(s,t) = (s, r_1, r_2, t)$ on a terrain surface. We define $|\cdot|$ to be the weighted distance of a path. For example, $|\Pi^*(s,t)|$ is the weighted distance of $\Pi^*(s,t)$. Given a face f_i, and two points p and q on f_i, we define $\overline{pq}$ 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****:* Let $S = ((v_1, v_1'), \ldots, (v_l, v_l')) = (e_1, \ldots, e_l)$ be a sequence of edges that $\Pi^*(s,t)$ connects from s to t in order based on T.

and S is said to be *passed by* $\Pi^*(s,t)$. Let l be the number of edges in S. Let $F(S) = (f_1, f_2, \ldots, f_l, f_{l+1})$ be a sequence of adjacent faces with respect to S, such that for every f_i with $i \in \{2, \ldots l\}$, f_i is the face containing e_i and e_{i+1} in S, while f_1 is the adjacent face of f_2 at e_1 and f_{l+1} is the adjacent face of f_l at e_l. Let $W(S) = (w_1, \ldots, w_l, w_{l+1})$ be a weight list with respect to $F(S)$ such that for every w_i with $i \in \{1, \ldots l+1\}$, w_i is the face weight of f_i in $F(S)$. We define α_i and β_i to be the angles of incidence and refraction of $\Pi^*(s,t)$ on e_i for $i \in \{1, \ldots l\}$, respectively. Proposition 1 states Snell's law with $i \in \{1, \ldots, l\}$.

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 point s (or t) is not in V, we can regard this point as a new vertex and then add 3 new triangles each involving this point and two vertices of the face containing this point. Then, we can use the V2V path query on this modified setting for the A2A path query (since the point becomes a new vertex). Thus, in this paper, we focus on the V2V path query and study the A2A path query in the appendix. A notation table can be found 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 [14] for finding the *exact* shortest path on-the-fly is to unfold the 3D terrain surface into a 2D terrain surface, and connect the source s and destination t using a straight line segment. However, it cannot be used for the weighted case. Even in 2D, when the *exact* weighted shortest path passes through the boundary of two faces with different weights, it will bend when it crosses an edge in E every time [34], as illustrated in Figure 2. Thus, we cannot connect s and t with a straight line segment in the unfolded 2D weighted 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 [16]. In Figure 3, 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$\mathbb{Q}$) [16].

III. Related Work

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

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

1) ***Wavefront approach****:* It calculates the weighted shortest path by Snell's law and continuous Dijkstra's algorithm. Algorithm *Continuous Wavefront* (*ConWave*) [34] returns a path with a $(1+\epsilon)$-approximate 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. To handle this, algorithm *ConWave* 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, the 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, and uses Snell's law on S to find the result path, then the running time can be reduced.

2) ***Heuristics approach****:* It uses simulated annealing [25] and genetic algorithm [37] to calculate the weighted shortest path, but it cannot guarantee on the length of the path returned with a given time limit and a given memory capacity.

3) ***Steiner point approach****:* It uses Dijkstra's algorithm on the weighted graph constructed using Steiner points and V to calculate the weighted shortest path. Algorithm *Fixed Steiner Point placement scheme* (*FixSP*) [22], [27] places Steiner points on edges in E, and algorithm *Unfixed Steiner Point placement scheme* (*UnfixSP*) [8] places Steiner points on faces in F. The experimental results of work [20] show that the running time of algorithm *UnfixSP* is 10^3 larger than that of algorithm *FixSP*. Thus, algorithm *FixSP*, which runs in $O(n^3 \log n)$ time, is regarded as the best-known algorithm for solving the weighted region problem. But, it is still very slow due to two reasons.

(3a) *No usage of Snell's law*: It does not utilize any geometry information on T *between two adjacent faces in* F *that share one edge* (with the help of Snell's law). So, they need to place many Steiner points on edges in E, and each two adjacent Steiner points on the same edge are very close to each other. (3b) *Same number of Steiner points per edge*: It 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, and the length of an edge). So, it always places the same number of Steiner points on edges in E. That is, no matter how T looks like, it always needs to place $O(n^2)$ Steiner points per edge to bound the error [27]. But, we utilize this information, and we only place $O(n \log c)$ Steiner points per edge, where c is a constant depending on the error and the geometry information of T (where $c \in [2, 5]$ on a terrain surface with 50k faces on average). Figure 5 (a) and Figure 5 (c) show an example of the placement of Steiner points in algorithm *FixSP* and our algorithm with the same error, respectively. Our experimental results show that for a terrain surface with 50k faces and $\epsilon = 0.1$, our algorithm just places 10 Steiner points per edge to find a rough path in 71s ($\approx$ 1.2 min), and finds a refined path in 2s, but the best-known algorithm [22], [27] places more than 600 Steiner points per edge to find the resulting path in 119,000s ($\approx$ 1.5 days).

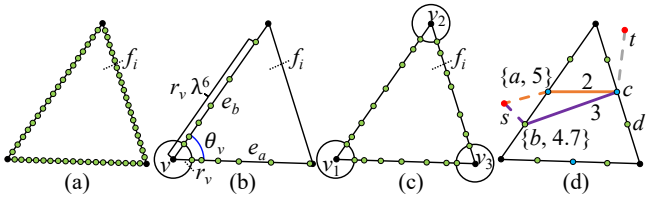

Fig. 5. (a) Steiner points on f_i in *FixSP*, (b) Steiner points on e_a and e_b in our algorithm, (c) Steiner points on f_i in our algorithm, and (d) pruned Dijkstra's algorithm

Algorithm [22] is originally designed for finding the unweighted shortest path, but it can be adapted for solving the weighted region problem in the following way. In the unweighted case, given a face f_i, and two points p and q on f_i, the weight of an edge in the graph (constructed by the Steiner point and V) 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. Then, algorithm [22] can solve the weighted region problem. It is the most recent and most efficient *on-the-fly* algorithm. Furthermore, the running time of algorithm [22] 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 $n' = O(n^3)$ [27], so the adapted running time of algorithm [22] is $O(n^3 \log n)$. The adapted algorithm [22] in the weighted case is essentially equivalent to algorithm [27].

B. Other related work

There are some other studies related to our problem although they are not exactly our problem.

(1) *On-the-fly algorithms on the unweighted terrain surface* [9], [14]: As mentioned in Section II-B1, algorithm [14] is recognized as the best-known exact on-the-fly algorithm for the unweighted shortest path calculation, but it cannot be adapted to the weighted case. Algorithm [9] uses a *finite element* method for unweighted shortest path calculation, but it cannot guarantee on the quality of the path returned with a given time limit and a given memory capacity. There is no known algorithm to adapt both algorithms [9], [14] to the

weighted case. (2) *Oracle-based algorithms on the terrain surface* [20], [35], [36]: The work [20] (resp. the work [35], [36]) builds an oracle on a weighted (resp. unweighted) terrain surface. The algorithm used for pre-computing the weighted shortest paths in the work [20] is algorithm *FixSP* [27], which runs in $O(mn\log(mn))$ times, where m is the number of Steiner points per edge, and $m = O(n^2)$ [27]. The algorithm used for pre-computing the unweighted shortest paths in the work [35], [36] is algorithm [22], which runs in $O((n+n')\log(n+n'))$ time. After we adapt algorithm [22] to the weighted case, the oracle [35], [36] can be adapted on a weighted terrain surface. But, we focus on the weighted *on-the-fly* shortest path algorithm in this paper. This is because the weighted *on-the-fly* shortest path algorithm is a fundamental operator of building an oracle [20], [35], [36] for the weighted region problem.

IV. Methodology

A. Overview

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

(1a) **The weighted graph**: It is used for the shortest path calculation in 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 that can be *Roug* or *Ref*. To construct G_A, we first define a set of Steiner points on each edge of E as SP_A, where A can be *Roug* or *Ref*. Let $G_A.V = SP_A \bigcup V$. 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 5 (d), (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 $G_{Ref}.E$.

(1b) **The removing value**: It is a constant (denoted by k and usually set to 2) used for calculating SP_{Roug}. In Figure 7 (b), we first place Steiner points according to SP_{Ref}, and then suppose that we move from v_1 to v_2. Whenever we encounter a Steiner point, we keep one and iteratively remove the next $k = 1$ point(s). We repeat it for all edges in E to obtain a set of remaining Steiner points, SP_{Roug}.

(1c) **The node information**: It is calculated in algorithm *Roug*, and is used to reduce the running time in algorithm *Ref*. In Dijkstra's algorithm, given a source node s, a set of nodes $G_A.V$ where A can be *Roug* or *Ref*, for each $u \in G_A.V$, we define $dist_A(u)$ to be the weighted shortest distance from s to u, define $prev_A(u)$ to be the previous node of u along the weighted shortest path from s to u. Give a source node s, after running algorithm *Roug*, *node information* stores $dist_{Roug}(u)$ and $prev_{Roug}(u)$ (based on s) for each $u \in G_{Roug}.V$. In Figure 5 (d), the numbers 5 and 4.7 next to a and b are the weighted shortest distances from s to a and b, the numbers 2 and 3 are the Euclidean distances of orange and purple edges. Suppose that the weight of this face is 1. After running algorithm *Roug*, the weighted shortest path from s to c is $s \to a \to c$, i.e., (1) the weighted shortest distance from s to c is $5 + 1 \times 2 = 7$ and (2) the previous node of c along this path is a, so we have $dist_{Roug}(c) = 7$ and $prev_{Roug}(c) = a$ as node information of c.

(1d) **The full or non-full edge sequence**: Given an edge sequence $S = ((v_1, v_1'), \ldots, (v_l, v_l')) = (e_1, \ldots, e_l)$, S is said to be a *full edge sequence* if the length of each edge in S is larger than 0. In Figure 6 (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 6 (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 (ϕ_2, ϕ_2), (ϕ_3, ϕ_3) and (ϕ_4, ϕ_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*.

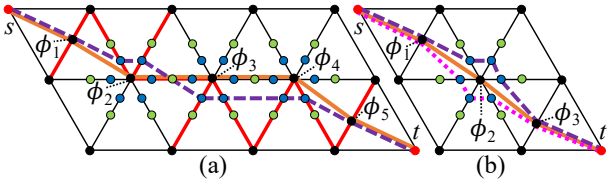

Fig. 6. (a) Successive endpoint case, and (b) single endpoint case

2) ***Overview of algorithm Roug****:* Given a terrain T, two vertices s and t in V, the error parameter ϵ, 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 (based on s) as node information. There are two steps:

- η **calculation**: In Figure 7 (b), given T, ϵ, and k, we first use ϵ and an efficient Steiner point placement scheme to get a set of Steiner points SP_{Ref}, and then use k to remove some Steiner points and get a set of remaining Steiner points SP_{Roug}, and then use SP_{Roug} to calculate $\eta\epsilon$.
- **Rough path calculation**: In Figure 7 (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}, and then 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 in this figure.

3) ***Overview of algorithm Ref****:* Given a terrain T, two vertices s and t in V, ϵ, 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 7 (d), given $\Pi_{Roug}(s,t)$ whose corresponding edge sequence is a non-full edge sequence (since the corresponding edge sequence

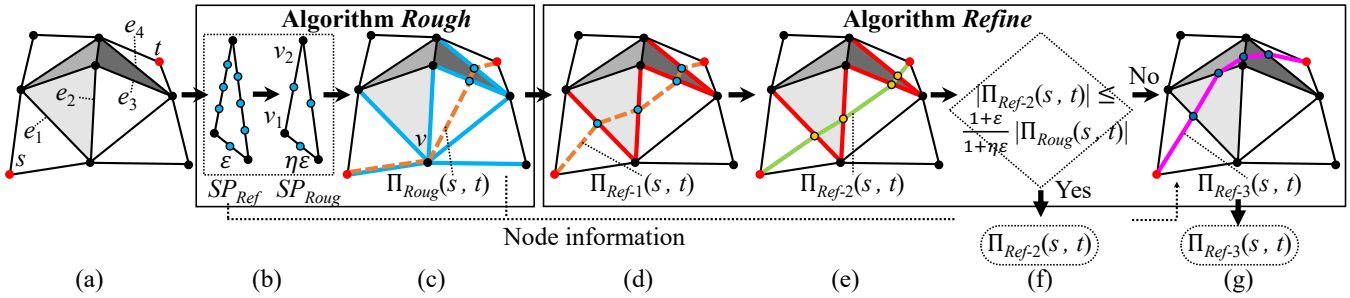

Fig. 7. Framework overview

of $\Pi_{Roug}(s,t)$ in Figure 7 (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. We denote the modified rough path as $\Pi_{Ref\text{-}1}(s,t)$, i.e., the orange dashed line in this figure, whose 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 7 (e), given T, s, t, and the edge sequence S (edges in red) of $\Pi_{Ref\text{-}1}(s,t)$ based on T, we use Snell's law and S to efficiently refine $\Pi_{Ref\text{-}1}(s,t)$. We denote the refined path on S as $\Pi_{Ref\text{-}2}(s,t)$, i.e., the green line in this figure.
- **Path checking**: In Figure 7 (f), given $\Pi_{Roug}(s,t)$, $\Pi_{Ref\text{-}2}(s,t)$, ϵ, and η, if $|\Pi_{Ref\text{-}2}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s,t)|$, we have $|\Pi_{Ref\text{-}2}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$ (since we can guarantee $|\Pi_{Roug}(s,t)| \leq (1+\eta\epsilon)|\Pi^*(s,t)|$ due to the error bound of Dijkstra's algorithm), and we can return $\Pi_{Ref\text{-}2}(s,t)$ as output. Otherwise, we need the next step.
- **Error guaranteed path refinement**: In Figure 7 (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, and then efficiently calculate a $(1+\epsilon)$-approximate weighted shortest path between s and t. We denote the refined path as $\Pi_{Ref\text{-}3}(s,t)$, i.e., the purple line in this figure. We can guarantee $|\Pi_{Ref\text{-}3}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$ due to the error bound of Dijkstra's algorithm.

*4) **Reasons of having algorithm Roug**:* (1) *Reduce the search area when using Snell's law*: Recall that the main reason why our algorithm is efficient is that we use Snell's law in algorithm *Ref*. Given two vertices s and t, we first obtain the edge sequence S passed by the rough path $\Pi_{Roug}(s,t)$ based on T, then apply Snell's law on S in algorithm *Ref*. 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 $\Pi_{Ref\text{-}2}(s,t)$ on S is a $(1+\epsilon)$-approximate weighted shortest path, so we need $\Pi_{Roug}(s,t)$ for assistance. In the path checking step, if $|\Pi_{Ref\text{-}2}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s,t)|$, we have $|\Pi_{Ref\text{-}2}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$ (since we can guarantee $|\Pi_{Roug}(s,t)| \leq (1+\eta\epsilon)|\Pi^*(s,t)|$), and we can terminate our algorithm by returning $\Pi_{Ref\text{-}2}(s,t)$ as output.

*5) **Reason of having algorithm Ref**:* In algorithm *Ref*, Snell's law can be used on S to efficiently refine the rough path to $\Pi_{Ref\text{-}2}(s,t)$, which is very close to the optimal weighted shortest path $\Pi^*(s,t)$, i.e., the distance between the intersection point of $\Pi_{Ref\text{-}2}(s,t)$ on each edge of S and that of $\Pi^*(s,t)$ is smaller than an error value δ (that is proportionally depending on ϵ), so there is no need to place many Steiner points on edges in E (recall the *no usage of Snell's law* drawback of the best-known algorithm *FixSP* [22], [27]).

B. Key Ideas in Algorithm Roug

We give the reason why algorithm *Roug* is efficient.

*1) **Efficient Steiner point placement scheme** (in the η 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 5 (c) 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 *same numbers of Steiner points per edge* drawback of the best-known algorithm *FixSP* [22], [27]) makes their algorithm very slow.

C. Key Ideas in Algorithm Ref

We give three reasons why algorithm *Ref* is efficient.

*1) **Novel full edge sequence conversion technique using progressive idea** (in the full edge sequence conversion 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 7 (c), if $\Pi_{Roug}(s,t)$ passes on a vertex v (the edge sequence S passed by $\Pi_{Roug}(s,t)$ is 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. That is, 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. However, we

do not need *all* such edges. Instead, we only need *some* of them. 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 7 (d).

Illustration: In Figure 6 (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 (ϕ_2, ϕ_2), (ϕ_3, ϕ_3), and $(\phi_4, h\phi_4)$. We then add more Steiner points and use this step *progressively* to find a new path segment, i.e., the purple dashed line between ϕ_1 and ϕ_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 that 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, in this case, we cannot guarantee the rough path to be a $(1+\eta\epsilon)$-approximate weighted shortest path.

*2) **Novel effective weight pruning technique** (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 4 (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 4 (a) and (b), we regard all the faces except the first face in $F(S)$ as one *effective face* (i.e., we regard f_1 and f_2 as one face $\triangle u_p p_1 q_1$), and use the ray in the purple line for calculating effective weight of $\triangle u_p p_1 q_1$. 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 $\triangle u_p p_1 q_1$. In Figure 4 (c), we find the ray starting from s and passing m_{ef} (i.e., the dark blue line). t is very close to this ray, 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 the new m_1 and the previous m_1 is smaller than δ. In Figure 4 (d), after processing e_1, we continue to process other edges in S, and we have the ray starting from ρ_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** (in the error guaranteed path refinement step, with the help of algorithm Roug):* In the path checking step, if $|\Pi_{\textit{Ref-2}}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{\textit{Roug}}(s,t)|$ (which occurs 99% of the time theoretically and 100% in our experiments), we do not need the error guaranteed path refinement step. Otherwise (i.e., only if the edge sequence 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 running time of algorithm *Roug-Ref* will not increase a lot.

Illustration: (1) In the η calculation step of algorithm *Roug*, we have calculated SP_{Ref} with error ϵ and SP_{Roug} with error $\eta\epsilon$. In Figure 7 (b), we first use ϵ and an efficient Steiner point placement scheme to get a set of Steiner points SP_{Ref}, and then use the remaining value k to remove some Steiner points and get a set of remaining Steiner points SP_{Roug}, and then use calculate SP_{Roug} to calculate $\eta\epsilon$, which implies that $SP_{Roug} \subseteq SP_{Ref}$ and $G_{Roug}.V \subseteq G_{Ref}.V$. (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*, we maintain a *priority queue* [15] in Dijkstra's algorithm (on the weighted graph G_{Ref}). For the priority queue, when we follow the path $s \to b \to c$ 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 that we do not insert a Steiner point into the queue in the error guaranteed path refinement step of 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 of 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 results show 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 k is larger, more Steiner points are removed, and η is larger, so the chance that $|\Pi_{\textit{Ref-2}}(s,t)| > \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{\textit{Roug}}(s,t)|$ 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 give implementation details of algorithm *Roug*. Recall the notations in Section II. We define ϵ' to be a different error parameter corresponding to ϵ, where $(2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = \epsilon$.

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, and 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. Let θ_v be the angle between any two edges of T that are incident to v. Figure 5 (b) shows an example of the polygonal cap C_v around v, with radius r_v, and the angle θ_v of v.

Detail: We introduce our efficient Steiner points placement scheme based on ϵ. 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 $a_1, a_2, \ldots, a_{\tau_a - 1}$ (resp. $b_1, b_2, \ldots b_{\tau_b - 1}$) 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 $2 \le j \le \tau_a - 1$ (resp. $2 \le k \le \tau_b - 1$). We repeat it on each edge of f_i. Figure 5 (b) and Figure 5 (c) show an example of Steiner points on e_a and e_b, and f_i based on ϵ, respectively. Knowing this, in Figure 7 (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

We give implementation details of algorithm *Ref*.

1) ***Full edge sequence conversion****:* Given a point v, v is said to be on the edge (resp. vertex) if it lies in the internal of an edge in E (resp. it lies on the vertex in V). In Figure 6 (a), ϕ_1 is on the edge, ϕ_2 is on the vertex.

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

(1.1) *Successive point*: Lines 4-14, see Figure 6 (a) with the orange path as input. Specifically, in lines 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 are on the vertex, so we store $v_s = \phi_1$ as the *start vertex*. In lines 6-7, $v_c = \phi_3$, $v_n = \phi_4$, $v_p = \phi_2$. In lines 8-14, $v_c = \phi_4$, $v_n = \phi_5$, $v_p = \phi_3$, we have found all the successive points, so we store $v_n = \phi_5$ as the *end vertex*. The blue points are the new Steiner points on E'. The purple dashed line between ϕ_1 and ϕ_5 represents $\Pi'_{Ref\text{-}1}(v_s, v_e)$, and the orange line between ϕ_1 and ϕ_5 represents $\Pi_{Roug}(v_s, v_e)$.

(1.2) *Single point*: Lines 15-24, see Figure 6 (b) with the orange path as input. $v_c = \phi_2$, $v_n = \phi_3$, $v_p = \phi_1$, we know only v_c is on the vertex. The blue points are new Steiner points. The orange, purple line-dashed, pink dot-dashed lines between ϕ_1 and ϕ_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 $\Pi_{Ref\text{-}2}(s,t) = (s, \rho_1, \ldots, \rho_l, t)$, where ρ_i for $i \in \{1, \ldots, l\}$ is a point on an edge in E. Given 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* $\Pi_c = (s, c_1, c_2, \ldots, c_g, R_g^c)$ starting from s, hitting c and following Snell's law on other edges in S, where $1 \le g \le l$, each c_i for $i \in \{1, \ldots, g\}$ is an intersection point in Π_c, and R_g^c is the last out-ray at $e_g \in S$. Figure 4 (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 $\Pi_{m^1} = (s, m_1^1, m_2^1, R_2^{m^1})$ (i.e., the purple line) that passes the whole S. If we can find ρ_1 on e_1 such that the 3D surface Snell's ray $\Pi_\rho = (s, \rho_1, \ldots, \rho_l, R_l^\rho)$ which hits t, then Π_ρ is the result of $\Pi_{Ref\text{-}2}(s,t)$. Recall that δ is a parameter used for controlling the error, we let $\delta = \frac{hew}{6lW}$.

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

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

1: $v_s \leftarrow$ *NULL*, $v_e \leftarrow$ *NULL*, $E' \leftarrow \emptyset$, $\Pi_{Ref\text{-}1}(s,t) \leftarrow \Pi_{Roug}(s,t)$
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: $v_c \leftarrow v$, $v_n \leftarrow v_c.next$, $v_p \leftarrow v_c.prev$
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 these new Steiner points and V
12: $\Pi'_{Ref\text{-}1}(v_s, v_e) \leftarrow$ *EdgSeqConv* ($\Pi'_{Roug}(v_s, v_e)$, ζ)
13: **if** $|\Pi'_{Ref\text{-}1}(v_s, v_e)| < |\Pi_{Roug}(v_s, v_e)|$ **then**
14: $\Pi_{Ref\text{-}1}(s,t) \leftarrow \Pi_{Ref\text{-}1}(s, v_s) \cup \Pi'_{Ref\text{-}1}(v_s, v_e) \cup \Pi_{Ref\text{-}1}(v_e, t)$
15: **else if** both v_n and v_n are on the edge **then**
16: **for** $i \leftarrow 1$ to ζ **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: $\Pi_{Ref\text{-}1}(s,t) \leftarrow \Pi_{Ref\text{-}1}(s, v_p) \cup \Pi_l(v_p, v_n) \cup \Pi_{Ref\text{-}1}(v_n, t)$
21: **break**
22: **else if** $|\Pi_r(v_p, v_n)| < |\Pi_{Roug}(v_p, v_n)|$ **then**
23: $\Pi_{Ref\text{-}1}(s,t) \leftarrow \Pi_{Ref\text{-}1}(s, v_p) \cup \Pi_r(v_p, v_n) \cup \Pi_{Ref\text{-}1}(v_n, t)$
24: **break**
25: $v_c \leftarrow v_n$, $v_n \leftarrow v_c.next$, $v_p \leftarrow v_c.prev$
26: **return** $\Pi_{Ref\text{-}1}(s,t)$

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

(2.1) *Binary search initial path finding*: Lines 6-20, see Figure 4 (a). Specifically, in lines 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 $[a_1, b_1] = [p_1, m_1]$. In lines 13-20, we then calculate m_1^1, the purple line $\Pi_{m^1} = (s, m_1^1, m_2^1, R_2^{m^1})$ passes the whole S, and t is on the right side of $R_2^{m^1}$, 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*: Lines 21-29. It is the first time for the purple line $\Pi_{m^1} = (s, m_1^1, m_2^1, R_2^{m^1})$ passing the whole S based on T, so we can use effective weight pruning on e_i. Specifically, in line 22 and Figure 4 (a), u is the intersection point between the purple line $R_2^{m^1}$ and the left edge $\overline{p_1 t}$ of f_2 that adjacent to t. In lines 23-24 and Figure 4 (b), u_p is the projected point of u, and $f_{ef} = \triangle u_p p_1 q_1$ is an effective face for f_1 and f_2. In line 25 and Figure 4 (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}. In lines 26-27 and Figure 4 (b), t_p is the projected point of t, we set m_{ef} to be unknown and use Snell's law in vector form [6], then build a quartic equation

Algorithm 2 *SneLawRef* (s, t, δ, S)

Input: source s, destination t, user parameter δ, and edge sequence S
Output: the refined path $\Pi_{Ref\text{-}2}(s,t)$
1: $\Pi_{Ref\text{-}2}(s,t) \leftarrow \{s\}$, *root* $\leftarrow s$
2: **for** each $e_i \in S$ with $i \leftarrow 1$ to $|S|$ **do**
3: $a_i \leftarrow e_i$ left endpoint, $b_i \leftarrow e_i$ right endpoint, $[a_i, b_i] \leftarrow$ an interval
4: **for** each $e_i \in S$ with $i \leftarrow 1$ to $|S|$ **do**
5: **while** $|a_i b_i| \geq \delta$ **do**
6: $m_i \leftarrow$ midpoint of $[a_i, b_i]$
7: compute the 3D surface Snell's ray with $\Pi_m = (root, m_i, m_{i+1}, \ldots, m_g, R_g^m)$ with $g \leq l$
8: **if** Π_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: $[a_i, b_i] \leftarrow [a_i, m_i], \ldots, [a_g, b_g] \leftarrow [a_g, m_g]$
11: **else if** e_{g+1} is on the right side of R_g^m **then**
12: $[a_i, b_i] \leftarrow [m_i, b_i], \ldots, [a_g, b_g] \leftarrow [m_g, b_g]$
13: **else if** Π_m passes the whole S, i.e., $g = l$ **then**
14: **if** t is on R_g^m **then**
15: $\Pi_{Ref\text{-}2}(s,t) \leftarrow \Pi_{Ref\text{-}2}(s,t) \cup \{m_i, m_{i+1}, \ldots, m_g, t\}$
16: **return** $\Pi_{Ref\text{-}2}(s,t)$
17: **else if** t is on the left side of R_g^m **then**
18: $[a_i, b_i] \leftarrow [a_i, m_i], \ldots, [a_g, b_g] \leftarrow [a_g, m_g]$
19: **else if** t is on the right side of R_g^m **then**
20: $[a_i, b_i] \leftarrow [m_i, b_i], \ldots, [a_g, b_g] \leftarrow [m_g, b_g]$
21: **if** have not used effective weight pruning on e_i **then**
22: $u \leftarrow$ the intersection point between R_l^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 $\overline{sm_i}$, $\overline{m_i u_p}$, 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 $\Pi_m = (root, m_i, \ldots, m_g, R_g^m)$
29: update $[a_i, b_i], \ldots, [a_g, b_g]$ same as in lines 14-20
30: $\rho_i \leftarrow [a_i, b_i]$ midpoint, $\Pi_{Ref\text{-}2}(s,t) \leftarrow \Pi_{Ref\text{-}2}(s,t) \cup \{\rho_i\}$, *root* $\leftarrow \rho_i$
31: $\Pi_{Ref\text{-}2}(s,t) \leftarrow \Pi_{Ref\text{-}2}(s,t) \cup \{t\}$
32: **return** $\Pi_{Ref\text{-}2}(s,t)$

Algorithm 3 *ErrGuarRef* (s, t, ϵ, *NodeInfo*)

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

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 [30] 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. In lines 28-29 and Figure 4 (c), we compute the dark blue line $\Pi_{m^2} = (s, m_{ef}, m_2^2, R_2^{m^2})$. Since t is on the left side of $R_2^{m^1}$, 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*: Lines 2, 6-20, 30-31, see Figure 4 (d). Specifically, in line 6-20, we perform binary search until $|a_1 b_1| = |m_1^1 m_{ef}| < \delta$. In line 30, ρ_1 is the midpoint of $[a_1, b_1] = [m_1^1, m_{ef}]$, and we have the pink dashed line $\Pi_{Ref\text{-}2}(s,t) = (s, \rho_1)$, and *root* $= \rho_1$. In line 2, we then iterate the procedure to obtain 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 $\Pi_{Ref\text{-}2}(s,t) = (s, \rho_1, \rho_2, t)$.

3) ***Error guaranteed path refinement****:* We define $Q = \{\{u_1, dist_{Ref}(u_1)\}, \{u_2, dist_{Ref}(u_2)\}, \ldots\}$ to be a priority queue that stores a set of nodes $u_i \in G_{Ref}.V$ waiting for processing. In Figure 5 (d), 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 5 (d)).

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

(3.2) *Priority queue looping*: Lines 7-15. Specifically, in lines 8-9, suppose Q stores $\{\{a, 5\}, \{b, 4.7\}\}$, we dequeue b with shortest distance value 4.7. In lines 12-15, one adjacent node of b is c, with the node information, we know $dist_{Ref}(c) = 7$, so $7 < dist_{Ref}(b) + w_{bc} \cdot d(b, c) = 4.7 + 1 \times 3 = 7.7$, there is no need to enqueue $\{c, dist_{Ref}(c) = 7.7\}$ into Q, which saves the time. But, without the node information, we just know $dist_{Ref}(c) = \infty$, and we need to enqueue $\{c, dist_{Ref}(c) = 7.7\}$ into Q, which increases the time.

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

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. (1) The *running time* contains (1a) $O(n \log n)$ for *Roug* due to Dijkstra's algorithm on G_{Roug} with n nodes, (1b) $O(n \log n)$ for the full edge sequence conversion step in *Ref* due to Dijkstra's algorithm, (1c) $O(l)$ 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 (1d) $O(n \log n)$ for the error guaranteed path refinement step in *Ref* due to Dijkstra's algorithm on G_{Ref} with n nodes and the node information. (2) The *memory usage* contains $O(n)$ for *Roug* due to (2a) Dijkstra's algorithm on G_{Roug} with n nodes, (2b) $O(n)$ for the full edge sequence conversion step in *Ref* due to Dijkstra's algorithm, (2c) $O(l)$ for the Snell's law path refinement step in *Ref* due to l edges in S, and (2d) $O(n)$ for the error guaranteed path refinement step in *Ref* due to Dijkstra's algorithm on G_{Ref} with n nodes. (3) For the

TABLE I
DATASETS

Name	$\|F\|$
Original dataset	
BearHead (*BH*) [3], [35], [36]	280k
EaglePeak (*EP*) [3], [35], [36]	300k
SeaBed (*SB*) [10]	2k
CyberPunk (*CP*) [2]	2k
PathAdvisor (*PA*) [39]	1k
Small-version dataset	
BH small-version (*BH-small*)	3k
EP small-version (*EP-small*)	3k
Multi-resolution dataset	
Multi-resolution of *EP*	1M, 2M, 3M, 4M, 5M
Multi-resolution of *EP-small*	10k, 20k, 30k, 40k, 50k

TABLE II
ALGORITHM COMPARISONS

Algorithm	Time	Size	Error
ConWave [34]	Gigantic	Large	$(1+\epsilon)$
UnfixSP [8]	Large	Large	$(1+\epsilon)$
FixSP [20], [22], [27]	Large	Large	$(1+\epsilon)$
LogSP	Medium	Medium	$(1+\epsilon)$
***Roug-Ref* (ours)**	**Small**	**Small**	$\mathbf{(1+\epsilon)}$

error bound, in the rough path calculation step, we guarantee $|\Pi_{Roug}(s,t)| \leq (1+\eta\epsilon)|\Pi^*(s,t)|$ (due to the error bound of Dijkstra's algorithm, see Theorem 3.1 of work [26]). So in the path checking step, (3a) if $|\Pi_{Ref\text{-}2}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s,t)|$, we guarantee $|\Pi_{Ref\text{-}2}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$; (3b) if not, we have the error guaranteed path refinement step, which guarantees $|\Pi_{Ref\text{-}3}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$ (due to the error bound of Dijkstra's algorithm, see Theorem 3.1 of work [26]). The detailed proof appears in the appendix. □

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 studies [22], [23], [32], [35], [36].

1) ***Datasets****:* Following existing studies [20], [22], [36], we conducted our experiment on 17 real terrain datasets in Table I. For *BH-small* and *EP-small* datasets, we generated them using *BH* and *EP* following the procedure in [32], [35], [36] (which creates a terrain surface with different resolutions). We generated multi-resolution of *EP* and *EP-small* datasets for scalability test. Following the work [20], we set the weight of a face in terrain datasets to be the slope of that face.

2) ***Algorithms****:* Algorithm *FixSP* [22], [27] (i.e., also the algorithm used for pre-computing the weighted shortest paths in the oracle [20]) in the *Steiner point* approach is regarded as the best-known baseline. 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. In Table II, we compare these two algorithms, together with algorithm *ConWave* [34], *UnfixedSP* [8], and our algorithm. *Roug-Ref* is the best algorithm since it has the smallest query time and memory usage.

But, we do not include algorithm *ConWave* [34] in the experiment, due to its large running time. There is also no implementation of it so far [26]. We do not include two algorithms [25], [37] in *heuristics* approach in the experiment, because they cannot guarantee on the quality of the path returned with a given time limit and a given memory capacity. We do not include algorithm *UnfixedSP* [8] in the experiment since its running time is 10^3 larger than algorithm *FixedSP* (as shown by the work [20]). We do not include *on-the-fly algorithms on the unweighted terrain surface* [9], [14] in the experiment since they cannot be adapted to the weighted case. We do not include *oracle-based algorithms* [20], [35], [36] in the experiment since we focus on *on-the-fly* algorithm in this paper, and we have included the algorithm used for pre-computing the weighted shortest paths in oracle [20], i.e., algorithm *FixSP*, in the experiment. In general, we compared our algorithm *Roug-Ref*, and the baseline algorithms, i.e., *FixSP* [20], [22], [27], and *LogSP* in the experiment. As we will discuss later, we also compared variations of *Roug-Ref* for the ablation study. The comparisons of all algorithms and their theoretical analysis can be found in the appendix.

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

4) ***Factors & Measurements****:* We studied three factors, namely (1) k (i.e., the removing value), (2) ϵ (i.e., the error parameter), 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., *Roug*, (2b) *query time for the second algorithm*, i.e., *Ref*, (2c) *query time for algorithm Roug-Ref*, (3a) *memory usage for the first algorithm* i.e., *Roug*, (3b) *memory usage for the second algorithm* i.e., *Ref*, (3c) *memory usage for algorithm Roug-Ref*, and (4) *distance error* (i.e., the error of the distance returned by the algorithm compared with the weighted shortest distance when $\epsilon = 0.05$, since no algorithm can solve the weighted region problem exactly so far).

B. Experimental Results

In this section, we provide (1) the ablation study of *Roug-Ref*, (2) the comparison of *Roug-Ref* with other baselines, (3) the scalability test of *Roug-Ref*, (4) the user study, (5) the motivation study, and (6) the experimental results summary. We use *FixSP* and set $\epsilon = 0.05$ 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 due to its expensive running time, the distance error is omitted on these datasets. Due to the same reason, we (1) compared all algorithms on datasets with less than 250k faces, and (2) compared algorithms not involving *FixSP* on datasets with more than 250k faces. For the (1) total query time, (2) query time for algorithm *Roug* and (3) query

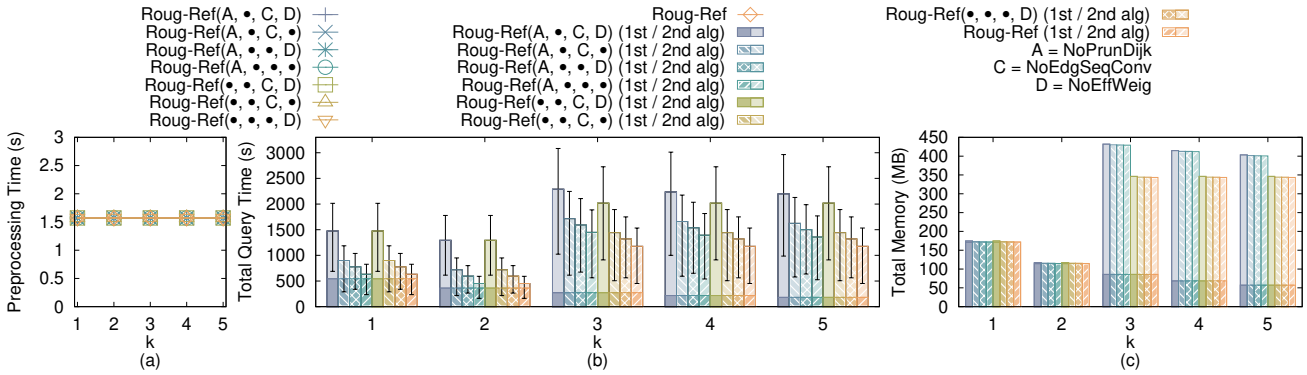

Fig. 8. Ablation study (effect of k on *EP* dataset)

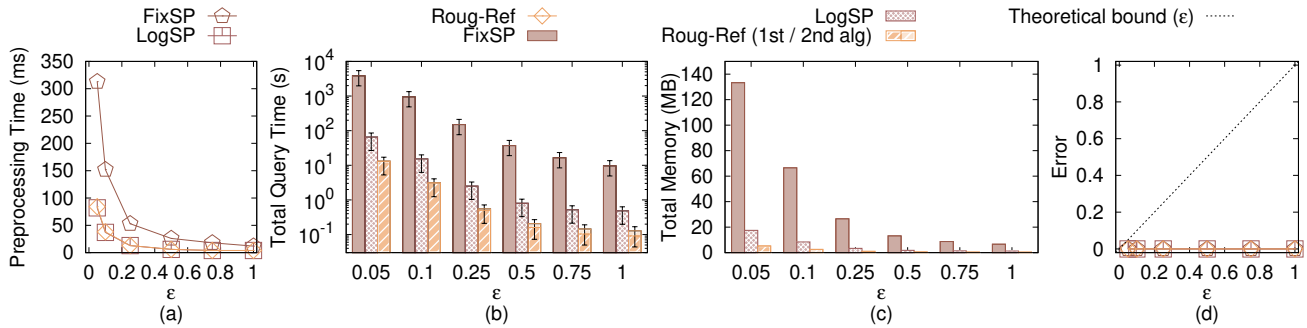

Fig. 9. Baseline comparisons (effect of ϵ on *BH-small* dataset)

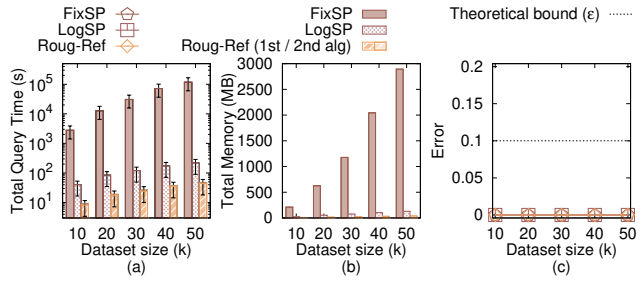

Fig. 10. Baseline comparisons (effect of dataset size on multi-resolution of *EP-small* datasets)

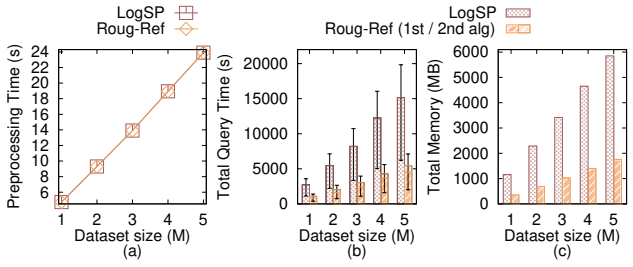

Fig. 11. Scalability test (on multi-resolution of *EP* datasets)

time for algorithm *Ref*, the vertical bar means the minimum and maximum results.

1) ***Ablation study of Roug-Ref****:* In our algorithm *Roug-Ref*, we hare 15 variations: (1) we do not use the node information for pruning in the error guaranteed path refinement step, (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, and (4) we remove the effective weight pruning out sub-step in the Snell's law path refinement step, for ablation study (they correspond to four techniques in Section IV-B and IV-C). 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*), where $\{A = NoPrunDijk, B = FixSP, C = NoEdgSeqConv, D = NoEffWeig\}$, and • means keeping the same component as in algorithm *Roug-Ref*. Since k will only affect the 15 variations of *Roug-Ref* and our algorithm *Roug-Ref*, we study the effect of k in this subsection.

Effect of *k*: In Figure 8, we tested 5 values of k from $\{1, 2, 3, 4, 5\}$ on *EP* dataset by setting ϵ to be 0.25 for ablation study (involving variations of *Roug-Ref*). For the sake of illustration, we compare the variations of *Roug-Ref* without *B* = *FixSP* component, because the query time of the variations of *Roug-Ref* with *FixSP* component is always larger than that of the variations of *Roug-Ref* without *FixSP* component. The overlapping lines of the preprocessing time mean the result is the same, since the variation of algorithms will not affect the preprocessing time. When k is varied, it may happen that the error guaranteed path refinement step in algorithm *Ref* is used, so the variations of *Roug-Ref* with or without *NoPrunDijk* component will have different effects on the measurements. We compared variations of *Roug-Ref* involves both with and

without *NoPrunDijk* component for completeness. When k is 1 or 2, the query time for *Roug-Ref* is the same as that of *Roug-Ref*(*NoPrunDijk*, •, •, •) since the error guaranteed path refinement step is not used. But, when k is larger than 2, the former one's query time is smaller. By setting k to be 2, the query time and memory usage of the algorithms using *Roug-Ref* framework are the smallest. This is because when k 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 k, i.e., 2. When k is larger than 2 (e.g., $k = 3$), the query time of *Roug-Ref* is 1330s, and the query time of *LogSP* (i.e., without the *Roug-Ref* framework) is 1300s. This shows that even if we select a value of k that is not the optimal value, the query time of *Roug-Ref* will not increase a lot, which implies the usefulness of the node information for pruning out in Dijkstra's algorithm in the error guaranteed path refinement step of *Ref*. In the following subsections, we set $k = 2$, and show that *Roug-Ref* can perform much better than baselines in terms of query time and memory usage.

2) ***Baseline comparisons****:* From the previous subsection, we know that *Roug-Ref* is the best algorithm among different variations. Starting from this subsection, we compare different baselines with *Roug-Ref*. Since ϵ and the dataset size will both affect the baselines and our algorithm *Roug-Ref*, we study the effect of ϵ and the dataset size in this subsection.

Effect of ϵ: In Figure 9, we tested 6 values of ϵ from $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on *BH-small* dataset by setting k to be 2. *Roug-Ref* performs better than *FixSP* and *LogSP* in terms of query time, and it is clearer to observe the superior performance of *Roug-Ref* when $\epsilon = 1$. In addition, when $\epsilon = 0.05$, the query time of *Roug-Ref* is only 14.6s, but the query time of *FixSP* is 23,800s ($\approx$ 7.2 hours), i.e., 1630 times larger than that of *Roug-Ref*. *LogSP* still performs badly in terms of query time and memory usage. This is because *LogSP* does not utilize Snell's law, and do not use the rough-refine idea. The error of all algorithms is close to 0%.

Effect of dataset size: In Figure 10, we tested 5 values of dataset size from {10k, 20k, 30k, 40k, 50k} on multi-resolution of *EP-small* datasets by setting ϵ to be 0.1 and k to be 2. *Roug-Ref* performs better than *FixSP* and *LogSP*. When the dataset size is 50k, *Roug-Ref*'s query time is 10^3 times and 6 times smaller than that of *FixSP* and *LogSP*, respectively.

3) ***Scalability test****:* In Figure 11, we tested 5 values of dataset size from {1M, 2M, 3M, 4M, 5M} on multi-resolution of *EP* datasets by setting ϵ to be 0.25 and k to be 2. *Roug-Ref* can still beat *LogSP* in terms of the query time and memory usage. When the dataset size is 5M, *Roug-Ref*'s query time is still reasonable. However, *FixSP*'s query time is larger than 7 days, so it is not included in the figure.

4) ***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* [39]. 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 [39]. We chose two places in Path Advisor as source and destination, and repeated it for 100 times to calculate the path (with $\epsilon = 0.5$). In Figure 1 (b), 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 1 (b) and the path distance result to 30 users (i.e., university students), and 96.7% of users think the blue path is the more realistic since it is not close to the obstacle and does not have sudden direction changes. The average query time for the state-of-the-art algorithm *FixSP* and our algorithm are 16.62s and 0.1s, respectively.

5) ***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 [10]. 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 [31], 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) [31]. We randomly selected two points as source and destination, respectively, and repeated it 100 times to calculate the path (with $\epsilon = 0.5$). In Figure 1 (c), 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 [24], when constructing a cable that will be used for 100 years [31], 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 are 22.50s and 0.2s, respectively.

6) ***Summary****:* Our algorithm *Roug-Ref* consistently outperforms the state-of-the-art algorithm, i.e., *FixSP*, in terms of query time and memory usage. 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 query time is 73s ($\approx$ 1.2 min), and memory usage is 43MB, but the state-of-the-art algorithm's query time is 119,000s ($\approx$ 1.5 days), and 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] "Cyberpunk 2077," 2023. [Online]. Available: https://www.cyberpunk.net

[2] "Cyberpunk 2077 map," 2023. [Online]. Available: https://www.videogamecartography.com/2019/08/cyberpunk-2077-map.html

[3] "Data geocomm," 2023. [Online]. Available: http://data.geocomm.com/

[4] "Google earth," 2023. [Online]. Available: https://earth.google.com/web

[5] "Metaverse," 2023. [Online]. Available: https://about.facebook.com/meta

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

[7] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack, "An ε-approximation algorithm for weighted shortest paths on polyhedral surfaces," in *Scandinavian Workshop on Algorithm Theory*. Springer, 1998, pp. 11–22.

[8] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, "Determining approximate shortest paths on weighted polyhedral surfaces," *Journal of the ACM (JACM)*, vol. 52, no. 1, pp. 25–53, 2005.

[9] G. Altintas, "Finding shortest path on a terrain surface by using finite element method," *arXiv preprint arXiv:2201.06957*, 2022.

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

[11] 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.

[12] R. Blandford and R. Narayan, "Fermat's principle, caustics, and the classification of gravitational lens images," *Astrophysical Journal, Part 1 (ISSN 0004-637X), vol. 310, Nov. 15, 1986, p. 568-582.*, vol. 310, pp. 568–582, 1986.

[13] 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.

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

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

[16] 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.

[17] 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.

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

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

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

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

[28] 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.

[29] 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.

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

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

[32] 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.

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

[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] 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.

[36] 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.

[37] E. K. Xidias, "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.

[38] 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.

[39] 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.

[40] 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 REFMENT STEP OF ALGORITHM *Ref*

We use a 1D example to illustrate why the effective weight pruning sub-step in the Snell's law path refinement step of algorithm *Ref* can prune out unnecessary checking. Let 0 and 100 to be the position of the two endpoints of e_1, and we have $[a_1 b_1] = [0, 100]$. Assume that the position of the optimal 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 $[50, 100]$, $[75, 100]$, $[75, 87.5]$, $[81.25, 87.5]$, $[84.375, 87.5]$, $[85.9375, 87.5]$, $[86.71875, 87.5]$, $[87.109375, 87.5] \cdots$. But, with the effective weight pruning sub-step in the Snell's law path refinement step of algorithm *Ref*, assume that we still need to use several iterations of the original binary search to let Π_m pass the whole S based on T, and we need

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

to check $[50, 100]$, $[75, 100]$, $[75, 87.5]$. After checking the interval $[75, 87.5]$, we get a Π_m that passes the whole S based on T. Assume we calculate m_{ef} as 87 using effective weight pruning step. As a result, in the next checking, we can directly limit the searching interval to be $[87, 87.5]$, which can prune out some unnecessary interval checking including $[81.25, 87.5]$, $[84.375, 87.5]$, $[85.9375, 87.5]$, $[86.71875, 87.5]$, and thus, the effective weight pruning sub-step in the Snell's law path refinement step of algorithm *Ref* can save the running time and memory usage.

APPENDIX D
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, and v_3, we define v *belongs to* f if the area of $f = \triangle v_1v_2v_3$ is equal to the sum of the area of $\triangle vv_1v_2$, $\triangle vv_1v_3$ and $\triangle vv_1v_2v_3$. 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 12 and Figure 13, (2) Figure 14 and Figure 15 show the result on the *BH-small* dataset when varying ϵ and removing value, respectively. (3) Figure 16 and Figure 17, (4) Figure 18 and Figure 19 show the result on the *BH* dataset when varying ϵ and removing value, respectively. (5) Figure 20 and Figure 21, (6) Figure 22 and Figure 23 show the result on the *EP-small* dataset when varying ϵ and removing value, respectively. (7) Figure 24 and Figure 25, (8) Figure 26 and Figure 27 show the result on the *EP-small* dataset when varying ϵ and removing value, respectively. (9) Figure 28 and Figure 29 show the result on multi-resolution of *EP-small* datasets when varying dataset size. (10) Figure 30 and Figure 31 show the result on multi-resolution of *EP* datasets when varying dataset size.

Effect of ϵ. In Figure 12, Figure 16, Figure 20 and Figure 24, we tested 6 values of ϵ from $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting removing value to be 2. Figure 13, Figure 17, Figure 21 and Figure 25 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. The error of all algorithms is close to 0%.

Effect of removing value. In Figure 14, Figure 18, Figure 22 and Figure 26, we tested 5 values of removing value from $\{1, 2, 3, 4, 5\}$ on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting ϵ to be 0.1 on *BH-small* and *EP-small* datasets, and 0.25 on *BH* and *EP* datasets. Figure 15, Figure 19, Figure 23 and Figure 27 are the separated query time and memory usage in two steps for these results. By setting the removing value to be 2, the total query time and total memory usage of 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.

TABLE IV
COMPARISON OF ALL ALGORITHMS, WHERE A = *NoPrunDijk*, B = *FixSP*, C = *NoEdgSeqConv*, D = *NoEffWeig*

Algorithm	Time		Size		Error
ConWave [34]	$O(n^8 \log(\frac{lNWL}{w\epsilon}))$	Gigantic	$O(n^4)$	Large	$(1+\epsilon)$
FixSP [20], [22], [27]	$O(n^3 \log n)$	Gigantic	$O(n^3)$	Large	$(1+\epsilon)$
LogSP	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	Large	$O(n \log \frac{LN}{\epsilon})$	Medium	$(1+\epsilon)$
Roug-Ref(A, B, C, D)	$O(n^3 \log n + nl^2 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n^3 + nl)$	Medium	$(1+\epsilon)$
Roug-Ref(A, B, C, •)	$O(n^3 \log n + nl)$	Large	$O(n^3 + nl)$	Medium	$(1+\epsilon)$
Roug-Ref(A, B, •, D)	$O(n^3 \log n + l^2 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n^3 + l)$	Medium	$(1+\epsilon)$
Roug-Ref(A, B, •, •)	$O(n^3 \log n + l)$	Large	$O(n^3 + l)$	Medium	$(1+\epsilon)$
Roug-Ref(A, •, C, D)	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + nl^2 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n \log \frac{LN}{\epsilon} + nl)$	Medium	$(1+\epsilon)$
Roug-Ref(A, •, C, •)	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + nl)$	Large	$O(n \log \frac{LN}{\epsilon} + nl)$	Medium	$(1+\epsilon)$
Roug-Ref(A, •, •, D)	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l^2 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n \log \frac{LN}{\epsilon} + l)$	Medium	$(1+\epsilon)$
Roug-Ref(A, •, •, •)	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$	Large	$O(n \log \frac{LN}{\epsilon} + l)$	Medium	$(1+\epsilon)$
Roug-Ref(•, B, C, D)	$O(n^2 \log n + nl^2 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n^2 + nl)$	Medium	$(1+\epsilon)$
Roug-Ref(•, B, C, •)	$O(n^2 \log n + nl)$	Large	$O(n^2 + nl)$	Medium	$(1+\epsilon)$
Roug-Ref(•, B, •, D)	$O(n^2 \log n + l^2 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n^2 + l)$	Medium	$(1+\epsilon)$
Roug-Ref(•, B, •, •)	$O(n^2 \log n + l)$	Large	$O(n^2 + l)$	Medium	$(1+\epsilon)$
Roug-Ref(•, •, C, D)	$O(n \log n + nl^2 \log(\frac{lNWL}{w\epsilon}))$	Medium	$O(nl)$	Small	$(1+\epsilon)$
Roug-Ref(•, •, C, •)	$O(n \log n + nl)$	Medium	$O(nl)$	Small	$(1+\epsilon)$
Roug-Ref(•, •, •, D)	$O(n \log n + l^2 \log(\frac{lNWL}{w\epsilon}))$	Medium	$O(n+l)$	Small	$(1+\epsilon)$
***Roug-Ref* (ours)**	$\boldsymbol{O(n \log n + l)}$	**Small**	$\boldsymbol{O(n+l)}$	**Small**	$\boldsymbol{(1+\epsilon)}$

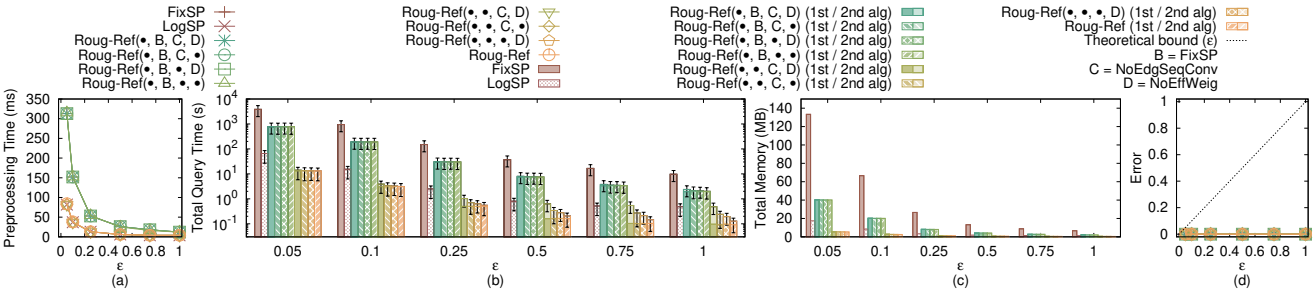

Fig. 12. Effect of ϵ on *BH-small* dataset (V2V path query)

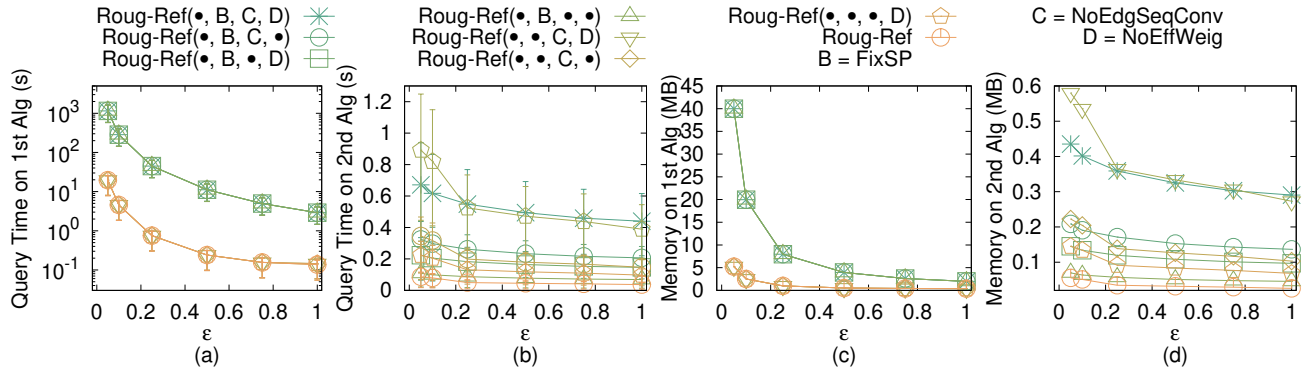

Fig. 13. Effect of ϵ on *BH-small* dataset with separated query time and memory usage in two steps (V2V path query)

Effect of dataset size (scalability test). In Figure 28 and Figure 30, we tested 5 values of dataset size from {10k, 20k, 30k, 40k, 50k} on multi-resolution of *EP-small* datasets by setting ϵ to be 0.1, and {1M, 2M, 3M, 4M, 5M} on multi-resolution of *EP* datasets by setting ϵ to be 0.25 for scalability test. Figure 29 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. 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 ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} and setting removing value to be 2on *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

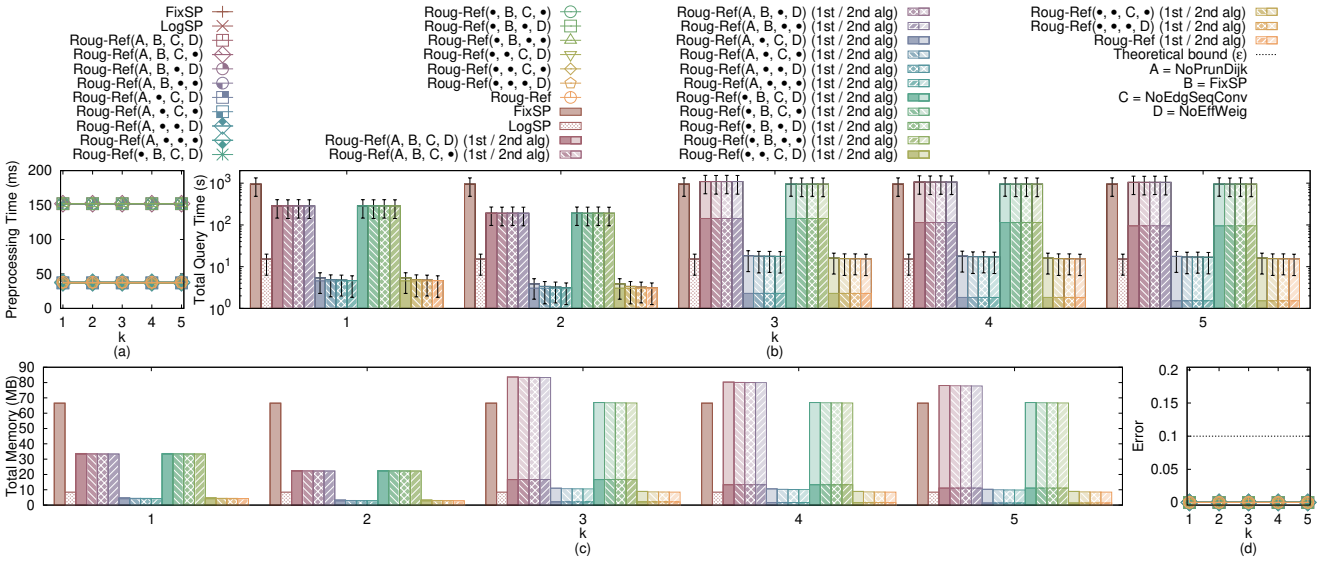

Fig. 14. Effect of removing value on *BH-small* dataset (V2V path query)

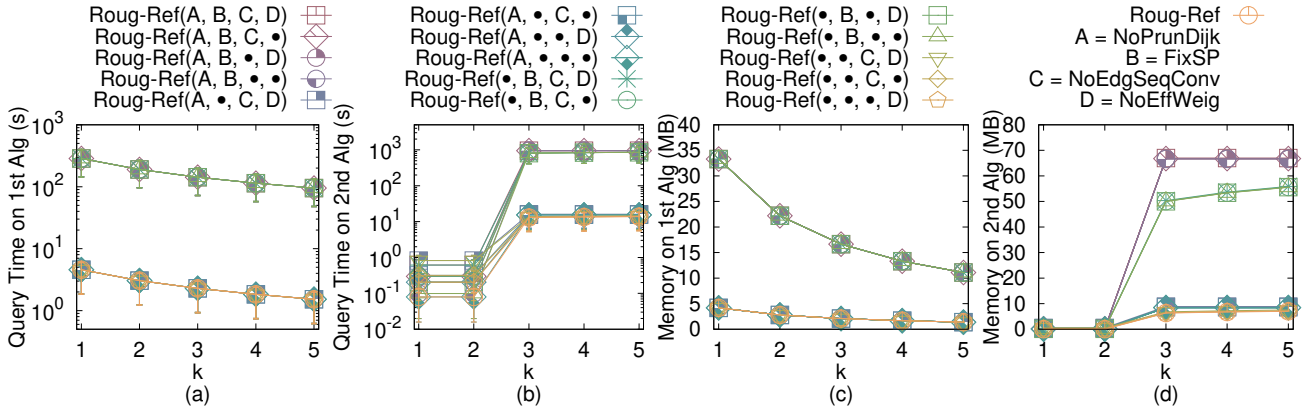

Fig. 15. Effect of removing value on *BH-small* dataset with separated query time and memory usage in two steps (V2V path query)

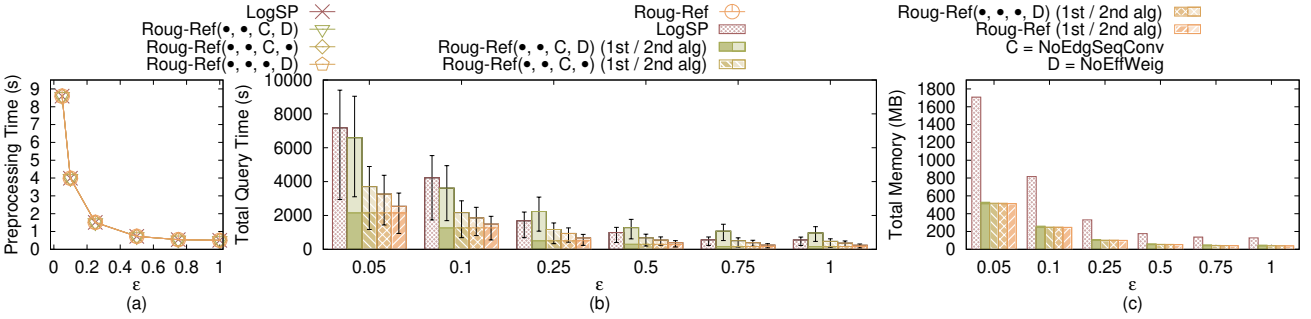

Fig. 16. Effect of ϵ on *BH* dataset (V2V path query)

all the remaining algorithms in terms of query time. 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 ϵ. Our user study in Section V-B4 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-B4 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.

D. Experimental Results on User Study (Cyberpunk 2077)

We conducted another user study on Cyberpunk 2077 [1], a popular 3D computer game. The dataset is *CP* dataset [2] 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

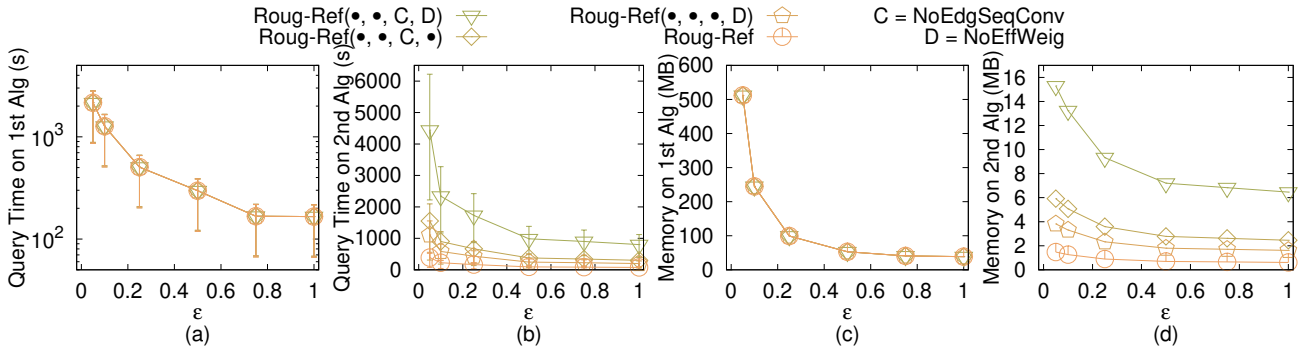

Fig. 17. Effect of ϵ on *BH* dataset with separated query time and memory usage in two steps (V2V path query)

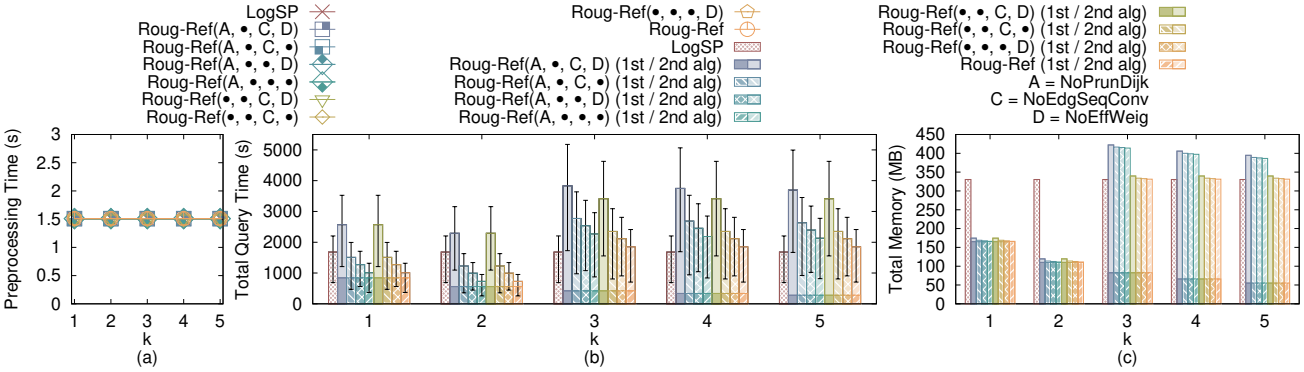

Fig. 18. Effect of removing value on *BH* dataset (V2V path query)

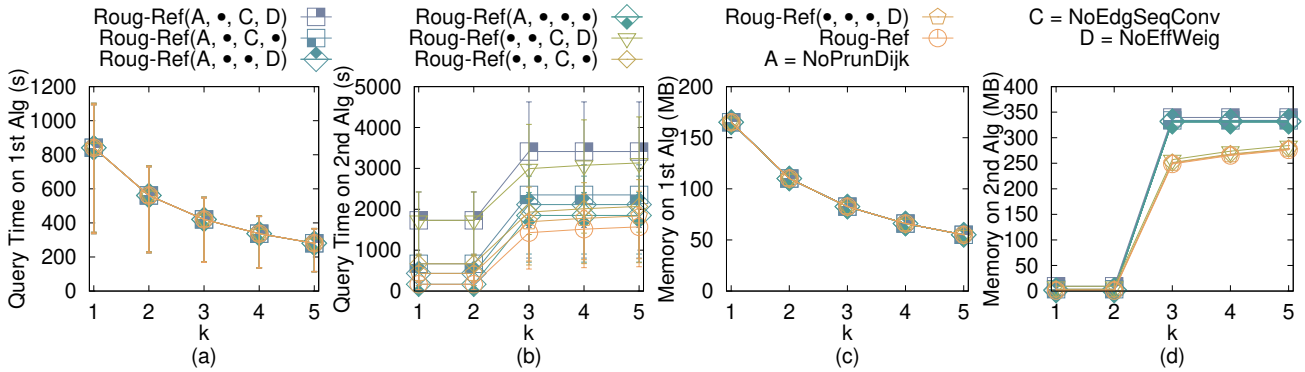

Fig. 19. Effect of removing value on *BH* dataset with separated query time and memory usage in two steps (V2V path query)

to calculate the path. Figure 38 and Figure 39 show the result for Cyberpunk 2077 user study when varying ϵ. When $\epsilon = 0.5$, the average query time for the state-of-the-art algorithm *FixSP* and our algorithm *Roug-Ref* are 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 ϵ. Our motivation study in Section V-B5 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 memeory usage.

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 [32], [35], [36]. Let $T_t = (V_t, E_t, F_t)$ be our target terrain that we want to generate with ex_t edges along x-coordinate, ey_t edges along y-coordinate and dataset size of DS_t, where $DS_t = 2 \cdot ex_t \cdot ey_t$. Let $T_o = (V_o, E_o, F_o)$ be the original terrain that we currently have with ex_o edges along x-coordinate, ey_o edges along y-coordinate and dataset size of DS_o, where $DS_o = 2 \cdot ex_o \cdot ey_o$. We then generate $(ex_t + 1) \cdot (ey_t + 1)$ 2D points (x, y) based on a Normal distribution $N(\mu_N, \sigma_N^2)$, where $\mu_N = (\overline{x} = \frac{\sum_{v_o \in V_o} x_{v_o}}{(ex_o+1)\cdot(ey_o+1)}, \overline{y} = \frac{\sum_{v_o \in V_o} y_{v_o}}{(ex_o+1)\cdot(ey_o+1)})$ and $\sigma_N^2 = (\frac{\sum_{v_o \in V_o} (x_{v_o} - \overline{x})^2}{(ex_o+1)\cdot(ey_o+1)}, \frac{\sum_{v_o \in V_o} (y_{v_o} - \overline{y})^2}{(ex_o+1)\cdot(ey_o+1)})$. In the end, we project each generated point (x, y) to the surface of T_o and take the

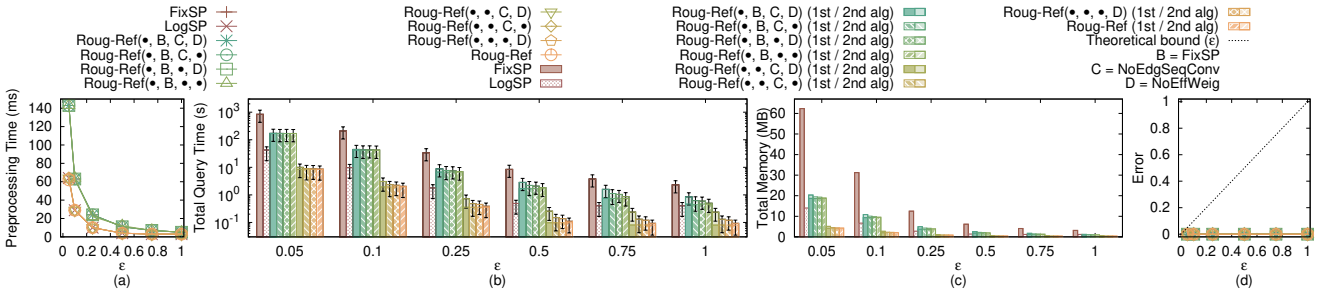

Fig. 20. Effect of ϵ on *EP-small* dataset (V2V path query)

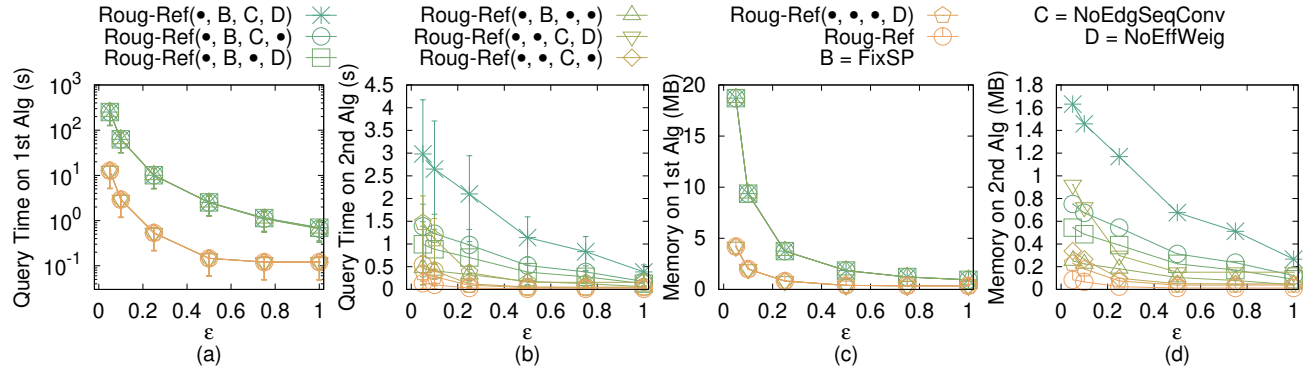

Fig. 21. Effect of ϵ on *EP-small* dataset with separated query time and memory usage in two steps (V2V path query)

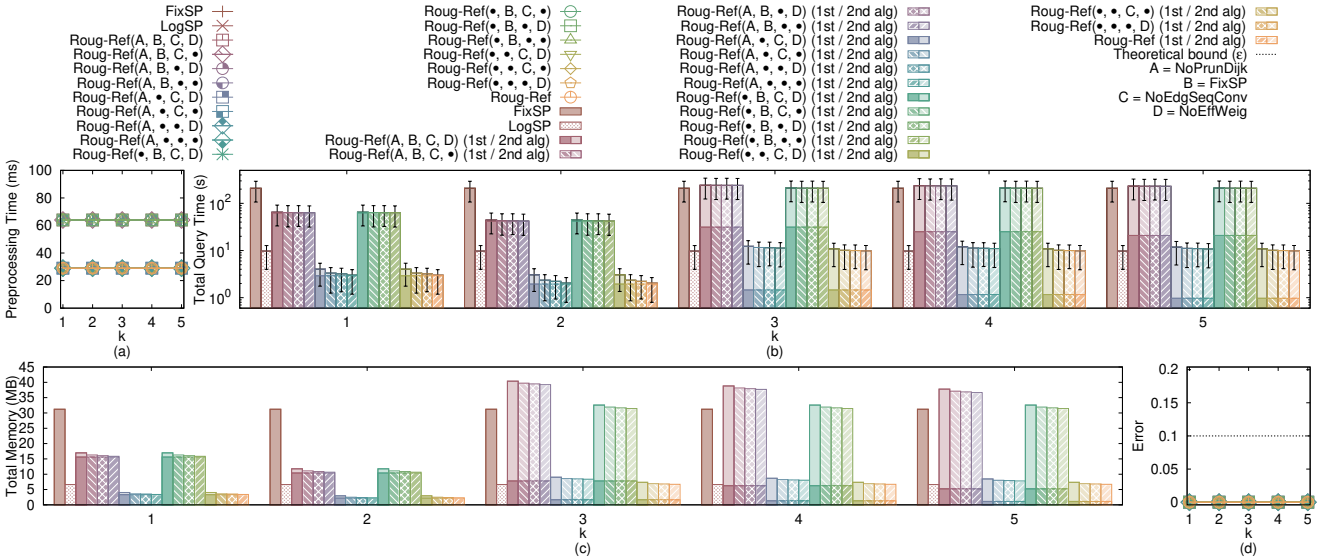

Fig. 22. Effect of removing value on *EP-small* dataset (V2V path query)

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 ϵ 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 ϵ 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 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 ϵ 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 ϵ' in terms of ϵ.*

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

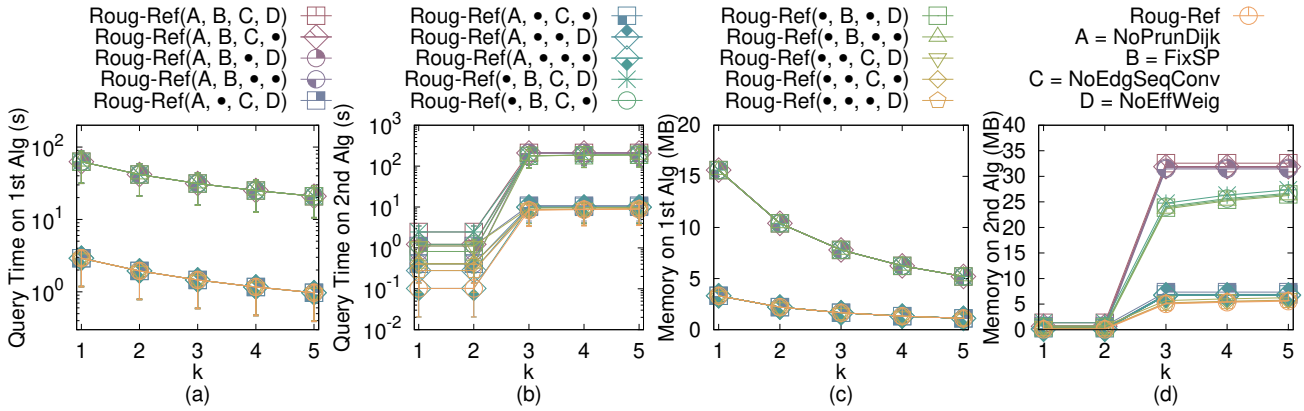

Fig. 23. Effect of removing value on *EP-small* dataset with separated query time and memory usage in two steps (V2V path query)

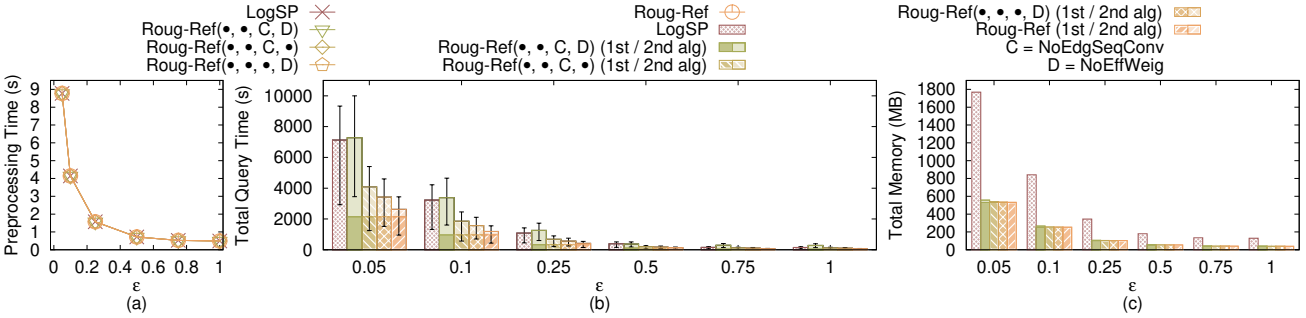

Fig. 24. Effect of ϵ on *EP* dataset (V2V path query)

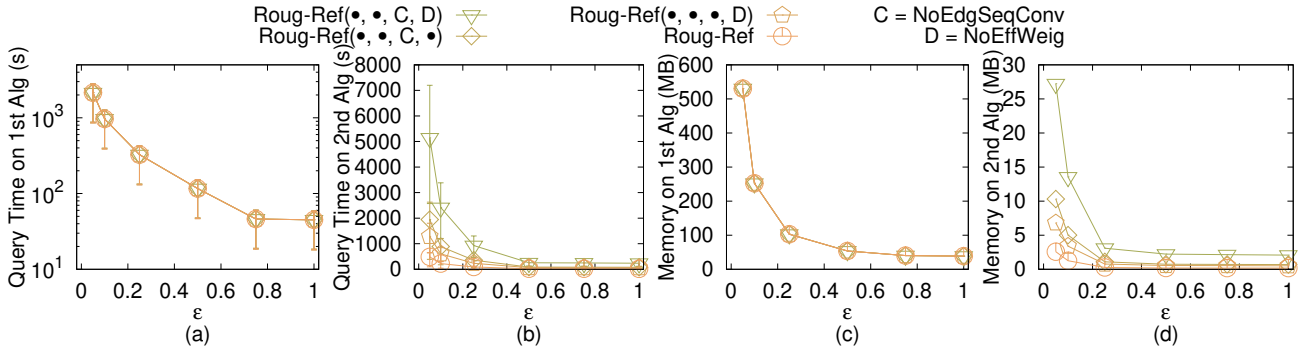

Fig. 25. Effect of ϵ on *EP* dataset with separated query time and memory usage in two steps (V2V path query)

follows.

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

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

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

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

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

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

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

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

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

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

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

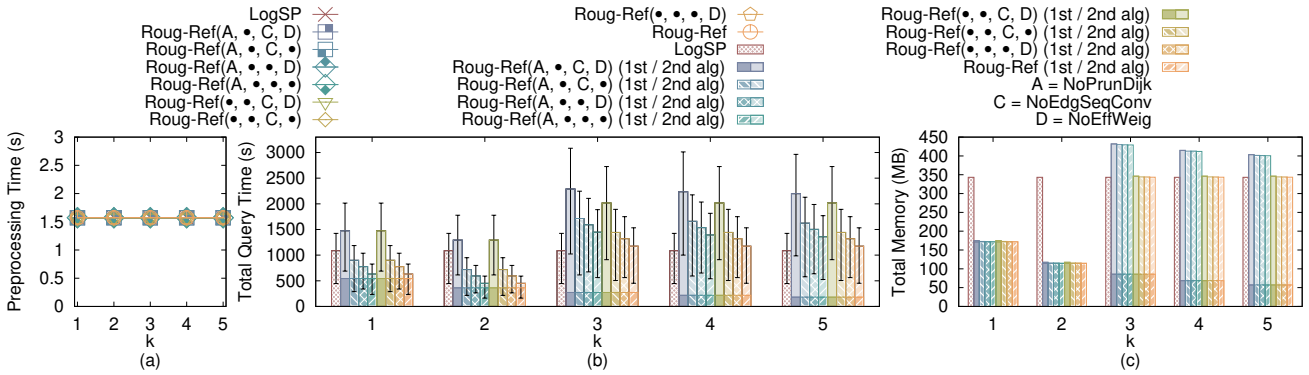

Fig. 26. Effect of removing value on *EP* dataset (V2V path query)

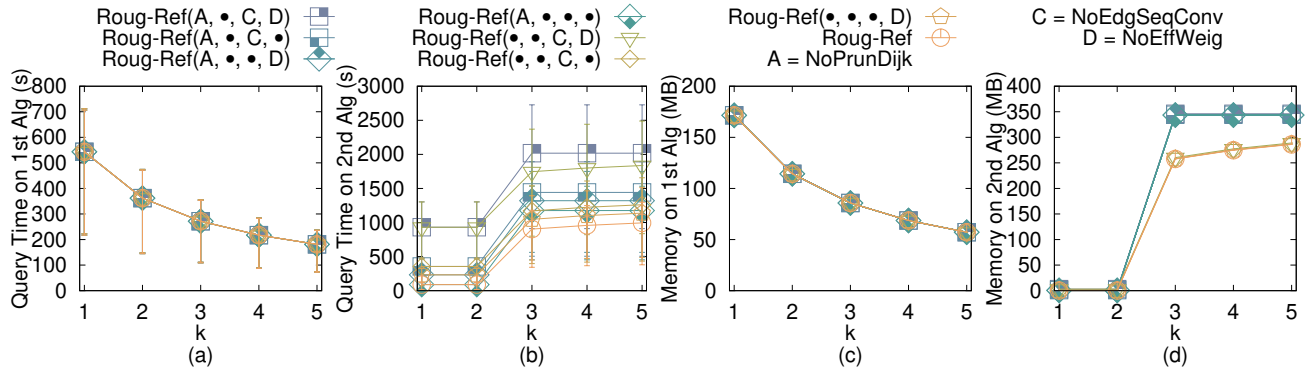

Fig. 27. Effect of removing value on *EP* dataset with separated query time and memory usage in two steps (V2V path query)

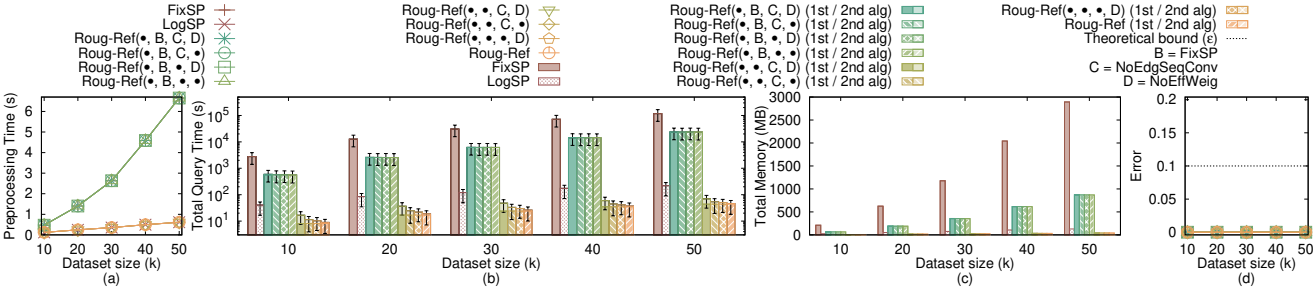

Fig. 28. Effect of dataset size on multi-resolution of *EP-small* datasets (V2V path query)

Proof of Theorem 2. Originally, if we do not remove Steiner points in algorithm *Roug*, i.e., we place Steiner point based on ϵ, then following Lemma 1, the number of Steiner points k_{SP} on each edge is $O(\log_\lambda \frac{L}{r})$, where $\lambda = (1 + \epsilon' \cdot \sin\theta)$ 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 η calculation and rough path calculation, the remaining Steiner points on each edge is $O(1)$. So $|G_{Roug}.V| = n$. Since we know for a graph with n vertices, the running time and memory usage of Dijkstra's algorithm on this graph are $O(n \log n)$ and n, so the running time of algorithm *Roug* is $O(n \log n)$ and the memory usage is $O(n)$. □

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

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

Secondly, we prove the *memory usage*. Algorithm *Roug* needs $O(n)$ memory since it is a common Dijkstra's algorithm,

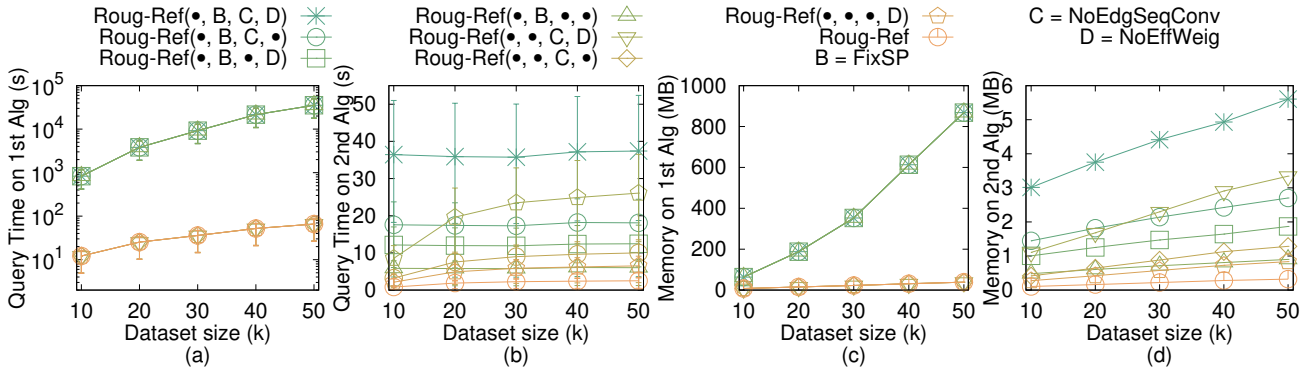

Fig. 29. Effect of dataset size on multi-resolution of *EP-small* datasets with separated query time and memory usage in two steps (V2V path query)

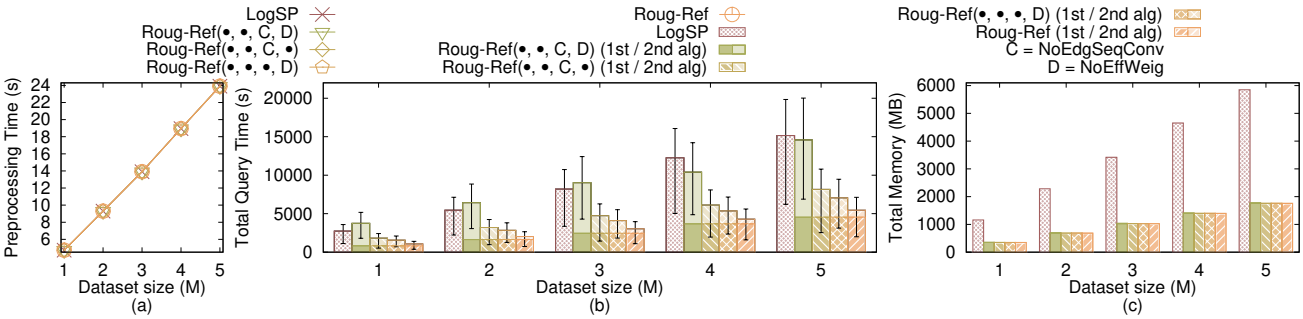

Fig. 30. Effect of dataset size on multi-resolution of *EP* datasets (V2V path query)

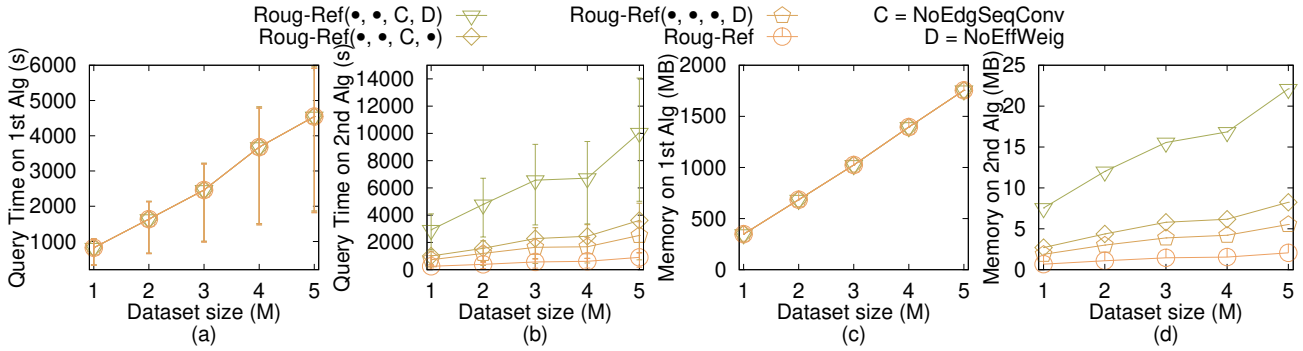

Fig. 31. Effect of dataset size on multi-resolution of *EP* datasets with separated query time and memory usage in two steps (V2V path query)

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* 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

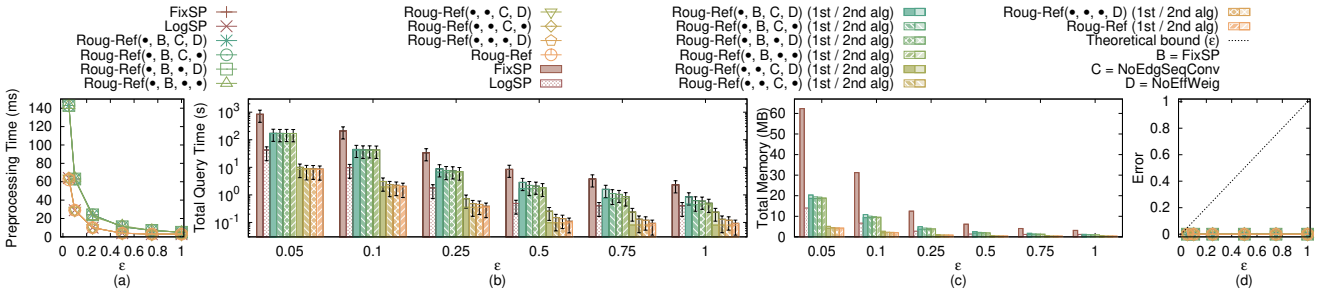

Fig. 32. A2A path query on *EP-small* dataset

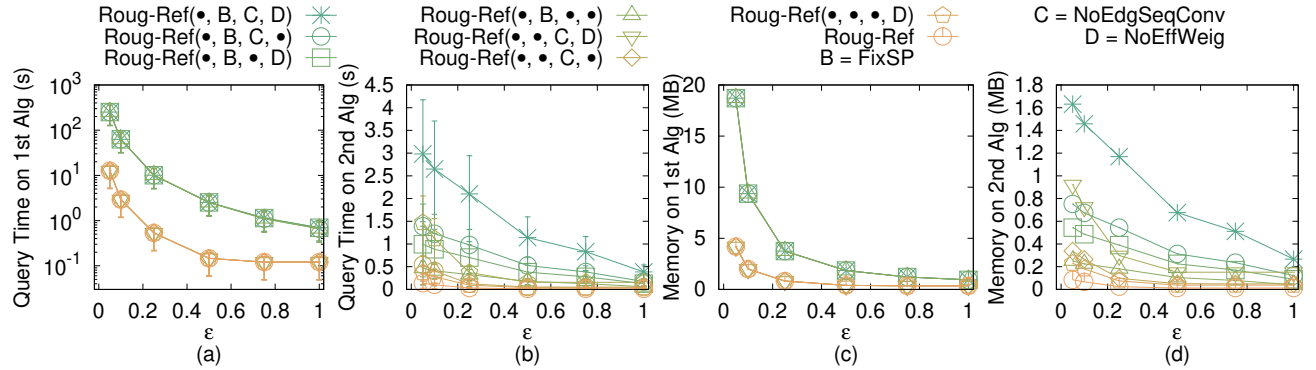

Fig. 33. A2A path query on*EP-small* dataset with separated query time and memory usage in two steps

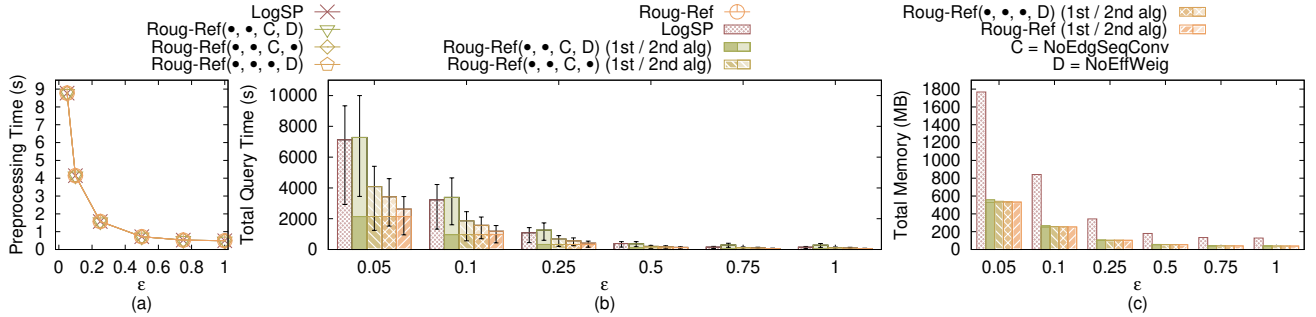

Fig. 34. A2A path query on *EP* dataset

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

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

Finally, we prove the *error bound*. Recall one baseline algorithm *LogSP*, 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 ϵ). We define the path calculated by algorithm *LogSP* between s and t to be $\Pi_{LogSP}(s,t)$. The work [7], [26] show that $|\Pi_{LogSP}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. A proof sketch appears in Theorem 1 of work [7] and a detailed proof appears in Theorem 3.1 of work [26]. But, the work [7], [26] 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

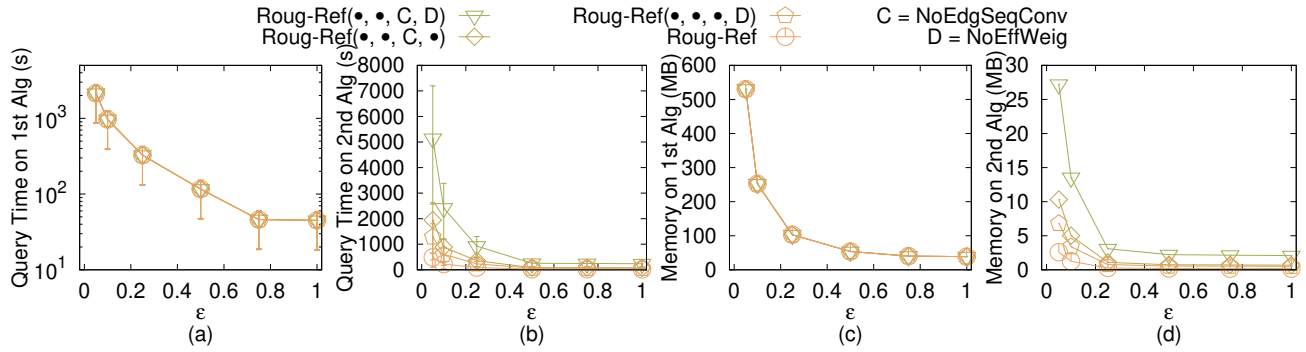

Fig. 35. A2A path query on *EP* dataset with separated query time and memory usage in two steps

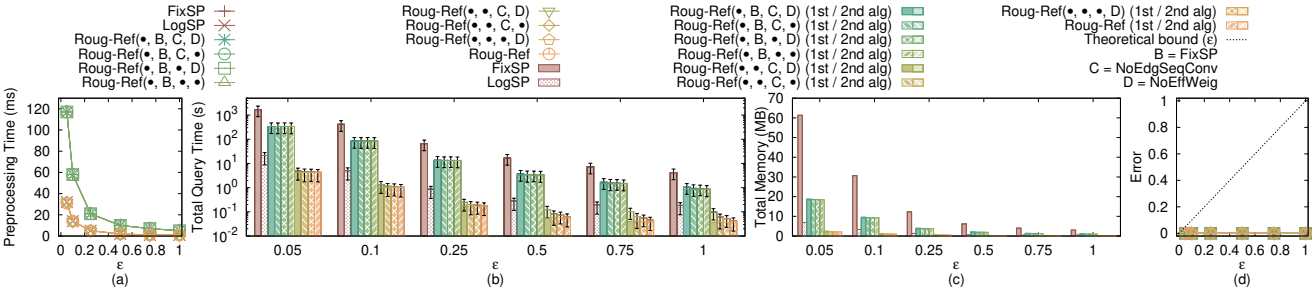

Fig. 36. Effect of ϵ for Path Advisor user study

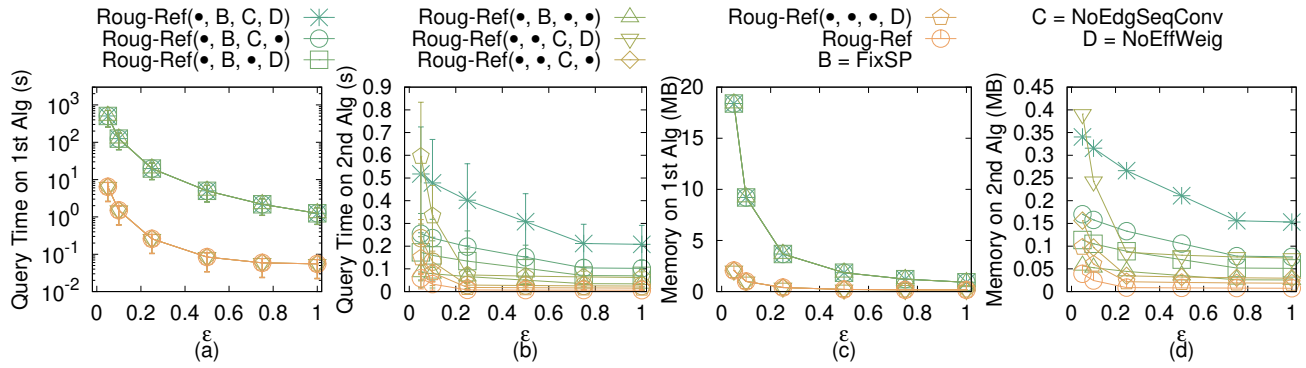

Fig. 37. Effect of ϵ for Path Advisor user study with separated query time and memory usage in two steps

$|\Pi_{Roug}(s,t)| \leq (1+\eta\epsilon)|\Pi^*(s,t)|$. Next, in the path checking step of algorithm *Ref*, if $|\Pi_{Ref\text{-}2}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s,t)|$, we return $\Pi_{Ref\text{-}2}(s,t)$ as output $\Pi(s,t)$, which implies that $|\Pi_{Ref\text{-}2}(s,t)| = |\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. Otherwise, we use the error guaranteed path refinement step in algorithm *Ref*, and we return $\Pi_{Ref\text{-}3}(s,t)$ as output $\Pi(s,t)$, where the error bound is the same as in algorithm *LogSP*, i.e., $|\Pi_{Ref\text{-}3}(s,t)| = |\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. 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 [27]. Since there are total n edges, there are total $O(n^3)$ Steiner points in the weighted graph, so the running time and memory usage of using Dijkstra's algorithm, i.e., algorithm *FixSP*, on this graph are $O(n^3 \log n)$ and $O(n^3)$.

Secondly, we prove the *error bound*. Theorem 2.6 of work [26] shows that $|\Pi_{FixSP}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. □

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 $|\Pi_{LogSP}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{LogSP}(s,t)$ 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 $|\Pi_{LogSP}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$. □

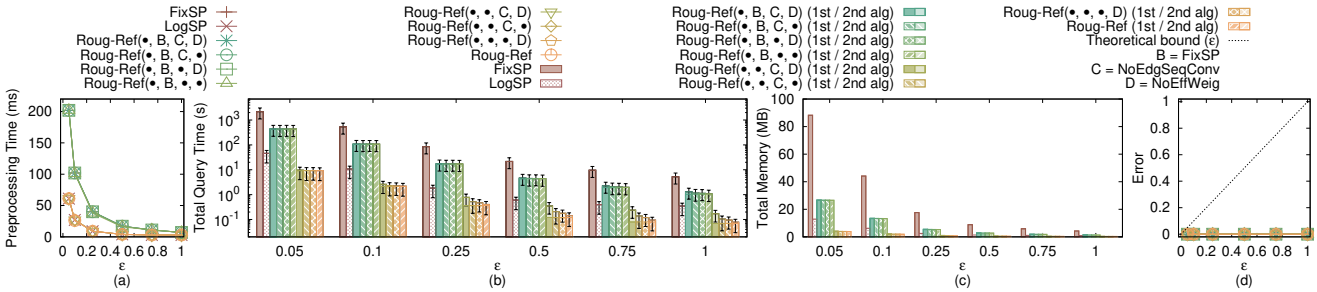

Fig. 38. Effect of ϵ for Cyberpunk 2077 user study

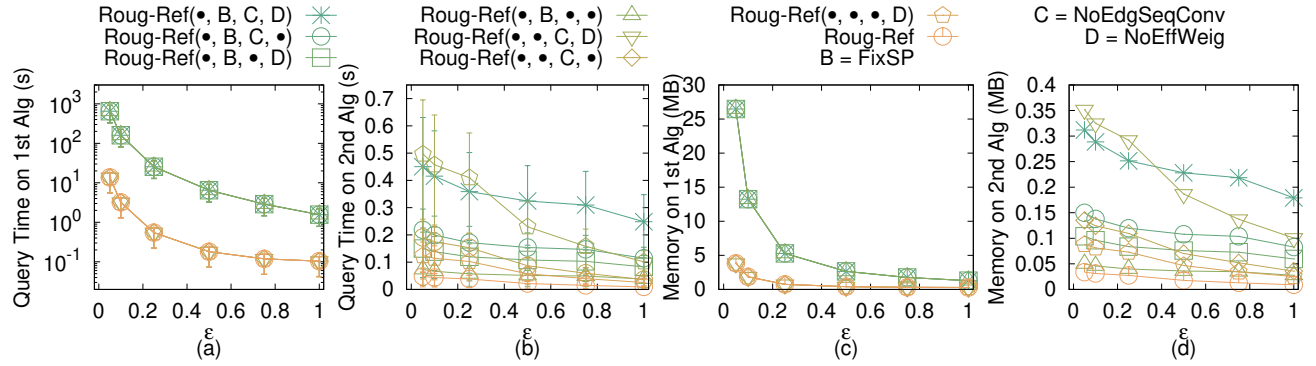

Fig. 39. Effect of ϵ for Cyberpunk 2077 user study with separated query time and memory usage in two steps

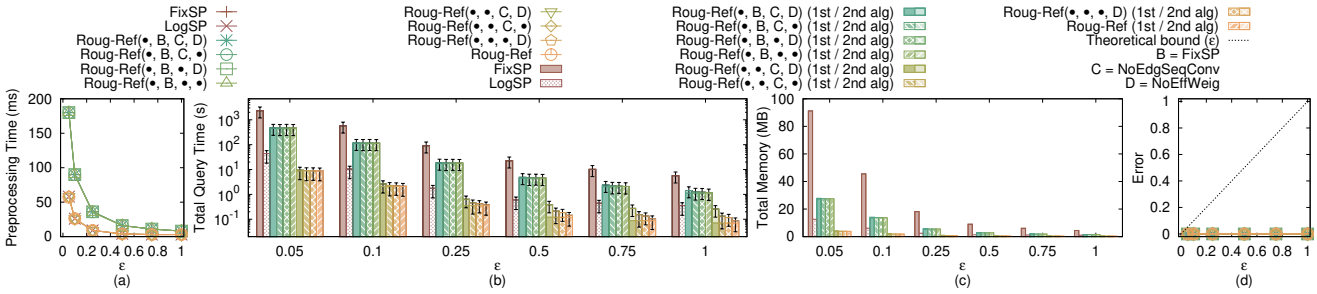

Fig. 40. Effect of ϵ for seabed motivation study

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* $|\Pi_{\textit{Roug-Ref(NoPrunDijk,•,•,•)}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$*, where* $\Pi_{\textit{Roug-Ref(NoPrunDijk,•,•,•)}}(s,t)$ *is the path result returned by algorithm Roug-Ref(NoPrunDijk, •, •, •).*

Proof 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. 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

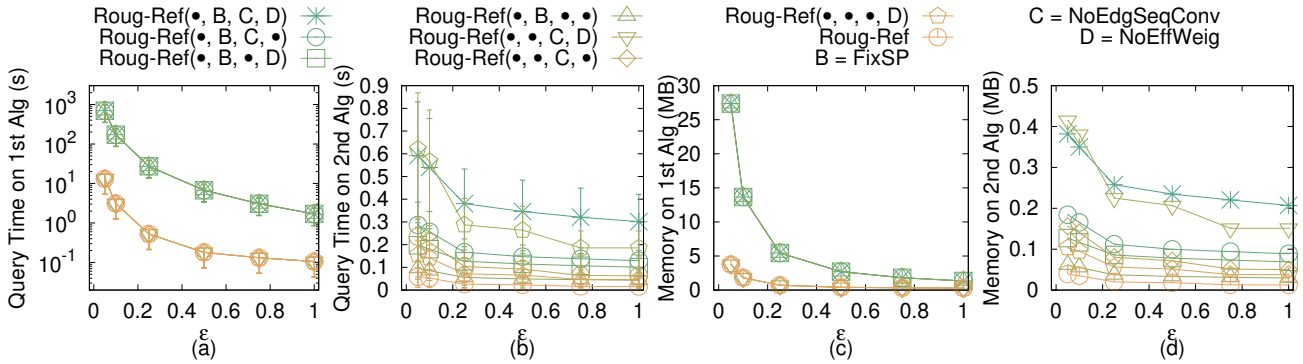

Fig. 41. Effect of ϵ for seabed motivation study with separated query time and memory usage in two steps

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 $|\Pi_{\textit{Roug-Ref}(\bullet,\textit{FixSP},\bullet,\bullet)}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{Roug-Ref}(\bullet,\textit{FixSP},\bullet,\bullet)}(s,t)$ 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 [27]. Since there are total n edges, there are total $O(n^3)$ Steiner points in the weighted graph, so the running time and memory usage of using Dijkstra's algorithm on this graph are $O(n^3 \log n)$ and $O(n^3)$. By using the framework of *Roug-Ref*, we obtain the total running time and total memory usage, i.e., $O(n^2 \log n + l)$ and $O(n^2 + l)$.

Secondly, we prove the *error bound*. From Theorem 5, we know that $|\Pi_{\textit{FixSP}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{FixSP}}(s,t)$ is the path result returned by algorithm *FixSP*. Since from Theorem 1, we also have $|\Pi_{\textit{LogSP}}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{LogSP}}(s,t)$ is the path result returned by algorithm *LogSP*, so by using the framework of *Roug-Ref*, we obtain the error bound of algorithm *Roug-Ref*(•, *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 $|\Pi_{\textit{Roug-Ref}(\bullet,\bullet,\textit{NoEdgSeqConv},\bullet)}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{Roug-Ref}(\bullet,\bullet,\textit{NoEdgSeqConv},\bullet)}(s,t)$ 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 $|\Pi_{\textit{Roug-Ref}(\bullet,\bullet,\bullet,\textit{NoEffWeig})}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{\textit{Roug-Ref}(\bullet,\bullet,\bullet,\textit{NoEffWeig})}(s,t)$ 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 Π_m since there are l edges in S and we need to use Snell's law l times to calculate the intersection point on each edge. Then, 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 $\delta = \frac{h\epsilon w}{6lW}$ and $L_i \leq L$ for $\forall i \in \{1, \ldots, l\}$, the running time is $O(\log \frac{lWL}{h\epsilon w})$. Since we run the above two nested loops l times, the total running time is $O(l^2 \log \frac{lWL}{h\epsilon w})$. So the running time of the

Snell's law path refinement step without the effective weight pruning sub-step in algorithm *Ref* is $O(l^2 \log \frac{lWL}{h\epsilon w})$. By using the framework of *Roug-Ref*, we obtain the total running time is $O(n \log n + l^2 \log(\frac{lNWL}{w\epsilon}))$.

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