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 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 exceeds two, we propose an efficient $(1+\epsilon)$-approximate algorithm to solve it. Our experimental results show that our algorithm is up to 1630 times and 40 times better than the best-known algorithm in terms of running time and memory usage in realistic settings.

PVLDB Reference Format:
Yinzhao Yan and Raymond Chi-Wing Wong. Efficiently Finding Shortest Path on 3D Weighted Terrain Surfaces. PVLDB, 17(1): XXX-XXX, 2024.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/yanyinzhao/WeightedTerrainCode.

1 INTRODUCTION

In recent years, terrain data becomes increasingly widespread in industry and academia [36]. In industry, many companies and applications, including Metaverse [5, 25, 26], Cyberpunk 2077 [1] and Google Earth [4], are using terrain datasets, 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, the shortest path query on terrain datasets is a prevalent research topic in the field of databases [15, 16, 19–21, 29, 32–37]. A terrain surface contains a set of *faces* each of which is denoted by a triangle. Each face consists of three *edges* connecting at three *vertices*. The *weighted* (resp. *unweighted*) *shortest path* on a terrain surface means the shortest path between a source s and a destination t passing on the face of the terrain where each face is assigned with a *weight* (resp. each face weight is set to a fixed value, e.g., 1). Figure 1 (a) shows a terrain surface with a weighted (resp. unweighted) shortest path in the blue (resp. purple dashed) line from s to t.

Proceedings of the VLDB Endowment, Vol. 17, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

1.1 Motivation

Given a source s and a destination t, computing the weighted shortest path on terrain surfaces between s and t is involved in numerous applications with different interpretations of the faces' weights on the terrain, including autonomous vehicles' obstacle avoidance path planning and human's route-recommendation systems [18, 31, 33, 37, 38].

(1) **Robot path planning**: Figure 1 (a) shows the navigation of a robot on a weighted 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 avoids 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 purple dashed path that passes water is larger, and the robot will choose the blue path that does not pass water (leading to a smaller weighted length).

(2) **Path Advisor**: Figure 1 (b) shows a navigation tool called *Path Advisor* [37] for calculating the shortest path between two rooms in a university campus. The floor of the building is represented in a weighted terrain surface. For safety, the path should maintain a minimum distance (e.g., at least 0.2 meters) from obstacles. So, faces on the floor closer to the aisle boundaries (resp. centers) are assigned higher (resp. lower) weights. In this figure, the weighted shortest path in blue line is more realistic than the unweighted shortest path in purple dashed line, since the former path maintains a safe distance from obstacles and avoids abrupt changes in direction.

(3) **Cable placement**: Figure 1 (c) shows an example of the placement of undersea optical fiber cable on the seabed (which is a weighted terrain surface), where over 1.35×10^5km of undersea cables have been deployed nowadays [22]. We want to minimize the weighted length of the cable for cost saving. Deeper sea levels result in higher hydraulic pressure, which reduces the cable's lifespan and increases repair and maintenance costs [11]. We assign larger weights to the terrain faces representing deeper sea levels. So, we can avoid placing the cable in these regions and reduce costs. In this figure, the weighted shortest path in blue line (resp. unweighted shortest path in purple dashed line) is routed in the regions with shallower (resp. deeper) sea levels. Our motivation study shows that the estimated cost of the cable following the weighted (resp. unweighted) shortest path is USD \$45.8M (resp. \$54.8M), which shows the usefulness of finding the weighted shortest path.

Motivated by these, we aim to solve the *weighted region problem*, i.e., 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.

1.2 Snell's law

Consider a weighted terrain surface T with n vertices. Let V, E and F be the set of vertices, edges and faces of T, respectively. Snell's

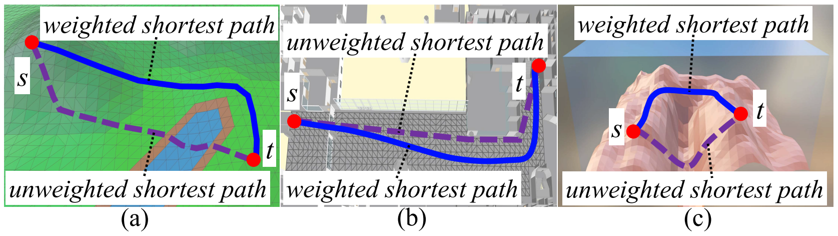

Figure 1: Weighted and unweighted shortest paths on terrain surfaces

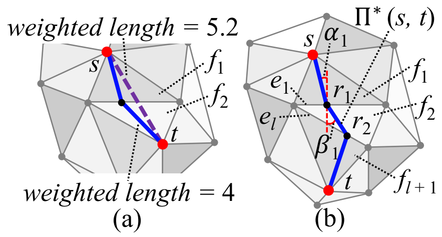

Figure 2: Snell's law illustration

law (one widely known fact from Physics) [30] is an important geometric information of T between two adjacent faces in F that share one edge. This law 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 since light seeks the path with the minimum time as described by *Fermat's Principle* [10]. The angles of incidence and refraction for the light satisfy Snell's law. Figures 2 (a) and (b) show examples for calculating the weighted shortest path using Snell's law. In Figure 2 (a), if all faces have the same weights (=1) (which is the setting of the unweighted terrain), it is obvious that the purple dashed straight line is the (unweighted) shortest path. However, when faces f_1 and f_2 have weights 3 and 2, the weighted shortest path, satisfying Snell's law, is the blue path, which involves two line segments and has a weighted length of 4. We can see that the two line segments bend at the boundary between f_1 and f_2. Note that the weighted length of the purple line is 5.2 > 4. In Figure 2 (b), the blue path is another example of a weighted shortest path.

1.3 Limitations in existing studies

No algorithm can solve the weighted region problem *exactly* when the number of faces in T exceeds two [14]. There are three categories of algorithms for solving this problem on-the-fly *approximately*: (1) *wavefront* approach [31], (2) *Steiner point* approach [7, 20, 24], and (3) *edge sequence* approach [33].

(1) **Wavefront approach**: Algorithm [31] uses continuous Dijkstra's algorithm (like a wavefront) and Snell's law to calculate a $(1+\epsilon)$-approximate weighted shortest path (where $\epsilon > 0$ is the *error parameter*) in more than $O(n^8)$ time, which is very large, since it involves many redundant checking operations.

(2) **Steiner point approach**: Algorithms [7, 20, 24] first place discrete points (i.e., Steiner points) on edges in E, and then use Dijkstra's algorithm [17] on a weighted graph constructed using these Steiner points and V to calculate an approximate weighted shortest path. The best-known algorithm [20, 24] calculates a $(1+\epsilon)$-approximate weighted shortest path in $O(n^3 \log n)$ time, which is still very slow, since it does not utilize any geometric information on T and thus, need to place many Steiner points. Our experimental results show that the best-known algorithm [20, 24] runs in 119,000s ($\approx$ 1.5 days) on a terrain with 50k faces, which is not acceptable.

(3) **Edge sequence approach**: Algorithm [33] first uses the best-known algorithm [20, 24] to calculate a $(1+\epsilon)$-approximate weighted shortest path and an edge sequence that this path passes based on T, and then uses Snell's law on the edge sequence to calculate a result path with a shorter distance. It runs in more than $O(n^3 \log n)$ time (i.e., an additive of the running time of the best-known algorithm [20, 24] and usage of Snell's law), since it does not consider any pruning technique in the edge sequence finding.

1.4 Our Efficient Algorithm

We propose an efficient on-the-fly two-step algorithm for solving 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. Algorithm *Roug-Ref* is a step-by-step algorithm involving algorithm *Roug* and algorithm *Ref*. (1) In algorithm *Roug*, given T, s, t and ϵ, we efficiently find a $(1+\eta\epsilon)$-approximate *rough path* between s and t using Steiner points, where $\eta > 1$ is a constant and is calculated based on T and ϵ (note that $\eta \in (1, 2]$ on average in our experiments), as shown in Figures 3 (a) to (d). (2) In algorithm *Ref*, given the rough path, we efficiently *refine* this path to be a $(1+\epsilon)$-approximate weighted shortest path using Snell's law, as shown in Figures 3 (e) to (i). Algorithm *Roug-Ref* achieves outstanding performance in terms of running time and memory usage due to the rough-refine concept, i.e., (1) a *novel* pruning step in algorithm *Roug* during the rough path calculation (achieved by transferring the pruned-out information from algorithm *Roug* to algorithm *Ref*, and after conducting necessary checks in algorithm *Ref*, there is no need to perform calculations on the pruned-out information anymore), which is absent in algorithms [7, 31, 33] and the best-known algorithm [20, 24], (2) an *efficient* reduction of the search area in algorithm *Roug* before Snell's law refinement (achieved by the calculation of the rough path), which is absent in algorithm [7, 31] and the best-known algorithm [20, 24], and (3) the usage of *Snell's law* in algorithm *Ref* for efficient refinement, which is absent in algorithm [7] and the best-known algorithm [20, 24].

We have four additional novel techniques to further reduce the algorithm running time and memory usage, including (1) an *efficient* Steiner point placement scheme in algorithm *Roug* for reducing the number of Steiner points during the rough path calculation (achieved by considering geometric information of each face in F on T, such as face weight, internal angle and edge length), such that this technique alone can already calculate a $(1+\epsilon)$-approximate weighted shortest path (but using only this technique without incorporating our other techniques will result in a larger running time and memory usage), (2) a *progressive* approach in algorithm *Ref* for minimizing the search area before Snell's law refinement (achieved by progressively exploring the local search area instead of directly using the global search area), (3) an *effective weight* pruning technique in algorithm *Ref* for faster processing during Snell's law

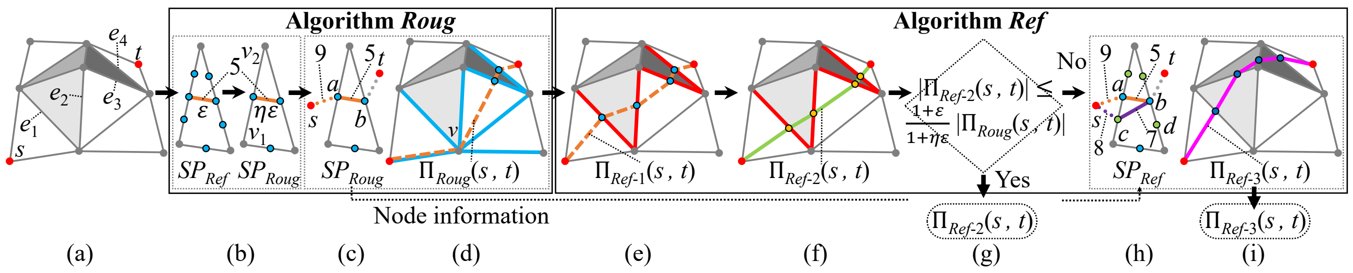

Figure 3: Framework overview

refinement (achieved by considering additional information on T), and (4) a *novel* error guaranteed pruning technique in algorithm *Ref* for handling rare cases when we are unable to use Snell's law to refine a rough path to a $(1+\epsilon)$-approximate weighted shortest path, then an additional step is required for error guarantee, but the total running time of algorithm *Roug-Ref* remains comparable to the best-known algorithm [20, 24] (achieved by re-using the calculated information in algorithm *Roug*). These techniques are absent in algorithms [7, 31, 33] and the best-known algorithm [20, 24].

1.5 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 no existing study has employed the rough-refine concept to calculate the weighted shortest path within the error bound.

(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 [20, 24] 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 [20, 24] 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 [20, 24] runs in 119,000s ($\approx$ 1.5 days) and 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 2 provides the preliminary. Section 3 covers the related work. Section 4 presents our algorithm. Section 5 presents the experimental results and Section 6 concludes the paper.

2 PRELIMINARY

2.1 Notations and Definitions

2.1.1 ***Terrain surfaces and paths.*** Consider a weighted terrain surface T with n vertices. Let V, E and F be the set of vertices, edges and faces of T, respectively. Each vertex $v \in V$ has three coordinate values, x_v, y_v and z_v. Let L be the length of the longest edge of T, and N be the smallest integer value that is larger than or equal to the coordinate value of any vertex in V. If two faces share a common edge, they are said to be *adjacent*. Each face $f_i \in F$ is assigned a weight $w_i > 0$, and the weight of an edge is the smaller weight of the two faces containing the edge. The maximum and minimum weights of the face in F are denoted by W and w, 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 $\overline{pq}$ to be a line segment between p and q on f_i, $d(p,q)$ 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, let $\Pi^*(s,t) = (s, r_1, \ldots, r_l, t)$ be the *optimal weighted shortest path* on T (with $l \geq 0$) such that $\sum_{i=0}^{l} D(r_i, r_{i+1})$ is the minimum, where $r_0 = s$ and $r_{l+1} = t$. Here, for each $i \in \{1, \ldots, l\}$, r_i is a point on an edge in E, is named as an *intersection point* in $\Pi^*(s,t)$. The blue path in Figure 2 (b) shows an example of $\Pi^*(s,t) = (s, r_1, r_2, t)$ on T. We define $|\cdot|$ to be the weighted distance of a path. For example, $|\Pi^*(s,t)|$ is the weighted distance of $\Pi^*(s,t)$. Let $\Pi(s,t)$ be the path result of our algorithm.

2.1.2 ***Snell's Law.*** Let $S = ((v_1, v_1'), \ldots, (v_l, v_l')) = (e_1, \ldots, e_l)$ be a sequence of edges that $\Pi^*(s,t)$ connects from s to t in order based on T, where S is said to be *passed by* $\Pi^*(s,t)$. Let l be the number of edges in S. Let $F(S) = (f_1, f_2, \ldots, f_l, f_{l+1})$ be an adjacent face sequence corresponds to S, such that for every f_i with $i \in \{2, \ldots l\}$, f_i is the face that contains e_i and e_{i+1} in S, while f_1 is the face that adjacent to f_2 at e_1 and f_{l+1} is the face that adjacent to f_l at e_l. Let $W(S) = (w_1, \ldots, w_{l+1})$ be a weight list with respect to $F(S)$ such that w_i is the face weight of f_i in $F(S)$ with $i \in \{1, \ldots l+1\}$. We define α_i and β_i to be the incidence and refraction angles of $\Pi^*(s,t)$ on e_i for $i \in \{1, \ldots l\}$, respectively. Proposition 2.1 states Snell's law with $i \in \{1, \ldots, l\}$ (see Figure 2 (b)).

Proposition 2.1. *$\Pi^*(s,t)$ has $w_i \cdot \sin \alpha_i = w_{i+1} \cdot \sin \beta_i$.*

2.1.3 ***Shortest path queries.*** There are two types of queries, (1) *vertex-to-vertex* (*V2V*) *query*, i.e., both s and t are in V, and (2) *any point-to-any point* (*A2A*) *query*, i.e., s and t are two any points on T. If point s (or t) is not in V, we can regard it as a new vertex and then add 3 new triangles each involving this point and three vertices of the face containing this point. Then, we can use the V2V query on this modified setting for the A2A query. We focus on the V2V query in this paper and study the A2A query in the appendix. A notation table can be found in the appendix of Table 3.

2.2 Problem

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

2.3 Challenges

2.3.1 ***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 algorithm [12] 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 and destination using a straight line segment. However, it cannot be used for the weighted case, since 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 [31]. Thus, we cannot connect s and t with a straight line segment in the unfolded 2D weighted terrain.

2.3.2 ***No exact algorithm.*** No algorithm can solve the weighted region problem *exactly* when the number of faces in T exceeds two [14]. In Figure 2 (b), 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 *optimal* weighted shortest path, where c is called *optimal point*. One may assume that we can set the position of c to be unknown and solve it using a polynomial equation. But, even if the path crosses only three faces, the equation will contain unknown with a degree of six, which is unsolvable by Algebraic Computation Model over the Rational Numbers [14].

3 RELATED WORK

3.1 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.

3.1.1 ***Wavefront approach.*** Algorithm Continuous Wavefront (*ConWave*) [31] uses continuous Dijkstra's algorithm and Snell's law to calculate a $(1+\epsilon)$-approximate weighted shortest path in $O(n^8 \log(\frac{n^2 NWL}{w\epsilon}))$ time.

Drawback of algorithm *ConWave*: It is very slow due to the *presence of unnecessary edges in the edge sequence results.* 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 have no information about where it goes next. To handle this, algorithm *ConWave* uses a continuous Dijkstra's algorithm to check all the combinations of the edge sequences, so the running time is very large. But, we efficiently reduces the search area by determining a unique edge sequence S passed by the weighted shortest path, and uses Snell's law on S to find the result path for reducing the running time.

3.1.2 ***Steiner point approach.*** Algorithms [7, 20, 24] use Dijkstra's algorithm on the weighted graph constructed using Steiner points and V to calculate an approximate weighted shortest path. Algorithm Fixed Steiner Point (*FixSP*) [20, 24] and algorithm Logarithmic Steiner Point (*LogSP*) [7] calculates the result path with an error $(1+\epsilon)$ and an error much larger than $(1+\epsilon)$, respectively. (1) Algorithm *FixSP*, which runs in $O(n^3 \log n)$ time, is regarded as the best-known $(1+\epsilon)$-approximate algorithm for solving the weighted region problem. One variant of algorithm *FixSP* [20] was originally designed for finding the unweighted shortest path, but it can also solve the weighted region problem by assigning a weight w_i to each face f_i, and the surface distance between any two points p and q on face f_i changes from $d(p,q)$ to $w_i \cdot d(p,q)$. The running time of algorithm [20] changes from $O((n+n')\log(n+n'))$ (in the unweighted case) to $O(n^3 \log n)$ (in the weighted case), where n' is the total number of Steiner points, and we have $n' = O(n^3)$ [24]. The weighted case of algorithm [20] is equivalent to algorithm [24]. (2) In algorithm *LogSP*, given an error parameter $\epsilon' > 0$, determining how far the Steiner's points are placed (which is different from our ϵ, a user parameter requiring the guarantee of the distance bound), it places $O(\log \frac{c}{\epsilon'})$ Steiner points per edge and has a distance error $(1 + (2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon')$, where $c \in [0.2, 1]$ is a constant depending on T. We adapt it to be algorithm *LogSP-Adapt*, such that given ϵ, we want to place Steiner points using ϵ and have a distance error $(1+\epsilon)$ (not $(1 + (2 + \frac{2W}{(1-2\epsilon)\cdot w})\epsilon)$). Specifically, the adaption is achieved by finding the relationship between ϵ and ϵ', such that algorithm *LogSP-Adapt* and *LogSP* can place the same number of Steiner points per edge, and their distance errors are the same, i.e., $1 + (2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = 1 + \epsilon$. We solve the quadratic equation and get $\epsilon' = \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4}$. So in algorithm *LogSP-Adapt*, given ϵ, we places $O(\log \frac{c}{\epsilon'}) = O(\log \frac{4c}{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}})$ Steiner points per edge and have a distance error $(1+\epsilon)$.

Drawbacks of algorithm *FixSP*: It is very slow due to two reasons. (1) *Absence of Snell's law*: It does not utilize any geometric information *between two adjacent faces in F that share one edge* on T (with the help of Snell's law). So, it places many Steiner points on edges in E, and every two adjacent Steiner points on the same edge are very close to each other. But, we utilize Snell's law to avoid placing many Steiner points on edges in E. (2) *Uniform number of Steiner points per edge*: It does not utilize any geometric information of *each face in F* on T, such as face weight, internal angle and edge length. So, no matter how T looks like, it always places a uniform number of (i.e., $O(n^2)$) Steiner points per edge to bound the error [24]. But, we utilize this information and place only $O(n \log c')$ Steiner points per edge, where $c' \in [2, 5]$ is a constant depending on T and ϵ. Figures 4 (a) and (b) show the placement of Steiner points in algorithm *FixSP* and our algorithm *Roug-Ref* with the same error, respectively. Our experimental results show that for a terrain surface with 50k faces and $\epsilon = 0.1$, algorithm *Roug-Ref* just places 10 Steiner points per edge to find a rough path in 71s ($\approx$ 1.2 min), and finds a refined path in 2s, but the best-known algorithm *FixSP* [20, 24] places more than 600 Steiner points per edge to find the result path in 119,000s ($\approx$ 1.5 days).

Drawbacks of algorithm *LogSP*: It has two drawbacks. (1) *Absence of Snell's law*: It has the same drawback of algorithm *FixSP*, making it not efficient. (2) *Larger distance error*: When $\epsilon = \epsilon'$, its distance error $(1 + (2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon')$ is always larger than other algorithms' distance error $(1+\epsilon)$, making it difficult to compare algorithm *LogSP* with other algorithms. But, we adapt it to be algorithm *LogSP-Adapt* (which corresponds to the efficient Steiner point placement scheme in algorithm *Roug-Ref*) for easier comparisons.

Drawback of algorithm *LogSP-Adapt*: It has the same first drawback of algorithm *FixSP*, i.e., *absence of Snell's law*, making it not efficient. Our experimental results show that for a terrain surface with 50k faces and $\epsilon = 0.1$, algorithm *Roug-Ref* runs in 73s ($\approx$ 1.2 min), but algorithm *LogSP-Adapt* runs in 220s ($\approx$ 3.7 min).

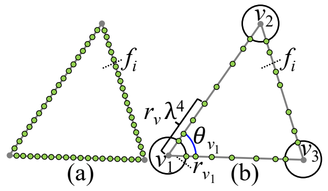

Figure 4: Steiner points in (a) *FixSP*, and (b) *Roug-Ref*

Table 1: Algorithm comparisons

Algorithm	Time/Size	Error
ConWave [31]	Large	Small
EdgSeq [33]	Large	Small
FixSP [19, 20, 24]	Large	Small
LogSP [7]	Medium	Large
LogSP-Adapt [7]	Medium	Small
***Roug-Ref* (ours)**	**Small**	**Small**

3.1.3 ***Edge sequence approach***. Algorithm $\underline{Edge}$ $\underline{Seq}$uence (*EdgSeq*) [33] first employs algorithm *FixSP* to calculate a $(1 + \epsilon)$-approximate weighted shortest path, then determines the edge sequence passed by this path based on T, and finally utilizes Snell's law on the edge sequence to compute a result path with a shorter distance in $O(n^3 \log n + n^4 \log(\frac{n^2 NWL}{w\epsilon}))$ time.

Drawbacks of algorithm *EdgSeq*: It is still very slow due to four reasons. (1) *Absence of pruning technique for the edge sequence finding*: It simply uses Snell's law after employing algorithm *FixSP*, so its running time is an additive result of the running time of algorithm *FixSP* and usage of Snell's law. But, we use a pruning technique, such that even with our efficient Steiner point placement scheme, we can further prune out more than half of the Steiner points. By transferring the pruned Steiner points from algorithm *Roug* to algorithm *Ref*, calculations on these pruned Steiner points can be avoided after conducting necessary checks in algorithm *Ref*. (2) *Uniform number of Steiner points per edge*: It has the second drawback as of algorithm *FixSP*. (3) *Presence of unnecessary edges in the edge sequence results*: It has the same drawback as of algorithm *ConWave*. Although algorithm *EdgSeq* first uses algorithm *FixSP* to find an edge sequence, there are still unnecessary edges in this edge sequence. In Figure 3 (d), suppose that the edges in blue are the calculated edge sequence of algorithm *EdgSeq*. When it uses Snell's law on the edges adjacent to v, only a subset of these edges is required (e.g., the edges in red in Figure 3 (e) is the required edge sequence), so it needs 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. But, we employ a progressive approach, which gradually explores the local search area instead of the global search area to minimize the overall search space for time-saving. (4) *Absence of pruning technique when using Snell's law*: After finding an edge sequence, it solely relies on binary search to find a weighted shortest path using Snell's law on the edge sequence, without utilizing any weight information of T for pruning. But, we consider this information and use *effective weight* for efficient pruning. Our experimental results show that for a terrain surface with 50k faces and $\epsilon = 0.1$, algorithm *EdgSeq* runs in 131,000s ($\approx$ 1.7 days), but *Roug-Ref* runs in 73s ($\approx$ 1.2 min).

3.2 Other related studies

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

(1) *On-the-fly algorithms on the unweighted terrain surface* [8, 12]: As mentioned in Section 2.3.1, algorithm [12] 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 [8] 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 [8, 12] to the weighted case. (2) *Oracles on the terrain surface* [15, 16, 19, 32, 34–36]: The study [19] (resp. the study [34, 35]) builds an oracle on a weighted (resp. unweighted) terrain surface for the shortest path query. The algorithm used for pre-computing the weighted shortest paths in the study [19] is algorithm *FixSP* [24], which runs in $O(mn \log(mn))$ time, where m is the number of Steiner points per edge, and $m = O(n^2)$ [24]. The algorithm used for pre-computing the unweighted shortest paths in the study [34, 35] is algorithm [20], which runs in $O((n + n') \log(n + n'))$ time. After we adapt algorithm [20] to the weighted case, the oracle [34, 35] can be adapted on a weighted terrain surface. Studies [15, 16, 32, 36] build oracles on an unweighted terrain surface for the *k-nearest neighbor* query. But, we focus on the weighted *on-the-fly* shortest path algorithm in this paper, which is a fundamental operator for oracle construction. (3) *Unweighted terrain surface simplification algorithms* [21, 29]: Studies [21, 29] propose simplification algorithms on an unweighted terrain surface. But, they are different from our paper since we focus on the weighted *on-the-fly* shortest path algorithm.

3.3 Comparisons

We compare our algorithm and other on-the-fly algorithms on the weighted terrain surface in Table 1. *Roug-Ref* is the best algorithm since it has the smallest query time and memory usage. The comparisons of all algorithms (with big-O notations) and their theoretical analysis can be found in the appendix.

4 METHODOLOGY

4.1 Overview

4.1.1 ***Concepts***. (1) *Weighted graph*, (2) *removing value*, (3) *node information*, and (4) *full or non-full edge sequence*.

(1) **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 build 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 weighted (surface) distance for edge $\overline{pq}$ is $w_{pq} \cdot d(p, q)$, where w_{pq} means the weight associated with the face or the edge that p and q lie on. In Figure 3 (b), the blue nodes are SP_{Ref} and SP_{Roug}, the blue and gray nodes are $G_{Ref}.V$ and $G_{Roug}.V$, the orange lines between a and b are the weighted edge with weighted distance 5 in $G_{Roug}.E$ and $G_{Ref}.E$.

(2) **The removing value**: It is a constant (denoted by k and usually set to 2) used for calculating SP_{Roug}. Recall that algorithm *Roug-Ref* has a novel and efficient pruning step by transferring

the pruned-out information from algorithm *Roug* to algorithm *Ref*. Indeed, the pruned-out information is the Steiner points. In Figure 3 (b), we first place Steiner points according to SP_{Ref}, and then suppose that 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}.

(3) **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 3 (c), suppose that the weight of this face is 1. After running algorithm *Roug*, the weighted shortest distance from s to a is 9, the Euclidean distance of the orange edge is 5, the weighted shortest path from s to b is $s \rightarrow a \rightarrow b$, i.e., (1) the weighted shortest distance from s to b is $9 + 1 \times 5 = 14$ and (2) the previous node of b along this path is a, so we have $dist_{Roug}(b) = 14$ and $prev_{Roug}(b) = a$ as node information of b.

(4) **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 5 (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, S is said to be a *non-full edge sequence* if there exists at least one edge in S whose length is 0. In Figure 5 (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. In Figure 5 (b), the edge sequences passed by the path in the purple line-dashed line and pink dot-dashed line are full edge sequences, and the edge sequence passed by the path in the orange line is a non-full edge sequence. As we will discuss later, a full (resp. non-full) edge sequence reduces (resp. increases) algorithm *Ref*'s running time.

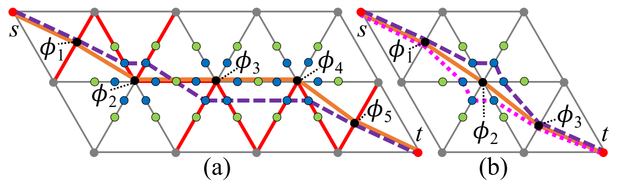

Figure 5: Full edge sequence conversion

4.1.2 *Overview of algorithm Roug.*

Given T, s, t, ϵ and k (the removing value), we find a-$(1+\eta\epsilon)$ approximate rough path between s and t, and store the shortest distance and previous node (based on s) as node information. There are two steps:

- η **calculation**: In Figures 3 (a) and (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 Figures 3 (c) and (d), given T, s, t and SP_{Roug}, we use SP_{Roug} and V, i.e., $G_{Roug}.V$, to build a weighted graph G_{Roug}, and then use 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.

4.1.3 *Overview of algorithm Ref.*

Given T, s, t, ϵ, $\Pi_{Roug}(s, t)$ and the node information, we refine $\Pi_{Roug}(s, t)$ and calculate a $(1 + \epsilon)$-approximate weighted shortest path. There are four steps:

- **Full edge sequence conversion**: In Figure 3 (e), given $\Pi_{Roug}(s, t)$ whose corresponding edge sequence is a non-full edge sequence (since the corresponding edge sequence of $\Pi_{Roug}(s, t)$ in Figure 3 (d) 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 3 (f), given T, s, t and 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 3 (g), given $\Pi_{Roug}(s, t)$, $\Pi_{Ref\text{-}2}(s, t)$, ϵ and η, if $|\Pi_{Ref\text{-}2}(s, t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)} |\Pi_{Roug}(s, t)|$, since we guarantee $|\Pi_{Roug}(s, t)| \leq (1 + \eta\epsilon)|\Pi^*(s, t)|$ due to the error bound of Dijkstra's algorithm, we have $|\Pi_{Ref\text{-}2}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$, and we can return $\Pi_{Ref\text{-}2}(s, t)$ as output. Otherwise, we need the next step.
- **Error guaranteed path refinement**: In Figures 3 (h) and (i), given T, s, t, SP_{Ref} and the node information, we use 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 Figure 3 (i). We can guarantee $|\Pi_{Ref\text{-}3}(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ due to the error bound of Dijkstra's algorithm.

4.2 Key Ideas of Rough-Refine Concept

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

4.2.1 *Novel Steiner point pruning step (in the η calculation step of algorithm Roug).*

In Figure 3 (b), after we place Steiner points SP_{Ref} with an error ϵ, we prune out some Steiner points using the removing value k. We only use the remaining Steiner points SP_{Roug} with an error $\eta\epsilon$ for the rough path calculation in algorithm *Roug*, so $SP_{Roug} \subseteq SP_{Ref}$ and $G_{Roug}.V \subseteq G_{Ref}.V$. We do not require a highly accurate calculation of the rough path $\Pi_{Roug}(s, t)$ since we only need the edge sequence passed by it. Then in algorithm *Ref*, the pruned-out Steiner points are transferred to the error guaranteed path refinement step, and it is very likely after the Snell's law path refinement, the path checking step finds that there is no need to perform Dijkstra's algorithm on these pruned-out Steiner points.

But, the *absence of pruning technique for the edge sequence finding* drawback of the algorithm *EdgSeq* [33] makes it inefficient.

4.2.2 ***Efficient reduction of the search area*** *(in the rough path calculation step of algorithm Roug).* In Figure 3 (d), the edge sequence S passed by $\Pi_{Roug}(s,t)$ based on T is used in the Snell's law path refinement step of algorithm *Ref.* If we do not know S, we need to try Snell's law on different combinations of edges in E (so the search area is large) and select the result path with the minimum length, which is time-consuming. This corresponds to the *presence of unnecessary edges in the edge sequence results* drawback of the algorithm *ConWave* [31].

4.2.3 ***Usage of Snell's law*** *(in the Snell's law path refinement step of algorithm Ref).* In Figure 3 (e), we utilize Snell's law on S to efficiently refine $\Pi_{Roug}(s,t)$ to $\Pi_{Ref\text{-}2}(s,t)$, which closely approximates 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 (proportionally depending on ϵ), so there is no need to place many Steiner points on edges in E. But, the *absence of Snell's law* drawback of algorithm *LogSP*, *LogSP-Adapt* [7], and the best-known algorithm *FixSP* [20, 24] results in numerous Steiner points per edge.

4.3 Key Ideas of Additional Techniques

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

4.3.1 ***Efficient Steiner point placement scheme with a $(1+\epsilon)$ error bound*** *(in the η calculation step).* In Figure 3 (b), we have an efficient Steiner point placement scheme (1) by considering the additional geometric information of T, such as face weight, internal angle, and edge length, etc., so that we can place *different* numbers of Steiner points on different edges of E, and the interval between two adjacent Steiner points on the same edge can be *different* (which allows us to minimize the total number of Steiner points), and (2) by using mathematical transformations to express the error in terms of $(1+\epsilon)$ (so that with the η calculation step, the rough path has an error $(1+\eta\epsilon)$). Figure 4 (b) shows an example of our placement of Steiner points, where there are 8, 7, and 7 Steiner points on the edge of f_i. In contrast, (1) the *uniform numbers of Steiner points per edge* drawback of the best-known algorithm *FixSP* [20, 24] and algorithm *EdgSeq* [33] makes their algorithms very slow, and (2) the *large distance error* drawback of algorithm *LogSP* [7] makes it not suitable in the rough path calculation.

4.3.2 ***Novel full edge sequence conversion technique using progressive approach*** *(in the full edge sequence conversion step).* In Figure 3 (d), before we use Snell's law on the edge sequence S passed by the rough path $\Pi_{Roug}(s,t)$, for efficient refinement, we find that $\Pi_{Roug}(s,t)$ may pass on a vertex v (the edge sequence S passed by $\Pi_{Roug}(s,t)$ is a non-full edge sequence), so 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 (which corresponds to the *presence of unnecessary edges in the edge sequence results* drawback of the algorithm *EdgSeq* [33]). However, only a subset of these edges is required. To solve it, we use the full edge sequence conversion step to modify the rough path efficiently (by *progressively* modifying the segments of the rough path near the edges with length 0) and ensure that all the edges passed by the modified rough path have lengths larger than 0, as shown in Figure 3 (e). We use Figure 5, which contains more details, for better illustration.

Illustration: In Figure 5 (a), given a rough path $(s,\phi_1,\phi_2,\phi_3,\phi_4,\phi_5,t)$ (i.e., the orange line 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 (ϕ_4,ϕ_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. Figure 5 (b) shows a very similar example (but has only one edge with a length equal to 0). In algorithm *Roug*, if we build a weighted graph using only the Steiner points (excluding both the Steiner points and V), and then use Dijkstra's algorithm on this graph, it seems that the rough path will not pass the vertex, but we cannot guarantee the rough path to be a $(1+\eta\epsilon)$-approximate weighted shortest path.

4.3.3 ***Novel effective weight pruning technique*** *(in the Snell's law path refinement step).* In Figure 3 (f), we need to utilize Snell's law to find the optimal point of the weighted shortest path on each edge of S, then connect these points to form the refined path. The basic idea is to use binary search, but we can efficiently prune out some checking and reduce the algorithm's running time by utilizing one useful information on T called *effective weight* (which corresponds to the *absence of pruning technique when using Snell's law* drawback of the algorithm *EdgSeq* [33]). We use Figure 6, which contains more details, for better illustration.

Illustration: In Figure 6 (a), for the first edge e_1 (opposite to s) in 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 for pruning. In Figures 6 (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 6 (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} closely approximates 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 6 (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).

4.3.4 ***Novel error guaranteed path refinement using pruning technique*** *(in the error guaranteed path refinement step, with the help of algorithm Roug).*

In Figure 3 (g), for the path checking step, if $|\Pi_{\mathit{Ref}\text{-}2}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{\mathit{Roug}}(s,t)|$ (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, as shown in Figures 3 (h) and (i). We hope that even with this extra step, the running time of algorithm *Roug-Ref* will not increase a lot, by using the node information calculated in algorithm *Roug*.

Illustration: (1) In the rough path calculation step of algorithm *Roug* as shown in Figure 3 (c), we have $\mathit{dist}_{\mathit{Roug}}(b) = 9 + 1 \times 5 = 14$ and $\mathit{prev}_{\mathit{Roug}}(b) = a$ as node information of b. (2) In the error guaranteed path refinement step in algorithm *Ref* as shown in Figure 3 (h), we maintain a *priority queue* [13] in Dijkstra's algorithm (on the weighted graph G_{Ref}). Since $G_{\mathit{Roug}}.V \subseteq G_{\mathit{Ref}}.V$, we can transfer the node information of b from G_{Roug} to G_{Ref}, so $\mathit{dist}_{\mathit{Ref}}(b) = 14$ and $\mathit{prev}_{\mathit{Ref}}(b) = a$. For the priority queue, suppose it stores nodes a and c with $\mathit{dist}_{\mathit{Ref}}(a) = 9$ and $\mathit{dist}_{\mathit{Ref}}(c) = 8$. We dequeue c with a shorter distance value ($8 < 9$), and one adjacent node of c is b. Since we have $\mathit{dist}_{\mathit{Ref}}(b) = 14 < \mathit{dist}_{\mathit{Ref}}(c) + w_{bc} \cdot d(b,c) = 8 + 1 \times 7 = 15$, we do not need to insert b and $\mathit{dist}_{\mathit{Ref}}(b) = 15$ 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 use 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 by performing 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 $|\Pi_{\mathit{Ref}\text{-}2}(s,t)| > \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{\mathit{Roug}}(s,t)|$ 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%.

4.4 Implementation Details of Algorithm *Roug*

We give implementation details of algorithm *Roug*. Figures 3 (a) to (d) show this algorithm. We mainly discuss our efficient Steiner points placement scheme with a $(1+\epsilon)$ error bound.

Detail: Given a vertex v in V, we let h_v be the minimum height starting from v on the faces containing v, and let C_v be a *sphere* centered at v. Let $r_v = \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4} \cdot h_v$ be the radius of C_v. Let θ_v be the angle between any two edges of T that are incident to v. Figure 4 (b) shows an example of the sphere C_{v_1} centered at v_1, with the radius r_{v_1} and the angle θ_{v_1} of v_1. Let $\lambda = (1 + \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4} \cdot \sin\theta_v)$ if $\theta_v < \frac{\pi}{2}$, and $\lambda = (1 + \frac{1+\epsilon+\frac{W}{w}-\sqrt{(1+\epsilon+\frac{W}{w})^2-4\epsilon}}{4})$ otherwise. For each vertex v of face f_i, we place Steiner points $p_1, p_2, \ldots, p_{\tau_p-1}$ on two edges of f_i incident to v, such that $|\overline{vp_j}| = r_v\lambda^{j-1}$, where $\tau_p = \log_\lambda \frac{|e_p|}{2\cdot r_v}$ for every integer $2 \leq j \leq \tau_p - 1$. Figure 4 (b) shows a set of Steiner points on f_i based on ϵ. Then we follow the remaining procedures in algorithm *Roug* to calculate $\Pi_{\mathit{Roug}}(s,t)$.

4.5 Implementation Details of Algorithm *Ref*

We give implementation details of algorithm *Ref*.

4.5.1 ***Full edge sequence conversion.***

Figure 3 (e) shows this step, and Figure 5 shows the detailed example of this step.

Detail and example: 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 both Figures 5 (a) and (b), ϕ_1 is on the edge, and ϕ_2 is on the vertex. Algorithm 1 shows this step. After we get $\Pi_{\mathit{Ref}\text{-}1}(s,t)$, we retrieve its edge sequence S based on T. The following illustrates it with an example.

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

```
Input: the rough path $\Pi_{Roug}(s,t)$ and $\zeta$ (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:         $\Pi'_{Ref\text{-}1}(v_s,v_e) \leftarrow EdgSeqConv\ (\Pi'_{Roug}(v_s,v_e), \zeta)$
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 $\zeta$ do
17:             add new Steiner points at the midpoints between $v_c$ and the nearest
                Steiner points of $v_c$ on the edges that adjacent to $v_c$
18:             $\Pi_l(v_p,v_n)$ (or $\Pi_r(v_p,v_n)$) $\leftarrow$ path passes the set of newly added Steiner
                points on the left (or 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 $\Pi_{Ref\text{-}1}(s,t)$
```

(1) *Successive point*: Lines 4-14, see Figure 5 (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 $\Pi_{\mathit{Roug}}(s,t)$ that are on the

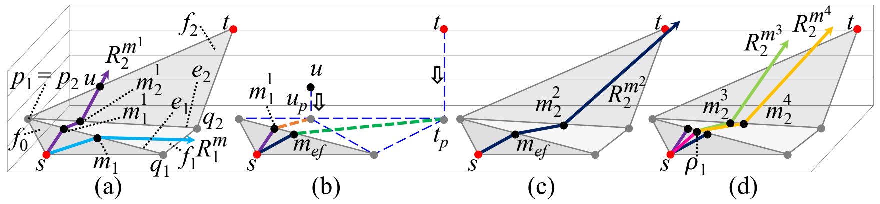

Figure 6: Snell's law path refinement step in *Ref* (a) with initial ray for calculating effective weight on the effective face $\triangle u_p p_1 q_1$, (b) for calculating m_{ef}, (c) with final ray passing through m_{ef}, and (d) processing on the remaining edges

Table 2: Datasets

Name	$\|F\|$
Original dataset	
BearHead (*BH*) [3, 34, 35]	280k
EaglePeak (*EP*) [3, 34, 35]	300k
SeaBed (*SB*) [9]	2k
CyberPunk (*CP*) [2]	2k
PathAdvisor (*PA*) [37]	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

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 $\Pi_{Roug}(v_s, v_e)$.

(2) *Single point*: Lines 15-24, see Figure 5 (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 and pink dot-dashed lines between ϕ_1 and ϕ_3 represent $\Pi_{Roug}(v_p, v_n)$, $\Pi_l(v_p, v_n)$ and $\Pi_r(v_p, v_n)$.

4.5.2 ***Snell's law path refinement***. Figure 3 (f) shows this step, and Figure 6 shows the detailed example of this step.

Detail and example: 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 6 (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 $\delta = \frac{h\epsilon w}{6lW}$. Algorithm 2 shows this step, and the following illustrates it with an example.

(1) *Binary search initial path finding*: Lines 6-20, see Figure 6 (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) *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 6 (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 6 (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 6 (b), by using the purple line $\overline{sm_1^1}$, the orange dashed line $\overline{m_1^1 u_p}$, f_0, f_{ef}, w_0 and

Algorithm 2 *SneLawRef* (s, t, δ, S)

```
Input: source $s$, destination $t$, user parameter $\delta$ 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 $|a_i b_i| \ge \delta$ do
 6:         $m_i \leftarrow$ midpoint of $[a_i, b_i]$
 7:         calculate a 3D surface Snell's ray with $\Pi_m = (root, m_i, m_{i+1}, \ldots, m_g, R_g^m)$
            with $g \le l$
 8:         if $\Pi_m$ does not pass the whole $S$, i.e., $g < l$ then
 9:             if $e_{g+1}$ is on the left side of $R_g^m$ then
10:                 $[a_i, b_i] \leftarrow [a_i, m_i], \ldots, [a_g, b_g] \leftarrow [a_g, m_g]$
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 $\Pi_m$ 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 $\Pi_{Ref\text{-}2}(s,t)$
```

Snell's law, we calculate w_{ef}. In lines 26-27 and Figure 6 (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 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 [27] 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 6 (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 $[a_2, b_2] = [m_2^1, m_2^2]$.

(3) *Binary search refined path finding*: Lines 2, 6-20, 30-31, see Figure 6 (d). Specifically, in line 6-20, we perform binary search until

$|a_1b_1| = |m_1^1 m_{ef}| < \delta$. In line 30, ρ_1 is the midpoint of $[a_1, b_1] = [m_1^1, m_{ef}]$, 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 $\Pi_{Ref\text{-}2}(s,t) = (s, \rho_1, \rho_2, t)$.

4.5.3 *Error guaranteed path refinement*.

Figures 3 (h) and (i) show this step.

Detail and example: Algorithm 3 shows this step, and the following illustrates it with an example (see Figure 3 (h)). We define $Q = \{\{u_1, dist_{Ref}(u_1)\}, \{u_2, dist_{Ref}(u_2)\}, \dots\}$ to be a priority queue that stores a set of nodes $u_i \in G_{Ref}.V$ waiting for processing. In Figure 3 (h), suppose we need to process a and c, so Q stores $\{\{a, 9\}, \{c, 8\}\}$.

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

```
Input: source $s$, destination $t$, error parameter $\epsilon$, 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 $\epsilon$, build 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 $\Pi_{Ref\text{-}3}(s,t)$
```

(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 b, we initialize $dist_{Ref}(b) = dist_{Roug}(b) = 9 + 1 \times 5 = 14$ and $prev_{Ref}(b) = prev_{Roug}(b) = a$.

(2) *Priority queue looping*: Lines 7-15. Specifically, in lines 8-9, suppose Q stores $\{\{a, 9\}, \{c, 8\}\}$, we dequeue c with a shorter distance value ($8 < 9$). In lines 12-15, one adjacent node of c is b, with the node information, we know $dist_{Ref}(b) = 14$, so $14 < dist_{Ref}(c) + w_{bc} \cdot d(b, c) = 8 + 1 \times 7 = 15$, there is no need to enqueue $\{b, dist_{Ref}(b) = 15\}$ into Q, which saves the time. But, without the node information, we just know $dist_{Ref}(b) = \infty$, and we need to enqueue $\{b, dist_{Ref}(b) = 15\}$ into Q, which increases the time.

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

4.6 Theoretical Analysis

The running time, memory usage, and error of algorithm *Roug-Ref* are in Theorem 4.1.

THEOREM 4.1. *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 (i) $O(n \log n)$ for *Roug* due to Dijkstra's algorithm on G_{Roug} with n nodes, (ii) $O(n \log n)$ for the full edge sequence conversion step in *Ref* due to Dijkstra's algorithm, (iii) $O(l)$ 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 (iv) $O(n \log n)$ 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 (i) Dijkstra's algorithm on G_{Roug} with n nodes, (ii) $O(n)$ for the full edge sequence conversion step in *Ref* due to Dijkstra's algorithm, (iii) $O(l)$ for the Snell's law path refinement step in *Ref* due to l edges in S, and (iv) $O(n)$ for the error guaranteed path refinement step in *Ref* due to Dijkstra's algorithm on G_{Ref} with n nodes. (3) For the *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 study [23]). So in the path checking step, (i) if $|\Pi_{Ref\text{-}2}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s,t)|$, we guarantee $|\Pi_{Ref\text{-}2}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, (ii) 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 study [23]). The detailed proof appears in the appendix. □

5 EMPIRICAL STUDIES

5.1 Experimental Setup

We conducted the experiments on a Linux machine with 2.67 GHz CPU and 48GB memory. All algorithms were implemented in C++. The experimental setup followed the setup used in previous studies [20, 21, 29, 34, 35].

Datasets: Following existing studies [19, 20, 35], we conducted our experiment on 17 real terrain datasets in Table 2. For *BH-small* and *EP-small* datasets, we generated them using *BH* and *EP* following the procedure in [29, 34, 35] (which creates a terrain surface with different resolutions). We generated multi-resolution of *EP* and *EP-small* datasets for scalability test. We use the slope of a face in terrain datasets as the weight of that face [19].

Algorithms: We compared our algorithm *Roug-Ref*, and the baseline algorithms with an error $(1+\epsilon)$, i.e., (1) algorithm *EdgSeq* [33], (2) the best-known algorithm *FixSP* [19, 20, 24], and (3) algorithm *LogSP-Adapt* [7] in the experiment. As we will discuss later, we also compared variations of *Roug-Ref* for the ablation study. There is no need to compare with (1) algorithm *ConWave* [31] due to its large running time and no implementation of it so far [23], (2) algorithm *LogSP* [7] since its error is larger than $(1+\epsilon)$, (3) *on-the-fly algorithms on the unweighted terrain surface* [8, 12] since they cannot be adapted to the weighted case, and (4) *oracles on the terrain surface* [15, 16, 19, 32, 34–36] since we focus on *on-the-fly* algorithm in this paper, and we have compared the algorithm *FixSP* used for pre-computing the weighted shortest paths in oracle [19].

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

Factors and Measurements: We studied three factors, namely (1) k (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 six measurements to evaluate the algorithm performance, namely (1) *preprocessing time* (i.e., the time for constructing the weighted graph using Steiner points), (2) *query time*, (3) *memory usage*, (4) *chances of using error guaranteed path refinement step*, (5) *average number of Steiner points per edge*, and (6) *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).

5.2 Experimental Results

In the experiment, we use *EdgSeq* and set $\epsilon = 0.05$ to simulate the exact weighted shortest path on datasets with less than 250k faces for measuring distance error. Since *EdgSeq* 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. The bar charts in dark and light colors refer to the measurements for (1) the first algorithm *Roug*, (2) the second algorithm *Ref*, and (3) the whole algorithm *Roug-Ref*. For the (1) total query time, (2) query time for algorithm *Roug* and (3) query time for algorithm *Ref*, the vertical bar means the minimum and maximum results.

5.2.1 ***Ablation study of Roug-Ref.*** In our algorithm *Roug-Ref*, we have 4 variations: (1) we do not use our efficient Steiner points placement scheme, i.e., we use algorithm *FixSP* for Steiner point placement, (2) we remove the full edge sequence conversion step, (3) we remove the effective weight pruning out sub-step in the Snell's law path refinement step, and (4) we do not use the node information for pruning in the error guaranteed path refinement step, for ablation study (they correspond to four techniques in Section 4.3). We use (1) *Roug-Ref-NoEffSP*, (2) *Roug-Ref-NoEdgSeqConv*, (3) *Roug-Ref-NoEffWeig*, and (4) *Roug-Ref-NoPrunDijk*, to denote these variations. We use *Roug-Ref-AllNaive* to denote the variation that does not use all of these four techniques, and it corresponds to the adapted *FixSP* and *EdgSeq* by using our rough-refine concept. There is no need to consider the case if we do not use the rough-refine concept. In this case, *Roug-Ref* becomes *LogSP-Adapt*. Since k will only affect the variations of *Roug-Ref* and our algorithm *Roug-Ref*, we study the effect of k in this subsection.

Effect of k: In Figure 7 (resp. Figure 8), we tested 5 values of k from $\{1, 2, 3, 4, 5\}$ on *BH-small* (resp. *BH*) dataset by setting ϵ to be 0.1 (resp. 0.25) for ablation study involving 6 variations of *Roug-Ref* (resp. involving 5 variations of *Roug-Ref* without *Roug-Ref-NoEffSP* and *Roug-Ref-AllNaive* due to their expensive running time). (1) The preprocessing time is independent of k, so varying k will not affect the time. (2) The overlapping lines of the preprocessing time mean that the result is the same, since the time is only affected by the Steiner point placement scheme in the algorithm. (3) When $k \leq 2$ and k increases, more Steiner points are removed and *Roug* runs faster, so the query time and memory usage of these algorithms decrease. But when $k > 2$, it has a higher chance (with a chance more than 99%) that *Ref* needs to perform the error guaranteed path refinement step, so the query time and memory usage have a sudden increase. Thus, the optimal k is 2. (4) When $k \leq 2$, the query time and memory usage for *Roug-Ref* is the same as that of *Roug-Ref-NoPrunDijk* since the error guaranteed path refinement step is not used. But, when $k > 2$, the former one's query time and memory usage is smaller due to the usage of the step. (5) When $k > 2$ (e.g., $k = 3$), the query time of *Roug-Ref* and *LogSP-Adapt* are 2030s and 2000s on *BH* dataset, respectively. 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*.

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

Effect of ϵ: In Figure 9, we tested 6 values of ϵ from $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on *EP-small* dataset. (1) When ϵ increases, the preprocessing time, query time and memory usage decrease since they are inversely proportional to ϵ. (2) *Roug-Ref* performs better than *EdgSeq*, *FixSP* and *LogSP-Adapt* in terms of query time and memory usage, and it is clearer to observe the superior performance of *Roug-Ref* when $\epsilon = 1$. This is due to the rough-refine concept (that results in a small average number of Steiner points per edge) and four novel techniques used in *Roug-Ref*. The error of all algorithms is close to 0%. (3) 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*. *EdgSeq* performs worse than *FixSP* since *EdgSeq* first uses *FixSP* and then uses Snell's law for the weighted shortest path calculation. *LogSP-Adapt* does not perform well since it does not utilize Snell's law.

Effect of dataset size: In Figure 10, we tested 5 values of dataset size from $\{10k, 20k, 30k, 40k, 50k\}$ on multi-resolution of *EP-small* datasets by setting ϵ to be 0.1. (1) When the dataset size increases, the query time and the average number of Steiner points per edge of all algorithms increase. (2) When the dataset size is 50k, *Roug-Ref*'s query time is 10^3 times, 10^3 times and 6 times smaller than that of *EdgSeq*, *FixSP* and *LogSP-Adapt*, respectively.

5.2.3 ***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. (1) *Roug-Ref* can still beat *LogSP-Adapt* in terms of the query time and memory usage. (2) When the dataset size is 5M, *Roug-Ref*'s query time is still reasonable. However, the query times for *EdgSeq* and *FixSP* are larger than 7 days, so they are excluded from the figure.

5.2.4 ***User study.*** We conducted a user study on *Path Advisor* [37] as mentioned in Section 1.1 by using *PA* dataset [37]. 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 weighted (resp. unweighted) shortest path has distance 105.8m (resp. 98.4m). We presented the figure and the path distance result to 30 users (i.e., university students), and 96.7% of users think the

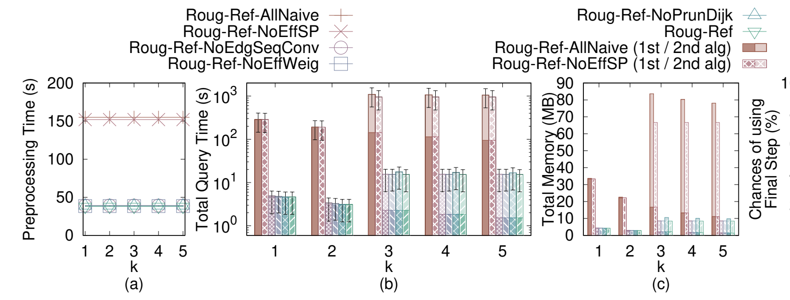

Figure 7: Ablation study (effect of k on *BH-small* dataset)

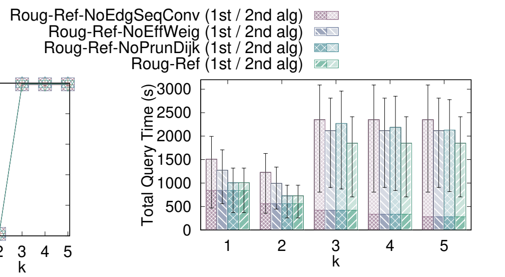

Figure 8: Ablation study (effect of k on *BH* dataset)

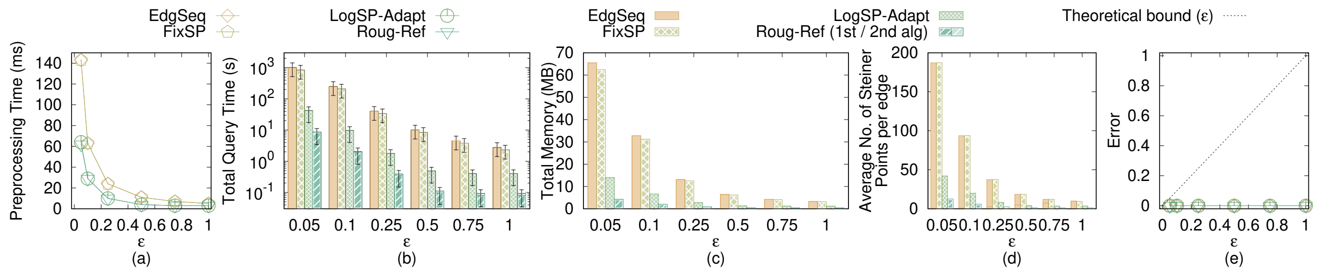

Figure 9: Baseline comparisons (effect of ϵ on *EP-small* dataset)

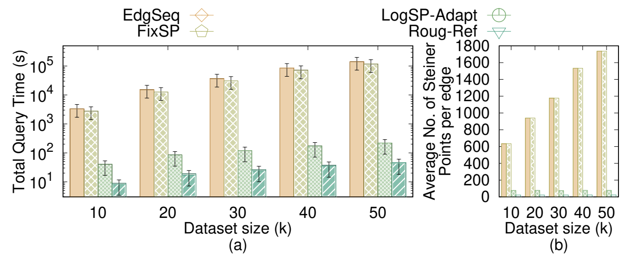

Figure 10: Baseline comparisons (effect of dataset size on multi-resolution of *EP-small* datasets)

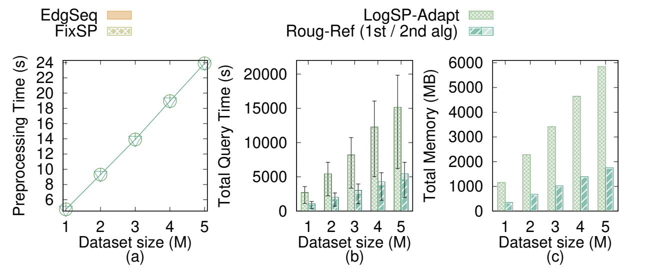

Figure 11: Scalability test

blue path is the more realistic since it maintains a safe distance from obstacles and avoids abrupt changes in direction. The average query time for the best-known algorithm *FixSP* and *Roug-Ref* are 16.62s and 0.1s, respectively.

5.2.5 ***Motivation study***. We also conducted a motivation study on the cable placement as mentioned in Section 1.1 by using *SB* dataset [9]. The average life expectancy of a cable is 25 years [28], 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) [28]. We randomly chose pairs of points as source and destination, respectively, and repeated it 100 times to calculate the path (with $\epsilon = 0.5$). In Figure 1 (c), the weighted (resp. unweighted) shortest path has distance 457.9km (resp. 438.3km). Given the cost of undersea cable, i.e., USD \$25,000/km [22], when constructing a cable that will be used for 100 years [28], the total estimated cost for the blue and dashed purple 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 best-known algorithm *FixSP* and *Roug-Ref* are 22.50s and 0.2s, respectively.

5.2.6 ***Summary***. *Roug-Ref* is up to 1630 times and 40 times better than the best-known algorithm *FixSP* in terms of running time and memory usage. When the dataset size is 50k with $\epsilon = 0.1$, *Roug-Ref*'s running time is 73s (≈ 1.2 min), and memory usage is 43MB, but *FixSP*'s running time is 119,000s (≈ 1.5 days), and memory usage is 2.9GB. The case study confirms that *Roug-Ref* is the optimal algorithm due to its fast execution time that supports real-time responses.

6 CONCLUSION

In our paper, we propose a two-step approximation algorithm for solving 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 best-known algorithm. Future work can be introducing a new pruning step (by considering other geometric information of the weighted terrain surface) to further reduce the algorithm's running time.

REFERENCES

[1] 2023. Cyberpunk 2077. https://www.cyberpunk.net

[2] 2023. Cyberpunk 2077 Map. https://www.videogamecartography.com/2019/08/cyberpunk-2077-map.html

[3] 2023. Data Geocomm. http://data.geocomm.com/

[4] 2023. Google Earth. https://earth.google.com/web

[5] 2023. Metaverse. https://about.facebook.com/meta

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

[7] Lyudmil Aleksandrov, Mark Lanthier, Anil Maheshwari, and Jörg-R Sack. 1998. An ϵ-approximation algorithm for weighted shortest paths on polyhedral surfaces. In *Scandinavian Workshop on Algorithm Theory*. Springer, 11–22.

[8] Gokhan Altintas. 2022. Finding Shortest Path on a Terrain Surface by Using Finite Element Method. *arXiv preprint arXiv:2201.06957* (2022).

[9] Christopher Amante and Barry W Eakins. 2009. ETOPO1 arc-minute global relief model: procedures, data sources and analysis. (2009).

[10] Roger Blandford and Ramesh Narayan. 1986. 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.* 310 (1986), 568–582.

[11] Christian Bueger, Tobias Liebetrau, and Jonas Franken. 2022. Security threats to undersea communications cables and infrastructure–consequences for the EU. *Report for SEDE Committee of the European Parliament, PE702* 557 (2022).

[12] Jindong Chen and Yijie Han. 1990. Shortest Paths on a Polyhedron. In *SOCG*. New York, NY, USA, 360–369.

[13] Mo Chen, Rezaul Alam Chowdhury, Vijaya Ramachandran, David Lan Roche, and Lingling Tong. 2007. Priority queues and dijkstra's algorithm. (2007).

[14] Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari, Megan Owen, and Michiel Smid. 2014. A note on the unsolvability of the weighted region shortest path problem. *Computational Geometry* 47, 7 (2014), 724–727.

[15] Ke Deng, Heng Tao Shen, Kai Xu, and Xuemin Lin. 2006. Surface k-NN query processing. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 78–78.

[16] Ke Deng, Xiaofang Zhou, Heng Tao Shen, Qing Liu, Kai Xu, and Xuemin Lin. 2008. A multi-resolution surface distance model for k-nn query processing. *The VLDB Journal* 17, 5 (2008), 1101–1119.

[17] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[18] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. 2010. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems* 57, 1 (2010), 65–100.

[19] Bo Huang, Victor Junqiu Wei, Raymond Chi-Wing Wong, and Bo Tang. 2023. EAR-Oracle: on efficient indexing for distance queries between arbitrary points on terrain surface. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[20] Manohar Kaul, Raymond Chi-Wing Wong, and Christian S Jensen. 2015. New lower and upper bounds for shortest distance queries on terrains. *Proceedings of the VLDB Endowment* 9, 3 (2015), 168–179.

[21] Manohar Kaul, Raymond Chi-Wing Wong, Bin Yang, and Christian S Jensen. 2013. Finding shortest paths on terrains by killing two birds with one stone. *Proceedings of the VLDB Endowment* 7, 1 (2013), 73–84.

[22] Jonathan Kim. 2022. Submarine cables: the invisible fiber link enabling the Internet. https://dgtlinfra.com/submarine-cables-fiber-link-internet/

[23] Mark Lanthier. 2000. *Shortest path problems on polyhedral surfaces.* Ph.D. Dissertation. Carleton University.

[24] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica* 30, 4 (2001), 527–562.

[25] Lik-Hang Lee, Tristan Braud, Pengyuan Zhou, Lin Wang, Dianlei Xu, Zijun Lin, Abhishek Kumar, Carlos Bermejo, and Pan Hui. 2021. All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda. *arXiv preprint arXiv:2110.05352* (2021).

[26] Lik-Hang Lee, Zijun Lin, Rui Hu, Zhengya Gong, Abhishek Kumar, Tangyao Li, Sijia Li, and Pan Hui. 2021. When creators meet the metaverse: A survey on computational arts. *arXiv preprint arXiv:2111.13486* (2021).

[27] Sang Han Lee, Sung Mo Im, and In Sung Hwang. 2005. Quartic functional equations. *J. Math. Anal. Appl.* 307, 2 (2005), 387–394.

[28] Jean-François Libert and Gary Waterworth. 2016. 13 - Cable technology. In *Undersea Fiber Communication Systems (Second Edition)*. Academic Press, 465–508.

[29] Lian Liu and Raymond Chi-Wing Wong. 2011. Finding shortest path on land surface. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 433–444.

[30] Born Max and Wolf Emil. 1959. Principles of optics. (1959).

[31] Joseph SB Mitchell and Christos H Papadimitriou. 1991. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM (JACM)* 38, 1 (1991), 18–73.

[32] Cyrus Shahabi, Lu-An Tang, and Songhua Xing. 2008. Indexing land surface for efficient knn query. *Proceedings of the VLDB Endowment* 1, 1 (2008), 1020–1031.

[33] Nguyet Tran, Michael J Dinneen, and Simone Linz. 2020. Close weighted shortest paths on 3D terrain surfaces. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*. 597–607.

[34] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, and David M. Mount. 2017. Distance oracle on terrain surface. In *SIGMOD/PODS'17*. New York, NY, USA, 1211–1226.

[35] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, David M Mount, and Hanan Samet. 2022. Proximity queries on terrain surface. *ACM Transactions on Database Systems (TODS)* (2022).

[36] Songhua Xing, Cyrus Shahabi, and Bei Pan. 2009. Continuous monitoring of nearest neighbors on land surface. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1114–1125.

[37] Yinzhao Yan and Raymond Chi-Wing Wong. 2021. Path Advisor: a multi-functional campus map tool for shortest path. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2683–2686.

[38] Xiaoming Zheng, Sven Koenig, David Kempe, and Sonal Jain. 2010. Multirobot forest coverage for weighted and unweighted terrain. *IEEE Transactions on Robotics* 26, 6 (2010), 1018–1031.

A SUMMARY OF FREQUENT USED NOTATIONS

Table 3 shows a summary of frequent used notations.

B 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 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.

C A2A QUERY

Apart from the V2V query that we discussed in the main body of this paper, we also present a method to answer the A2A 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 $f = \triangle v_1 v_2 v_3$ is equal to the sum of the area of $\triangle v v_1 v_2$, $\triangle v v_1 v_3$ and $\triangle v v_1 v_2 v_3$. With the definition, we are ready to introduce the adapted method to answer the A2A query in the weighted region. This adapted method is similar to the one presented in Section 4, the only difference is that if s or t is not in V, we can simply make them as vertices by adding

Table 3: Summary of frequent used notations

Notation	Meaning
T	The weighted terrain surface
$V/E/F$	The set of vertices / edges / faces of T
n	The number of vertices of T
L	The length of the longest edge in T
N	The smallest integer value which is larger than or equal to the coordinate value of any vertex
W/w	The maximum / minimum weights of T
h	The minimum height of any face in F
ϵ	The error parameter
η	The constant calculated using the remaining Steiner points for controlling the error
k	The removing value
$\Pi^*(s,t)$	The optimal weighted shortest path
r_i	The intersection point of $\Pi^*(s,t)$ and an edge in E
$\Pi(s,t)$	The final calculated weighted shortest path
$\|\Pi(s,t)\|$	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*

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 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 4.1.

D EMPIRICAL STUDIES

D.1 Experimental Results on the V2V Query

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

D.1.1 ***Ablation study of Roug-Ref.*** In our algorithm *Roug-Ref*, recall that we have five variations of algorithms, i.e., (1) *Roug-Ref-NoEffSP*, (2) *Roug-Ref-NoEdgSeqConv*, (3) *Roug-Ref-NoEffWeig*, (4) *Roug-Ref-NoPrunDijk*, and (5) *Roug-Ref-AllNavie*, for ablation study. Since k will only affect the variations of *Roug-Ref* and our algorithm *Roug-Ref*, we study the effect of k in this subsection.

Effect of k. In Figure 12, Figure 16, Figure 20 and Figure 23, we tested 5 values of k from $\{1, 2, 3, 4, 5\}$ on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting ϵ to be 0.1 on *BH-small* and *EP-small* datasets, and 0.25 on *BH* and *EP* datasets. Figure 13, Figure 17, Figure 21 and Figure 24 are the separated query time and memory usage in two steps for these results.

(1) The preprocessing time is independent of k, so varying k will not affect the time. (2) The overlapping lines of the preprocessing time mean the result is the same, since the time is only affected by *FixSP* component in the algorithm. (3) When $k \leq 2$ and k increases, more Steiner points are removed *Roug* can run faster, so the query time and memory usage of these algorithms decrease. But when $k > 2$, it has a higher chance (with a chance more than 99%) that *Ref* needs to perform the error guaranteed path refinement step, so the query time and memory usage have a sudden increase. Thus, the optimal k is 2 (the query time and memory usage of the algorithms using *Roug-Ref* framework are the smallest). (4) When $k \leq 2$, the query time and memory usage for *Roug-Ref* is the same as that of *Roug-Ref-NoPrunDijk* since the error guaranteed path refinement step is not used. But, when $k > 2$, the former one's query time and memory usage is smaller due to the usage of the step. (5) For *EP-small* dataset, when $k > 2$ (e.g., $k = 3$), the query time of *Roug-Ref* and *LogSP-Adapt* are 1330s and 1300s on *BH* dataset, respectively. 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*.

D.1.2 ***Baseline comparisons.*** From the previous subsection, *Roug-Ref* is the best algorithm among different variations. Starting

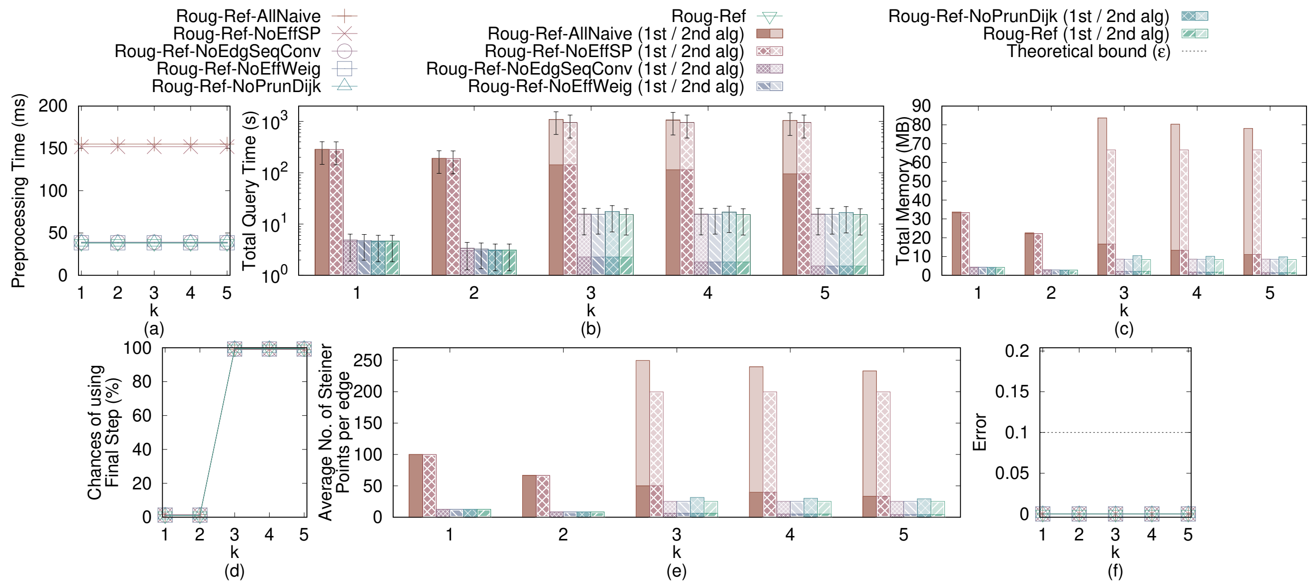
Figure 12: Ablation study (effect of k on *BH-small* dataset for the V2V query)

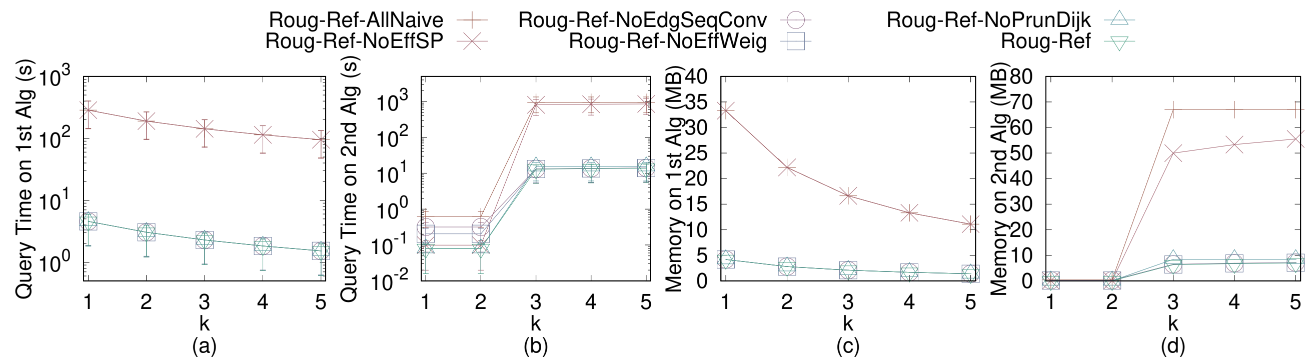
Figure 13: Ablation study (effect of k on *BH-small* dataset with separated query time and memory usage in two steps for the V2V query)

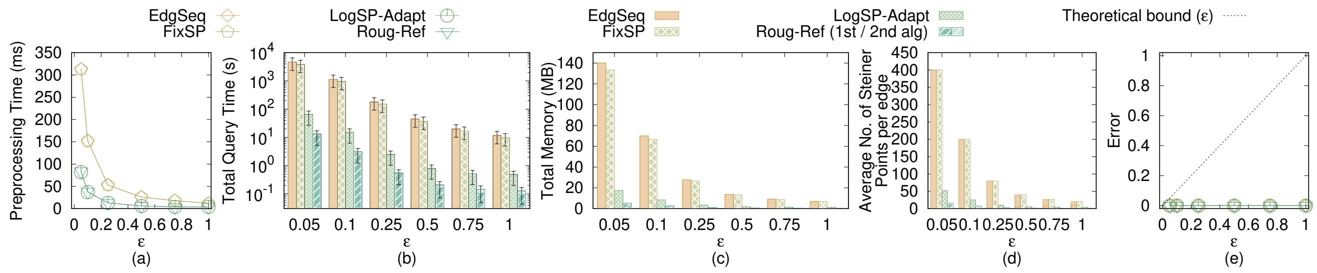
Figure 14: Baseline comparisons (effect of ϵ on *BH-small* dataset for the V2V query)

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

Effect of ϵ. In Figure 14, Figure 18, Figure 9 and Figure 25, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting k to be 2. Figure 15, Figure 19, Figure 22 and Figure 26 are the separated query time and memory usage in two steps for these results. (1) When ϵ increases, the preprocessing time, query time and memory usage decrease since they are inversely proportional to ϵ. (2) *Roug-Ref* performs better than *EdgSeq*, *FixSP* and *LogSP-Adapt* in terms of query time

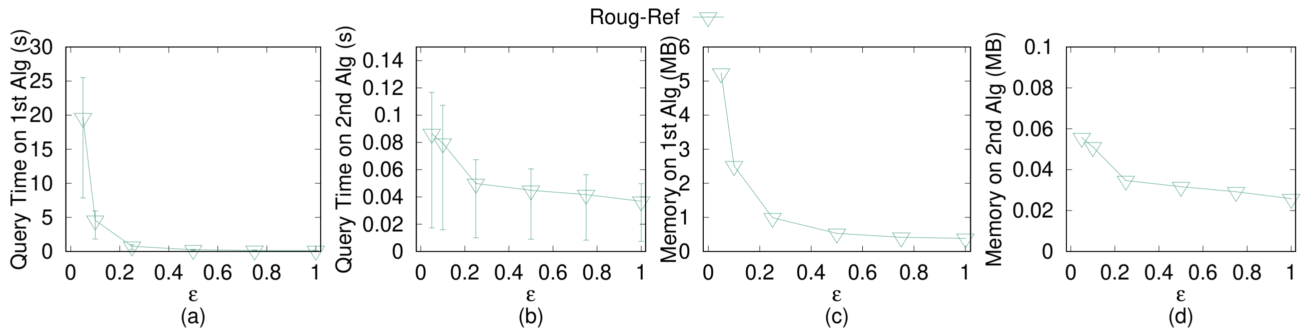

Figure 15: Baseline comparisons (effect of ϵ on *BH-small* dataset with separated query time and memory usage in two steps for the V2V query)

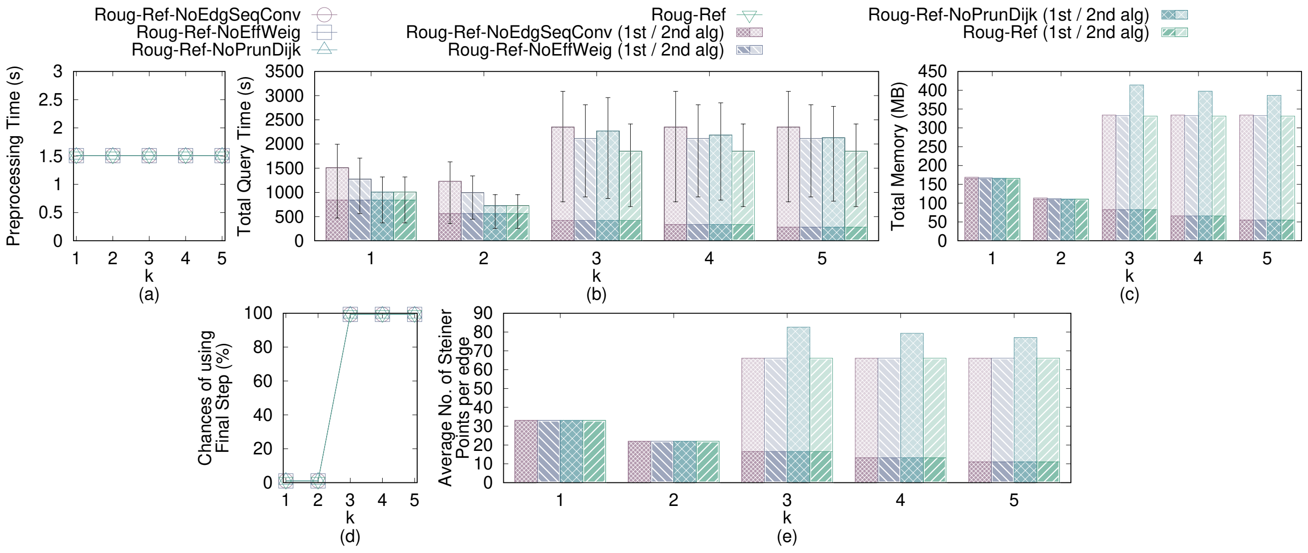

Figure 16: Ablation study (effect of k on *BH* dataset for the V2V query)

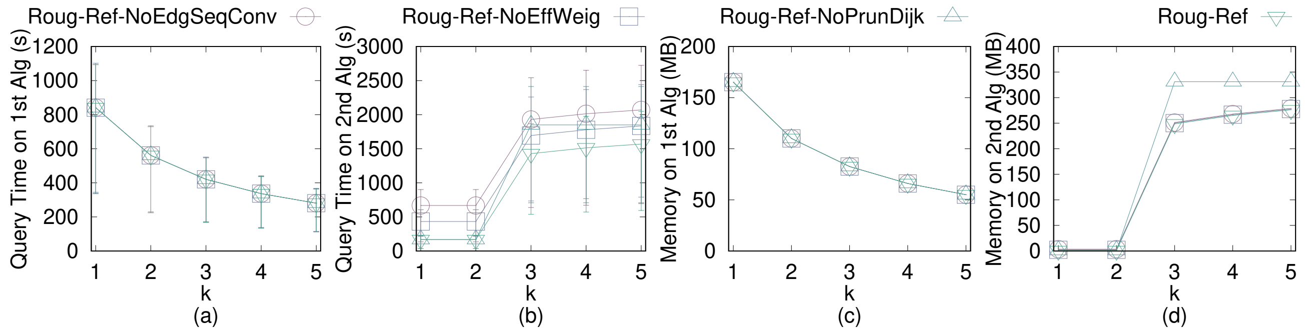

Figure 17: Ablation study (effect of k on *BH* dataset with separated query time and memory usage in two steps for the V2V query)

and memory usage, and it is clearer to observe the superior performance of *Roug-Ref* when $\epsilon = 1$. This is due to the rough-refine concept (that results in a small average number of Steiner points per edge) and four novel techniques used in *Roug-Ref*. The error of all algorithms is close to 0%. (3) When $\epsilon = 0.05$, the query time of *Roug-Ref* is only 14.6s, but the query time of *FixSP* is 23,800s (≈ 7.2 hours), i.e., 1630 times larger than that of *Roug-Ref*. *EdgSeq* performs worse than *FixSP* since *EdgSeq* first uses *FixSP* and then uses Snell's law for the weighted shortest path calculation. *LogSP-Adapt* still

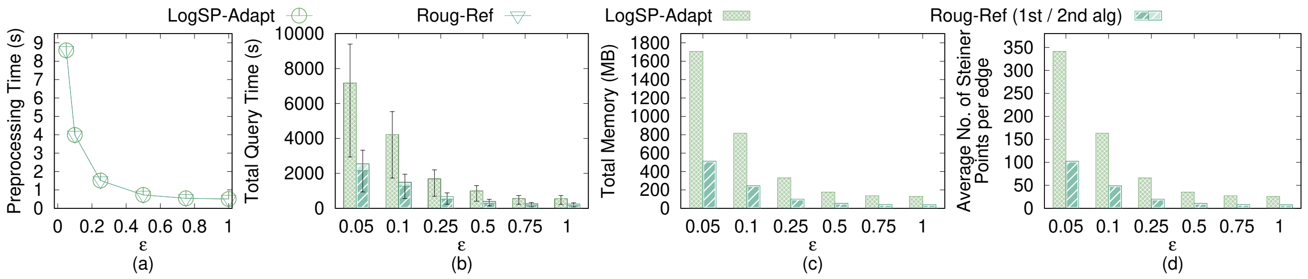

Figure 18: Baseline comparisons (effect of ϵ on *BH* dataset for the V2V query)

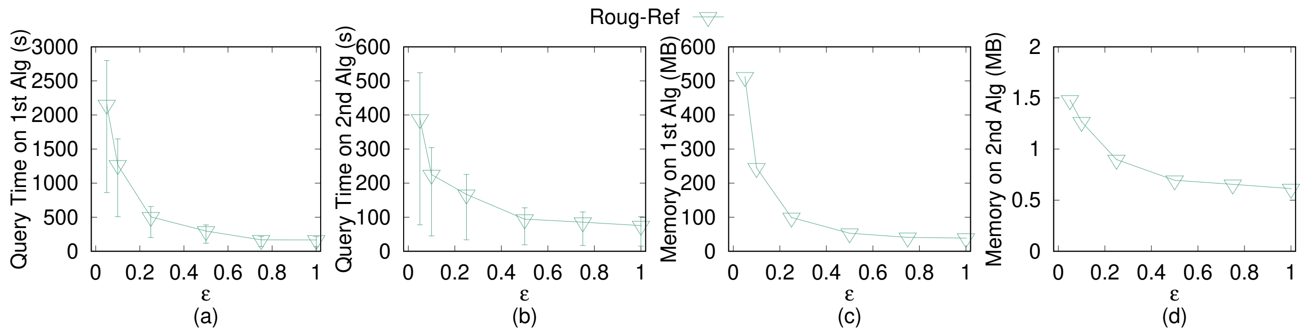

Figure 19: Baseline comparisons (effect of ϵ on *BH* dataset with separated query time and memory usage in two steps for the V2V query)

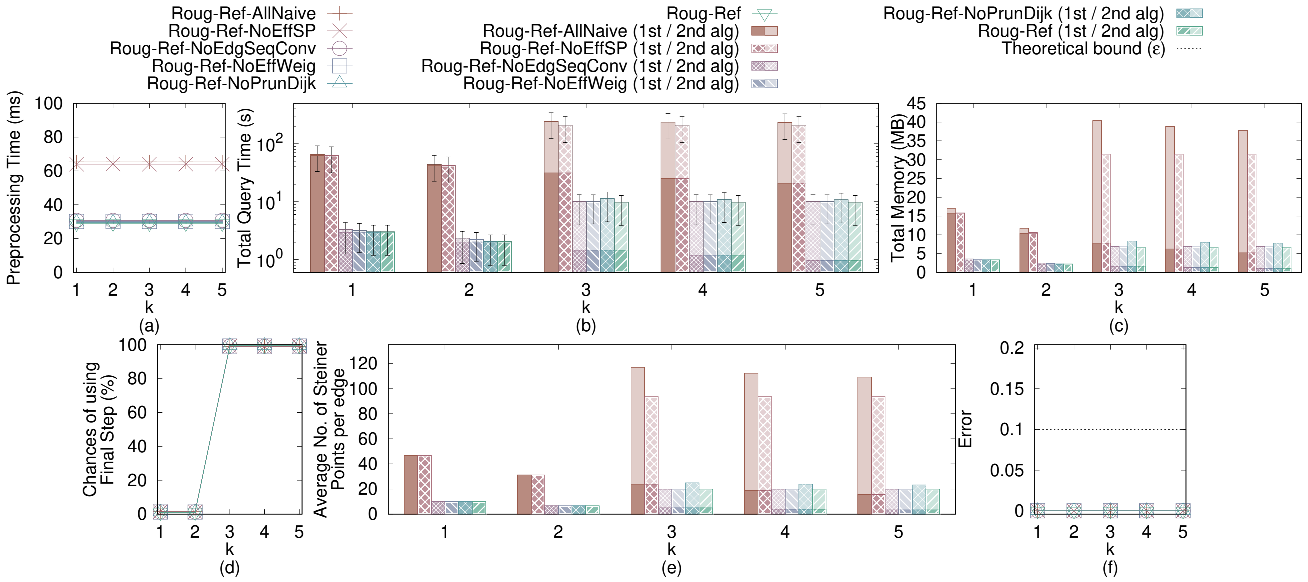

Figure 20: Ablation study (effect of k on *EP-small* dataset for the V2V query)

performs badly in terms of query time and memory usage since it does not utilize Snell's law.

Effect of dataset size: In Figure 27, 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. Figure 28 is the separated query time and memory usage in two steps for these results. *Roug-Ref* performs better than *FixSP* and *LogSP*. (1) When the dataset size increases, the query time and the average number of Steiner points per edge of all algorithms increase. (2) *Roug-Ref* has the smallest query time. When the dataset size is 50k, *Roug-Ref*'s query time is 10^3 times, 10^3 times and 6 times smaller than that of *EdgSeq*, *FixSP* and *LogSP-Adapt*, respectively.

D.1.3 ***Scalability test.*** In Figure 29, we tested 5 values of dataset size from {1M, 2M, 3M, 4M, 5M} on multi-resolution of *EP* datasets

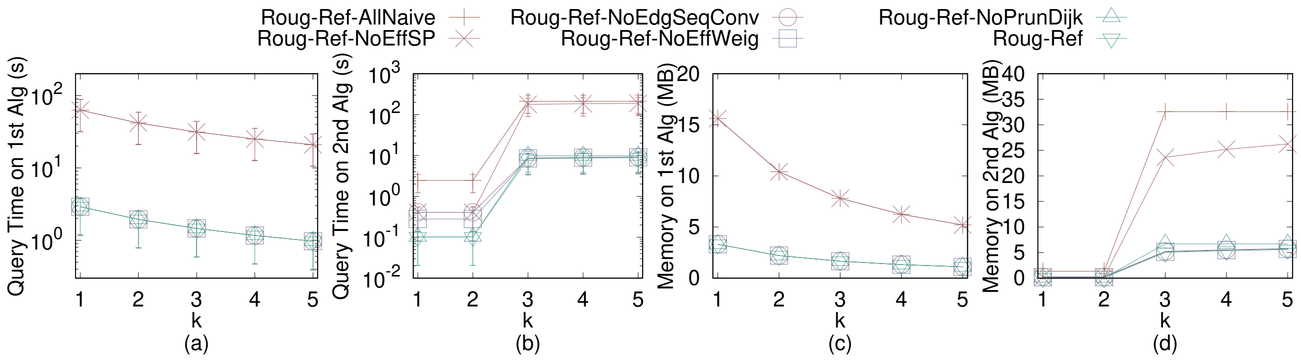

Figure 21: Ablation study (effect of k on *EP-small* dataset with separated query time and memory usage in two steps for the V2V query)

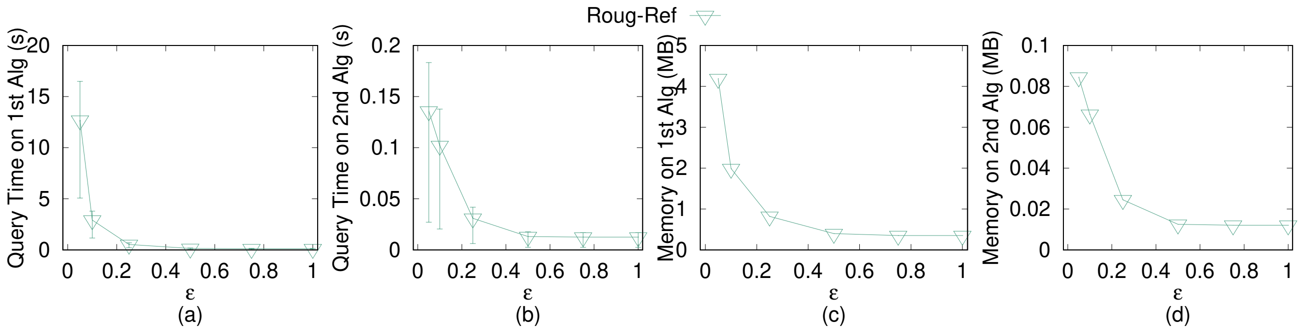

Figure 22: Baseline comparisons (effect of ϵ on *EP-small* dataset with separated query time and memory usage in two steps for the V2V query)

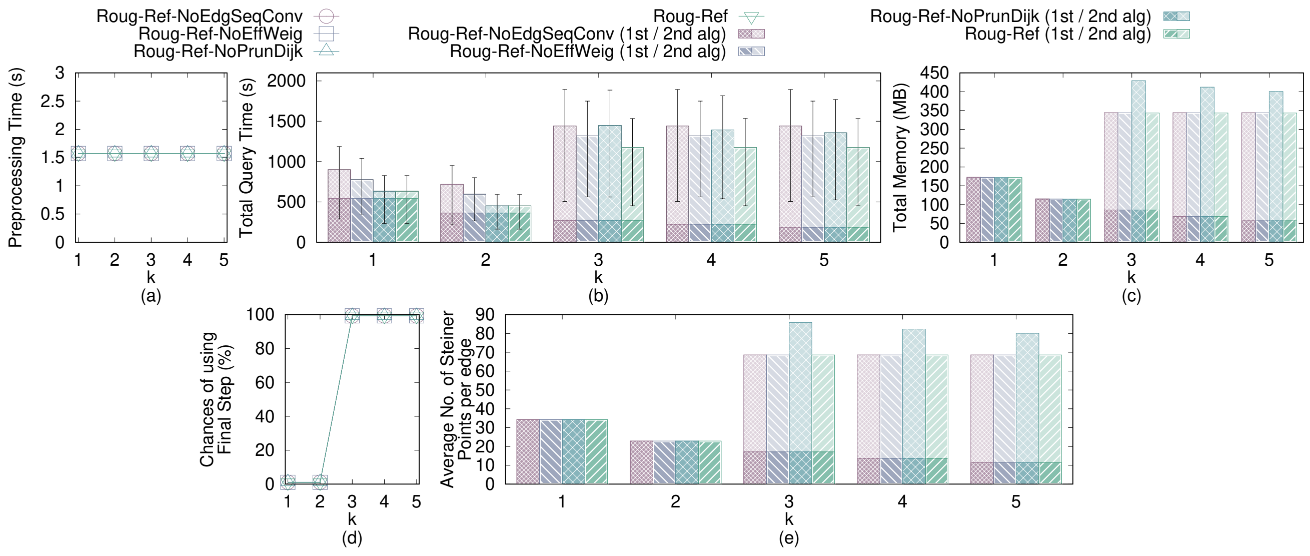

Figure 23: Ablation study (effect of k on *EP* dataset for the V2V query)

by setting ϵ to be 0.25 and k to be 2. Figure 30 is the separated query time and memory usage in two steps for these results. (1) *Roug-Ref* can still beat *LogSP-Adapt* in terms of the query time and memory usage. (2) When the dataset size is 5M, *Roug-Ref*'s query time is still reasonable. However, the query times for *EdgSeq* and *FixSP* are larger than 7 days, so they are excluded from the figure.

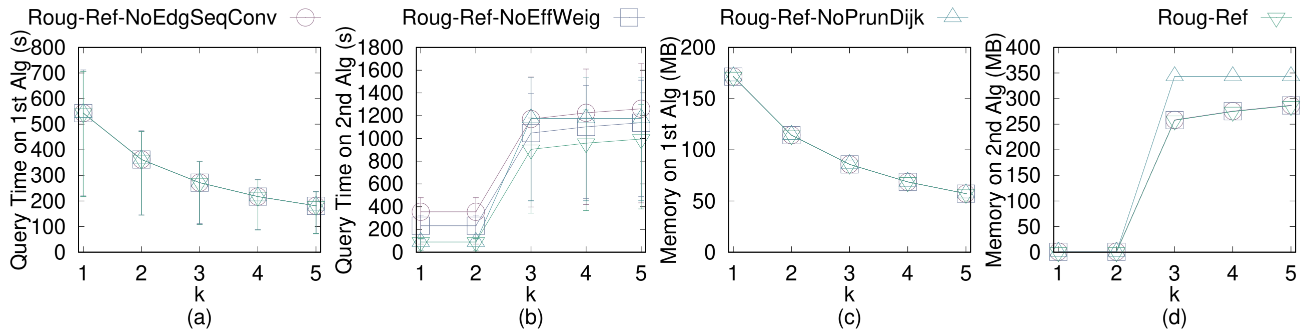

Figure 24: Ablation study (effect of k on *EP* dataset with separated query time and memory usage in two steps for the V2V query)

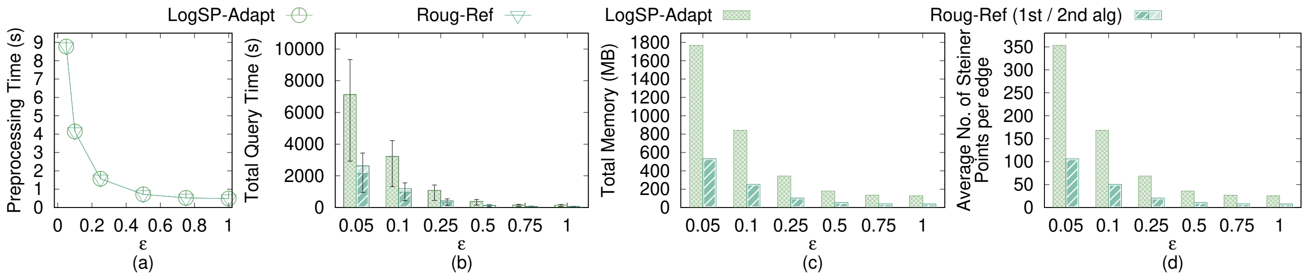

Figure 25: Baseline comparisons (effect of ϵ on *EP* dataset for the V2V query)

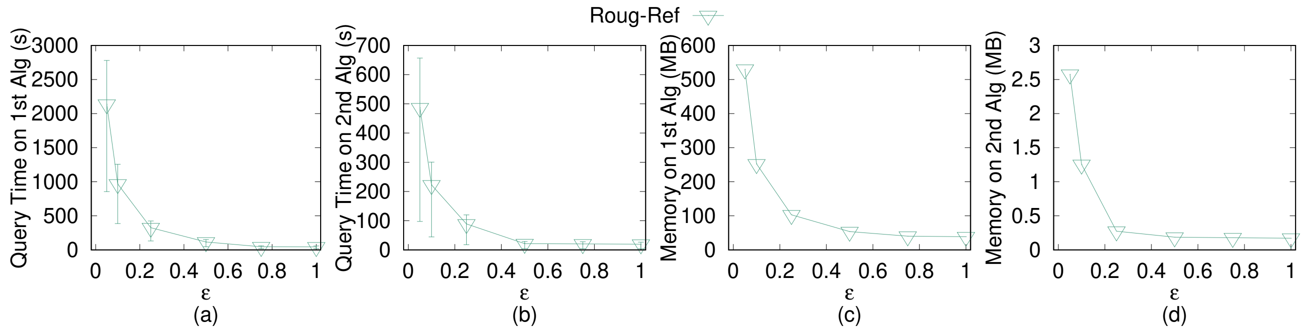

Figure 26: Baseline comparisons (effect of ϵ on *EP* dataset with separated query time and memory usage in two steps for the V2V query)

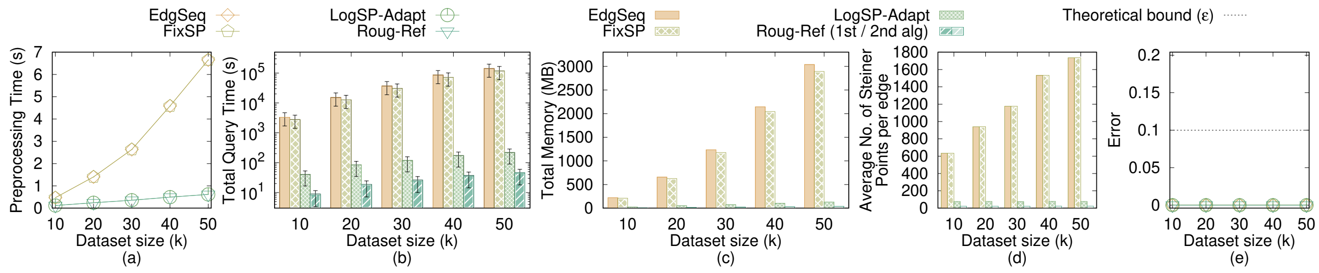

Figure 27: Baseline comparisons (effect of dataset size on multi-resolution of *EP-small* datasets for the V2V query)

D.2 Experimental Results on the A2A Query

In Figure 31 and Figure 32, we tested the A2A query by varying ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} and setting removing value to be 2 on *EP-small* dataset. In Figure 33 and Figure 34, we tested the A2A query by varying ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} and setting removing value to be 2 on *EP* dataset. Similar to the V2V query,

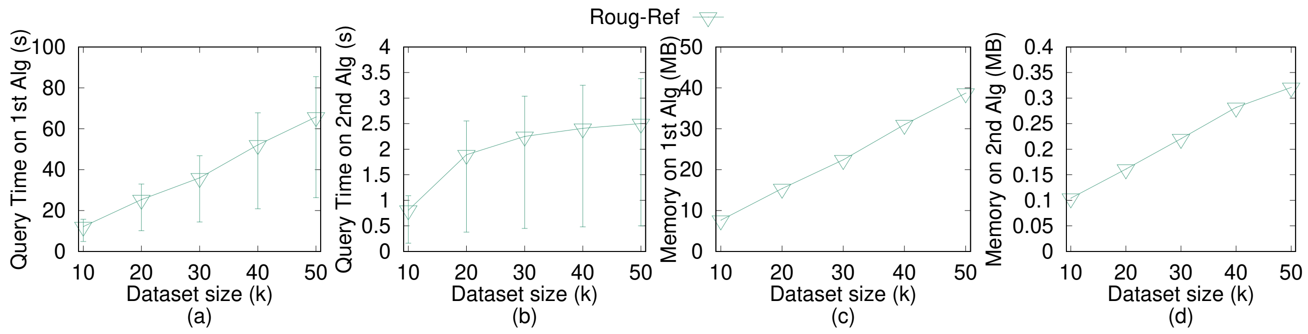

Figure 28: Baseline comparisons (effect of dataset size on multi-resolution of *EP-small* datasets with separated query time and memory usage in two steps for the V2V query)

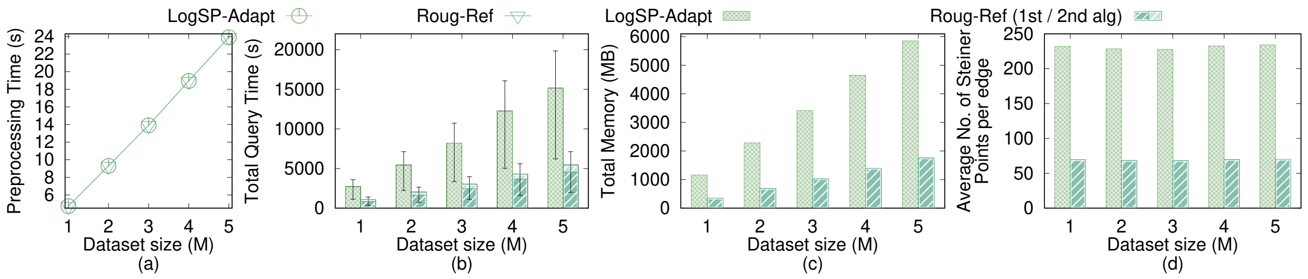

Figure 29: Scalability test (effect of dataset size on multi-resolution of *EP* datasets for the V2V query)

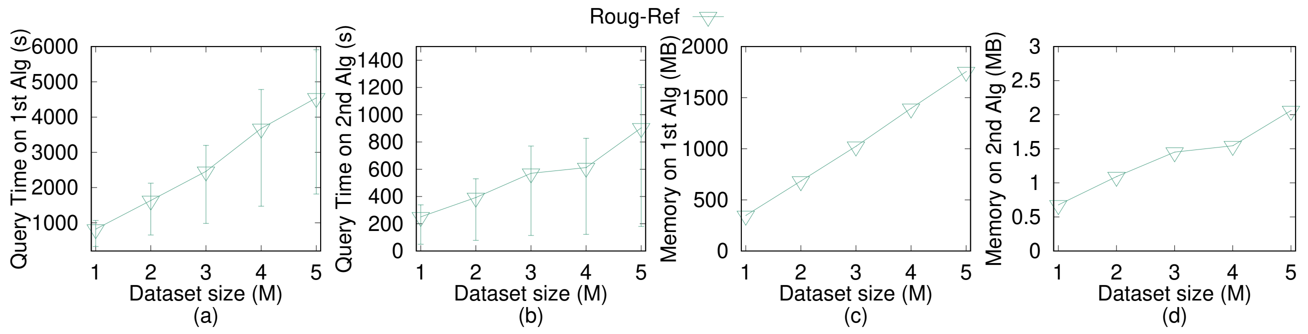

Figure 30: Scalability test (effect of dataset size on multi-resolution of *EP* datasets with separated query time and memory usage in two steps for the V2V query)

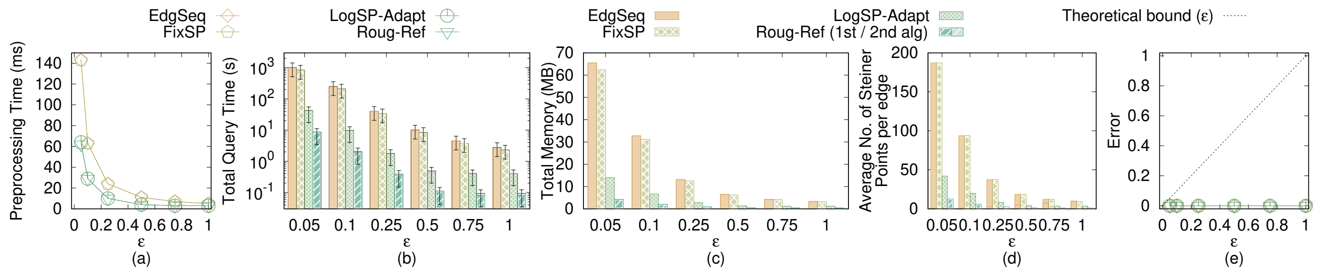

Figure 31: A2A query on *EP-small* dataset

for the A2A query, *Roug-Ref* still performs better than baseline algorithms in terms of query time. This is because the A2A query is very similar to the V2V query.

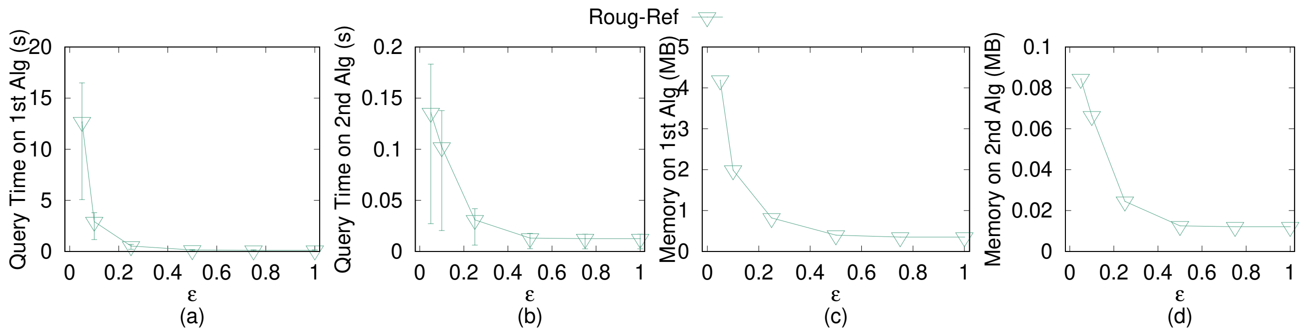

Figure 32: A2A query on *EP-small* dataset with separated query time and memory usage in two steps

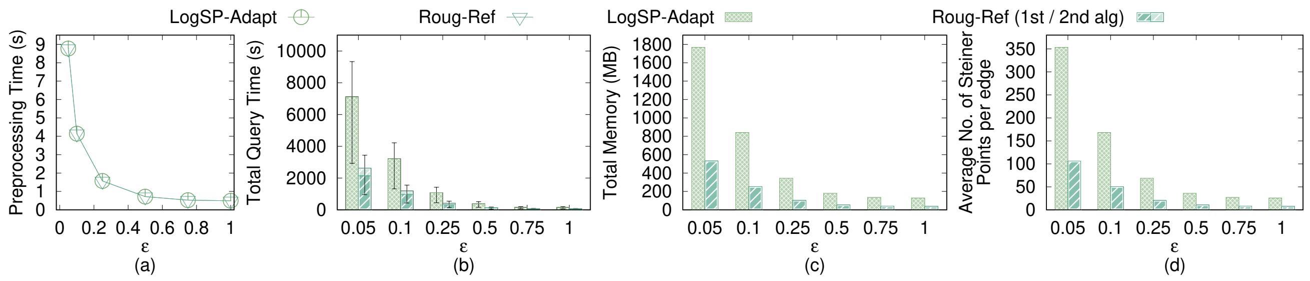

Figure 33: A2A query on *EP* dataset

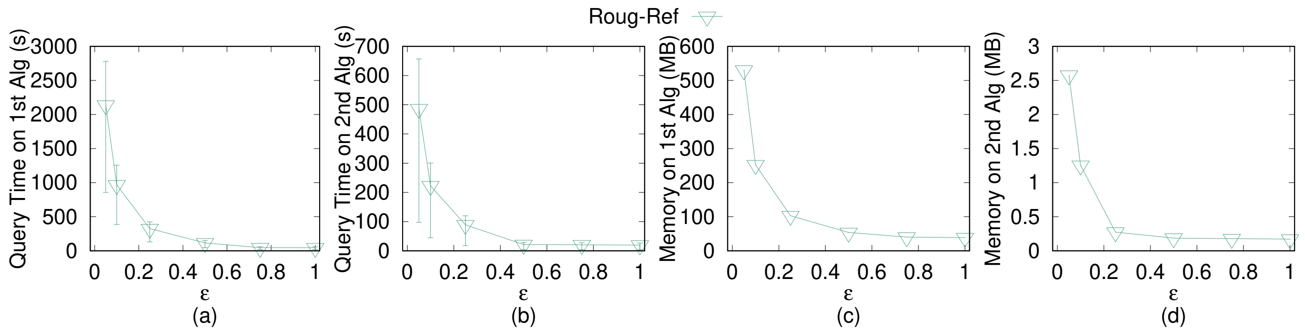

Figure 34: A2A query on *EP* dataset with separated query time and memory usage in two steps

D.3 Experimental Results on User Study (Path Advisor)

Figure 35 and Figure 36 show the result for Path Advisor user study when varying ϵ. Our user study in Section 5.2.4 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 5.2.4 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.4 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 to calculate the path. Figure 37 and Figure 38 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.

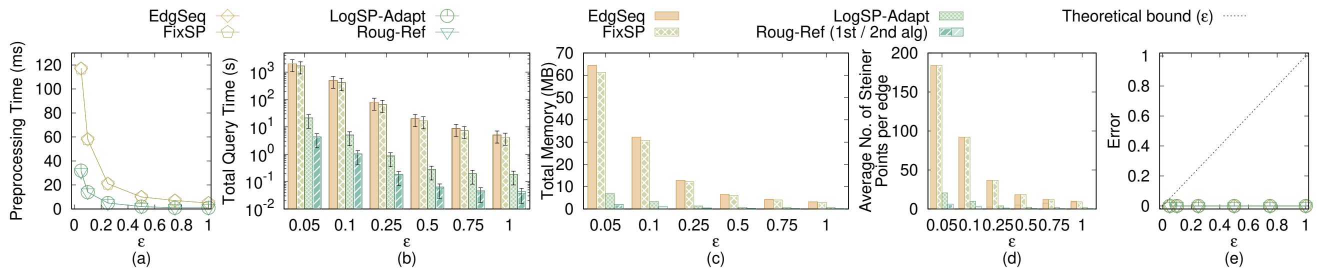

Figure 35: Path Advisor user study (effect of ϵ on *PA* dataset)

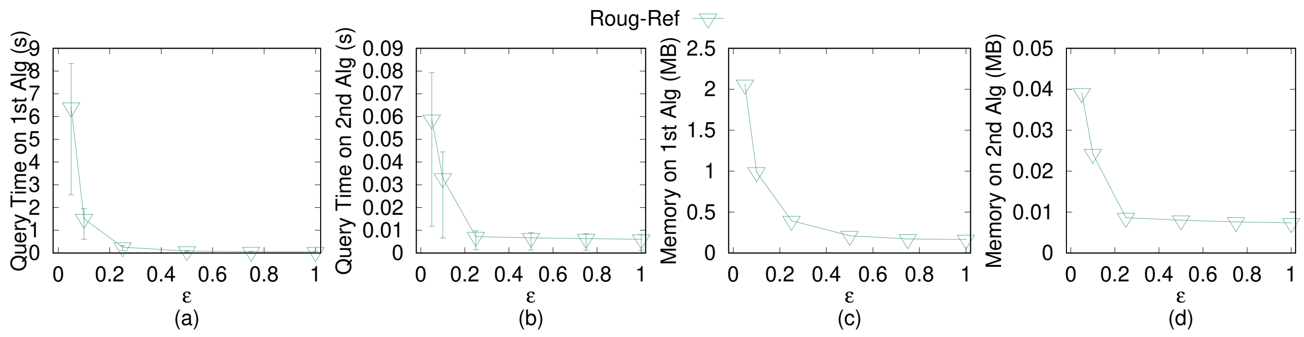

Figure 36: Path Advisor user study (effect of ϵ on *PA* dataset with separated query time and memory usage in two steps)

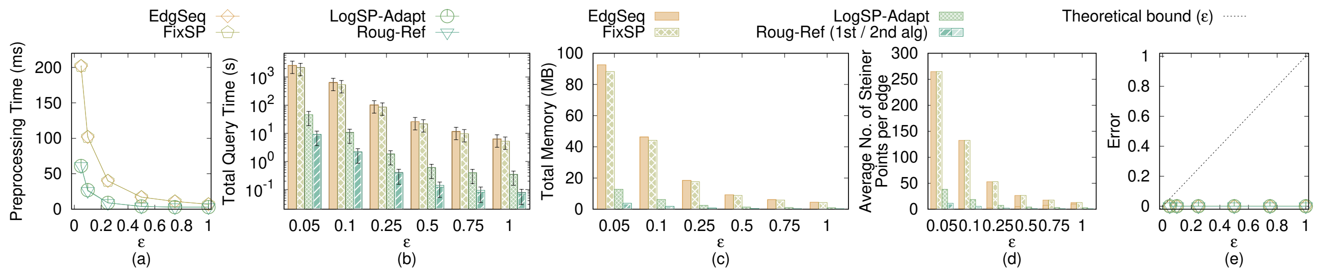

Figure 37: Cyberpunk 2077 user study (effect of ϵ on *CP* dataset)

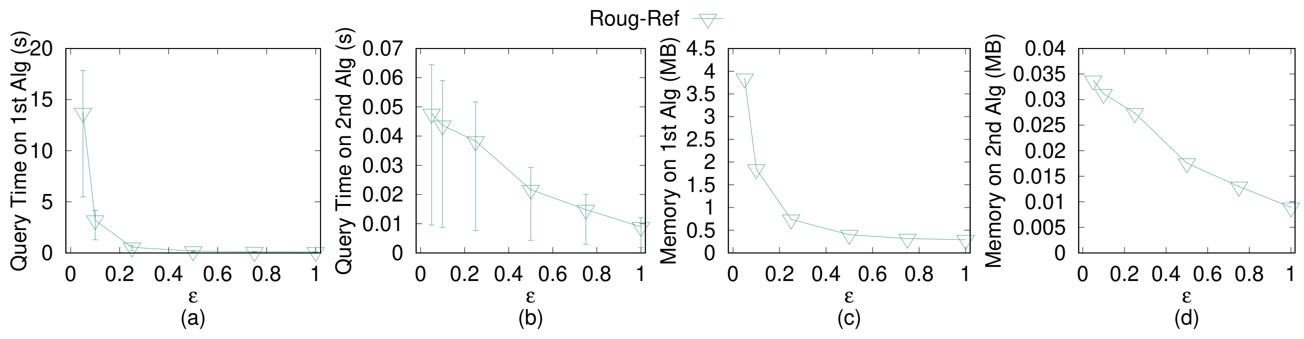

Figure 38: Cyberpunk 2077 user study (effect of ϵ on *CP* dataset with separated query time and memory usage in two steps)

D.5 Experimental Results on Motivation Study

Figure 39 and Figure 40 show the result for seabed motivation study when varying ϵ. Our motivation study in Section 5.2.5 has already shown that the blue path (i.e., the weighted shortest path) is the most realistic one since it can avoid the regions with higher hydraulic pressure, and thus, can reduce the construction cost of undersea optical fiber cable. *Roug-Ref* still has the smallest query time and memory usage.

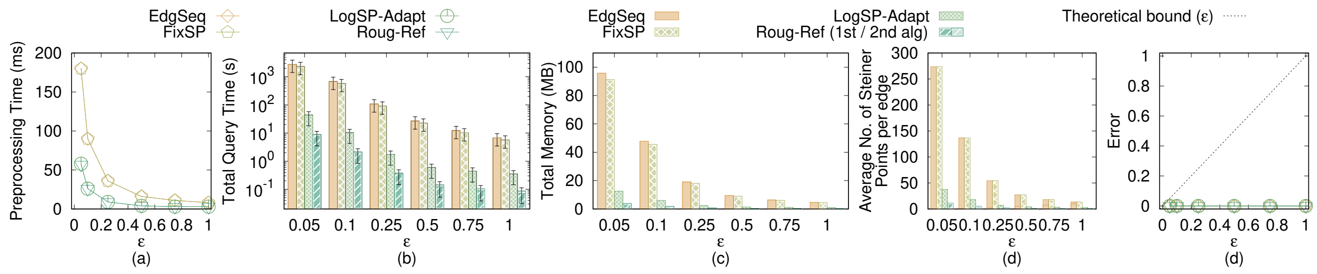

Figure 39: Seabed motivation study (effect of ϵ on *SB* dataset)

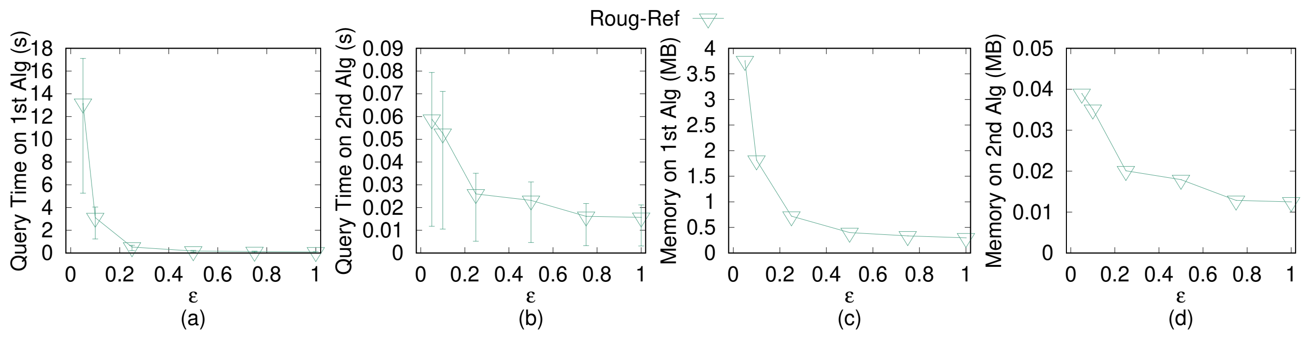

Figure 40: Seabed motivation study (effect of ϵ on *SB* dataset with separated query time and memory usage in two steps)

D.6 Generating Datasets with Different Dataset Sizes

The procedure for generating the datasets with different dataset sizes is as follows. We mainly follow the procedure for generating datasets with different dataset sizes in the study [29, 34, 35]. 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 projected point (also add edges between neighbours of two points to form edges and faces) as the newly generate T_t.

E COMPARISON OF ALL ALGORITHMS

Table 4 shows a comparison of all algorithms.

F PROOFS

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

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

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

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

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

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

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

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

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

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

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

Table 4: Comparison of all algorithms

Algorithm	**Time**		**Size**		**Error**
ConWave [31]	$O(n^8 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n^4)$	Large	$1+\epsilon$
EdgSeq [33]	$O(n^3 \log n + n^4 \log(\frac{n^2NWL}{w\epsilon}))$	Large	$O(n^3)$	Large	$1+\epsilon$
FixSP [19, 20, 24]	$O(n^3 \log n)$	Large	$O(n^3)$	Large	$1+\epsilon$
LogSP [7]	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	Medium	$O(n \log \frac{LN}{\epsilon})$	Medium	$1+(2+\frac{2W}{(1-2\epsilon)\cdot w})\epsilon$
LogSP-Adapt [7]	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	Medium	$O(n \log \frac{LN}{\epsilon})$	Medium	$1+\epsilon$
Roug-Ref-AllNaive	$O(n^3 \log n + nl^2 \log(\frac{lNWL}{w\epsilon}))$	Large	$O(n^3 + nl)$	Large	$1+\epsilon$
Roug-Ref-NoEffSP	$O(n^2 \log n + l)$	Large	$O(n^2 + l)$	Large	$1+\epsilon$
Roug-Ref-NoEdgSeqConv	$O(n \log n + nl)$	Medium	$O(nl)$	Small	$1+\epsilon$
Roug-Ref-NoEffWeig	$O(n \log n + l^2 \log(\frac{lNWL}{w\epsilon}))$	Medium	$O(n + l)$	Small	$1+\epsilon$
Roug-Ref-NoPrunDijk	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$	Medium	$O(n \log \frac{LN}{\epsilon} + l)$	Medium	$1+\epsilon$
***Roug-Ref* (ours)**	$\boldsymbol{O(n \log n + l)}$	**Small**	$\boldsymbol{O(n + l)}$	**Small**	$\boldsymbol{1+\epsilon}$

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

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

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

Theorem F.4. *The running time for algorithm Roug is $O(n \log n)$ and the memory usage is $O(n)$.*

Proof. Originally, if we do not remove Steiner points in algorithm *Roug*, i.e., we place Steiner point based on ϵ, then following Lemma F.1, the number of Steiner points k_{SP} on each edge is $O(\log_\lambda \frac{L}{r})$, where $\lambda = (1 + \frac{1+\epsilon+\frac{W}{w} - \sqrt{(1+\epsilon+\frac{W}{w})^2 - 4\epsilon}}{4} \cdot \sin\theta)$ and $r = \frac{1+\epsilon+\frac{W}{w} - \sqrt{(1+\epsilon+\frac{W}{w})^2 - 4\epsilon}}{4} \cdot h$. Following Lemma F.2 and Lemma F.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 F.5. *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. Firstly, we prove the *running time*. Given $\Pi_{Roug}(s,t)$, there are three cases on how to apply the full edge sequence conversion step in algorithm *Ref* on $\Pi_{Roug}(s,t)$, i.e., (1) some segments of $\Pi_{Roug}(s,t)$ passes on the edges (i.e., no need to use algorithm *Ref* full edge sequence conversion step), (2) some segments of $\Pi_{Roug}(s,t)$ belongs to single endpoint case, and (3) some segments of $\Pi_{Roug}(s,t)$ belongs to successive endpoint case. For the first case, there is no need to case about it. For the second case, we just need to add more Steiner points on the edges adjacent to the vertices passed by $\Pi_{Roug}(s,t)$, and using Dijkstra's algorithm to refine it, and the running time is the same as the one in algorithm *Roug*, which is $O(n \log n)$. For the third case, we just need to add more Steiner points on the edge adjacent to the vertex passed by $\Pi_{Roug}(s,t)$, and find a shorter path by running for ζ times, and there are at most $O(n)$ such vertices, so the running time is $O(\zeta n) = O(n)$. Therefore, the running time for the full edge sequence conversion step in algorithm *Ref* is $O(n \log n)$.

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

Theorem F.6. *The running time for the Snell's law path refinement step in algorithm Ref is $O(l)$, and the memory usage is $O(l)$.*

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

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

Proof of Theorem 4.1. 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 F.4, Theorem F.5 and Theorem F.6, we obtain the total running time $O(n \log n + l)$. (2) In some special cases, we need to use the error guaranteed path refinement step in algorithm *Ref* for error guarantee. The sum of the running time of algorithm *Roug* and the error guaranteed path refinement step in algorithm *Ref* is exactly the same as the running time that we perform Dijkstra's algorithm on the weighted graph G_{Ref} constructed by the original Steiner points (i.e., $k_{SP} = O(\log \frac{LN}{\epsilon})$ Steiner points per edge) and V, which is $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$. But the constant term $O(\log \frac{LN}{\epsilon})$ is not important and can be omitted, so we obtain the total running time $O(n \log n + l)$. (3) In general, the total running time is $O(n \log n + l)$.

Secondly, we prove the *total memory usage*. (1) In most of the cases, there is no need to use the error guaranteed path refinement step in algorithm *Ref*. In this case, the total memory usage is the sum of the memory usage using algorithm *Roug* and the first three steps in algorithm *Ref*. From Theorem F.4, Theorem F.5 and Theorem F.6, we obtain the average case total memory usage $O(n + l)$. (2) In some cases, we need to use the error guaranteed path refinement step in algorithm *Ref* for error guarantee. The sum of the memory usage of algorithm *Roug* and the error guaranteed path refinement step in algorithm *Ref* is exactly the same as the memory usage that we perform Dijkstra's algorithm on the weighted graph G_{Ref} constructed by the original Steiner points (i.e., $k_{SP} = O(\log \frac{LN}{\epsilon})$ Steiner points per edge) and V, which is $O(n \log \frac{LN}{\epsilon} + l)$. But the constant term $O(\log \frac{LN}{\epsilon})$ is not important and can be omitted, so we obtain the total memory usage $O(n + l)$. (3) In general, the total memory usage is $O(n + l)$.

Finally, we prove the *error bound*. Recall one baseline algorithm *LogSP*. We define the path calculated by algorithm *LogSP* between s and t to be $\Pi_{LogSP}(s,t)$. The study [7, 23] show that $|\Pi_{LogSP}(s,t)| \leq (1 + (2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon')|\Pi^*(s,t)|$. A proof sketch appears in Theorem 1 of study [7] and a detailed proof appears in Theorem 3.1 of study [23]. We adapt algorithm *LogSP* to be algorithm *LogSP-Adapt* by substituting $(2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = \epsilon$ with $0 < \epsilon' < \frac{1}{2}$ and $\epsilon > 0$, we have $|\Pi_{LogSP\text{-}Adapt}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$ where $\epsilon > 0$ and $\Pi_{LogSP\text{-}Adapt}(s,t)$ is the path result returned by algorithm *LogSP-Adapt*. Recall that algorithm *LogSP-Adapt* corresponds to the efficient Steiner point placement scheme in algorithm *Roug*. This error bound is always true no matter whether the edge sequence passed by $\Pi_{LogSP\text{-}Adapt}(s,t)$ 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-Adapt*, we have $|\Pi_{Roug}(s,t)| \leq (1 + \eta\epsilon)|\Pi^*(s,t)|$. Next, in the path checking step of algorithm *Ref*, if $|\Pi_{Ref\text{-}2}(s,t)| \leq \frac{(1+\epsilon)}{(1+\eta\epsilon)}|\Pi_{Roug}(s,t)|$, 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-Adapt*, 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 F.7. *The running time for algorithm FixSP is $O(n^3 \log n)$, and 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. 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 [24]. Since there are total n edges, there are total $O(n^3)$ Steiner points in the weighted graph, so the running time and memory usage of using Dijkstra's algorithm, i.e., algorithm *FixSP*, on this graph are $O(n^3 \log n)$ and $O(n^3)$.

Secondly, we prove the *error bound*. Theorem 2.6 of study [23] shows that $|\Pi_{FixSP}(s,t)| \leq (1 + \epsilon)|\Pi^*(s,t)|$. □

Theorem F.8. *The running time for algorithm LogSP is $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$, the memory usage is $O(n \log \frac{LN}{\epsilon})$. It guarantees that $|\Pi_{LogSP}(s,t)| \leq (1+(2+\frac{2W}{(1-2\epsilon)\cdot w})\epsilon)|\Pi^*(s,t)|$, where $\Pi_{LogSP}(s,t)$ is the path result returned by algorithm LogSP.*

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

Secondly, we prove the *error bound*. Theorem 1 of study [7] shows that $|\Pi_{LogSP}(s,t)| \leq (1 + (2 + \frac{2W}{(1-2\epsilon)\cdot w})\epsilon)|\Pi^*(s,t)|$. □

Theorem F.9. *The running time for algorithm LogSP-Adapt is $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$, and the memory usage is $O(n \log \frac{LN}{\epsilon})$. It guarantees that $|\Pi_{LogSP\text{-}Adapt}(s,t)| \leq (1 + \epsilon)|\Pi^*(s,t)|$, where $\Pi_{LogSP\text{-}Adapt}(s,t)$ is the path result returned by algorithm LogSP-Adapt.*

Proof. Firstly, we prove both the *running time* and *memory usage*. By using algorithm *LogSP-Adapt*, 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-Adapt*, on this graph are $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$ and $O(n \log \frac{LN}{\epsilon})$.

Secondly, we prove the *error bound*. Theorem 1 of study [7] shows that $|\Pi_{LogSP}(s,t)| \leq (1 + (2 + \frac{2W}{(1-2\epsilon)\cdot w})\epsilon)|\Pi^*(s,t)|$. After substituting $(2 + \frac{2W}{(1-2\epsilon')\cdot w})\epsilon' = \epsilon$ with $0 < \epsilon' < \frac{1}{2}$ and $\epsilon > 0$, we have $|\Pi_{LogSP\text{-}Adapt}(s,t)| \leq (1 + \epsilon)|\Pi^*(s,t)|$ where $\epsilon > 0$. □

Theorem F.10. *The running time for algorithm EdgSeq is $O(n^3 \log n + n^4 \log(\frac{n^2 NWL}{w\epsilon}))$, and the memory usage is $O(n^3)$. It guarantees that $|\Pi_{EdgSeq}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{EdgSeq}(s,t)$ is the path result returned by algorithm EdgSeq.*

Proof. Firstly, we prove both the *running time*. Algorithm *EdgSeq* first uses algorithm *FixSP* to calculate an edge sequence in $O(n^3 \log n)$ time, then uses Snell's law in a binary search manner to calculate the result path in $O(l'^2 \log \frac{L}{\delta})$ time according to Lemma

2.4 of study [33], where l' is the number of edges in the edge sequence calculated by algorithm *FixSP* and $\delta = \frac{h\epsilon w}{6l'W}$. According to Lemma 7.1 of study [31], $l' = O(n^2)$. Thus, the running time of algorithm *EdgSeq* is $O(n^3 \log n + n^4 \log(\frac{n^2NWL}{w\epsilon}))$.

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

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

Theorem F.11. *The total running time for algorithm Roug-Ref-NoEffSP is $O(n^2 \log n + l)$, and the total memory usage is $O(n^2 + l)$. It guarantees that $|\Pi_{Roug\text{-}Ref\text{-}NoEffSP}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{Roug\text{-}Ref\text{-}NoEffSP}(s,t)$ is the path result returned by algorithm Roug-Ref-NoEffSP.*

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

Firstly, we prove both the *running time* and *memory usage*. From Theorem F.7, we know that by using *FixSP*, we need to place $O(n^2)$ Steiner points per edge on E [24]. 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 F.7, we know that $|\Pi_{FixSP}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{FixSP}(s,t)$ is the path result returned by algorithm *FixSP*. Since from Theorem 4.1, we also have $|\Pi_{LogSP}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{LogSP}(s,t)$ is the path result returned by algorithm *LogSP*, so by using the framework of *Roug-Ref*, we obtain the error bound of algorithm *Roug-Ref-NoEffSP*. □

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

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

For both the *total running time* and the *total memory usage*, there are total n different cases of the edge sequence that we need to perform Snell's law on, that is, we need to use the Snell's law path refinement step in algorithm *Ref* for n times, and select the path with the shortest distance. Therefore, the total running time and the total memory usage need to include the Snell's law path refinement step in algorithm *Ref* for n times. By using the framework of *Roug-Ref*, we obtain the total running time and total memory usage, i.e., $O(n \log n + nl)$ and $O(nl)$. □

Theorem F.13. *The total running time for algorithm Roug-Ref-NoEffWeig is $O(n \log n + l^2 \log(\frac{lNWL}{w\epsilon}))$, and the total memory usage is $O(n + l)$. It guarantees that $|\Pi_{Roug\text{-}Ref\text{-}NoEffWeig}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{Roug\text{-}Ref\text{-}NoEffWeig}(s,t)$ is the path result returned by algorithm Roug-Ref-NoEffWeig.*

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

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

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

Theorem F.14. *The total running time for algorithm Roug-Ref-NoPrunDijk is $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + l)$, and the total memory usage is $O(n \log \frac{LN}{\epsilon} + l)$. It guarantees that $|\Pi_{Roug\text{-}Ref\text{-}NoPrunDijk}(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, where $\Pi_{Roug\text{-}Ref\text{-}NoPrunDijk}(s,t)$ is the path result returned by algorithm Roug-Ref-NoPrunDijk.*

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

Firstly, we prove the *total running time*. (1) In most of the cases, there is no need to use the error guaranteed path refinement step in algorithm *Ref*. In this case, the total running time is the sum of the running time using algorithm *Roug* and the first three steps in algorithm *Ref*. From Theorem F.4, Theorem F.5 and Theorem F.6, 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 F.4, Theorem F.5 and Theorem F.6, we obtain the average case total memory usage $O(n+l)$. (2) In some cases, we need to use the error guaranteed path refinement step in algorithm *Ref* for error guarantee. Since we do not use the node information for pruning out in Dijkstra's algorithm in the error guaranteed path refinement step of algorithm *Ref*, we need to perform Dijkstra's algorithm on the weighted graph G_{Ref} constructed by the original Steiner points (i.e., $k_{SP} = O(\log \frac{LN}{\epsilon})$ Steiner points per edge) and V in the error guaranteed path refinement step, and the memory usage is $O(n \log \frac{LN}{\epsilon})$, so we obtain the total memory usage $O(n \log \frac{LN}{\epsilon} + l)$. (3) In general, the total memory usage is $O(n \log \frac{LN}{\epsilon} + l)$. $\square$