Finding Shortest Path on 3D Weighted Terrain Surface using Divide-and-Conquer and Effective Weight

Anonymous
Anonymous
Anonymous

Anonymous
Anonymous
Anonymous

ABSTRACT

Nowadays, the rapid development of computer graphics technology and geo-spatial positioning technology promotes the growth of using the digital terrain data. Studying queries on terrain data has aroused widespread concern in industry and academia. The shortest distance query is one major and essential query among these queries. In this paper, we propose an efficient method for the *weighted region problem* on the surface of a three-dimensional (3D) weighted terrain. Specifically, the weighted region problem aims to find 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 two-dimensional (2D) environment, there is no exact solution for the weighted region problem, we propose a $(1+\epsilon)$-approximate method to solve it on the terrain surface. We divide the problem into two steps, (1) finding a sequence of edges that the shortest path passes on the weighted terrain surface, and (2) calculating the approximate shortest path on this sequence of edges. For these two steps, we solve them using (1) algorithm *Divide-and-conquer step plus Logarithmic scheme Steiner Point placement* (*DLSP*), and (2) algorithm *Effective Weight pruning technique plus binary search Snell's Law* (*EWSL*), respectively. In both the theoretical analysis and practical analysis, our two-step algorithm result in a shorter running time and less memory usage compared with the best-known algorithm.

ACM Reference Format:
Anonymous and Anonymous. 2022. Finding Shortest Path on 3D Weighted Terrain Surface using Divide-and-Conquer and Effective Weight. In *Proceedings of 2023 International Conference on Management of Data (SIGMOD '23).* ACM, New York, NY, USA, 23 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In recent years, the digital terrain data becomes increasingly widespread in industry and academia [31] due to the rapid development of computer graphics technology and geo-spatial positioning technology. In industry, many existing commercial companies/applications, such as Metaverse [5], Cyberpunk 2077 (i.e., a popular 3D computer game) [2] and Google Earth [4], are using terrain data with different features (e.g., water and grassland) to help avatars/users reach the destination faster. In academia, researchers paid considerable attention to studying spatial queries, such as shortest path queries, on terrain datasets [14, 20, 21, 25, 30, 32, 33].

Terrain data is usually represented by a set of *faces* each of which corresponds to a triangle. Each face (or triangle) has three line segments called *edges* connected with each other at three *vertices*. The terrain data for one three-dimensional (3D) object is usually represented as a *Triangular Irregular Network* (TIN) . Figure 1 shows an example of a TIN model.

There are two distance metrics that we are interested in. The *unweighted shortest path* on a terrain refers to the shortest path between a source point s and a destination point t that passes the face on the terrain where each face weight is set to a fixed value (e.g. 1), and the *weighted shortest path* refers to the shortest path between s and t where each face has a *different* weight. In Figure 1, the yellow line is the unweighted shortest path from s to t in this TIN model, and the blue line is the weighted shortest path from s to t in this TIN model. In this paper, we focus on the weighted shortest path.

1.1 Motivation

Given a source point s and a destination point t, computing the weighted shortest path on the surface of the terrain between s and t with different meanings of the face weights is involved in numerous applications, including obstacle avoidance in path planning for autonomous vehicles, overland route-recommendation systems for human, and laying pipelines or electrical cables [10, 18, 19, 22, 28, 33, 34]. For example, in Figure 1, consider a robot moves on a 3D terrain surface which consists of different features, including water (the faces with blue color) and grassland (the faces with green color). The robot would like to move from s to t, and avoid passing through the water. We could set the terrain faces corresponding to water with a larger weight, and set the terrain faces corresponding to grassland with a smaller weight. So, the robot could avoid the water, and the path is realistic. In addition, considering a real-life example for placement of undersea optical fibre cable on the seabed, i.e., a terrain, and we aim to minimize the length of the cable for cost saving (thus, it is a shortest path query on a 3D terrain surface). For some regions with deeper sea, the hydraulic pressure is higher, and it is more expensive to build the cable. We set the terrain faces corresponding to this type of regions with a larger weight. So, we could avoid placing the cable on these regions, and the construction cost is cheaper.

1.2 Weighted Region Problem

Motivated by these, we aim to find the shortest path on a 3D terrain surface between two points passing different regions on the terrain surface and different regions are assigned with different weights,

SIGMOD '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

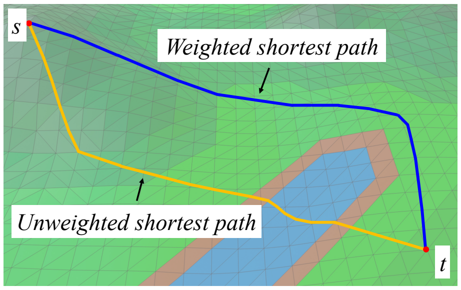

Figure 1: An example of TIN model, unweighted shortest path and weighted shortest path

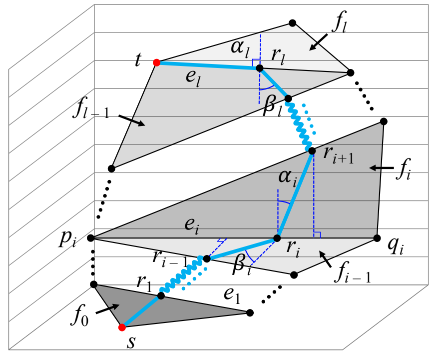

Figure 2: An example of $\Pi^*(s,t)$ in 3D that follows Snell's law

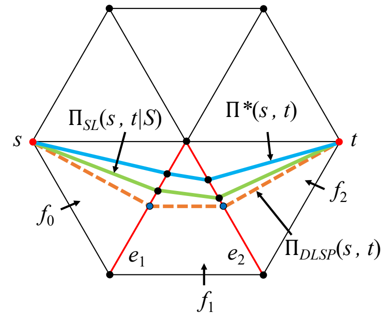

Figure 3: An example of $\Pi^*(s,t)$, $\Pi_{DLSP}(s,t)$ and $\Pi_{SL}(s,t|S)$ that passes S

and this problem is called *weighted region problem*. The weight on the 3D terrain surface is usually set according to the problem itself.

Consider a terrain T with n vertices. Let V, E, and F be the set of vertices, edges, and faces of the terrain, respectively. Snell's law of refraction from physics is one widely known fact of the weighted region problem that the weighted shortest path must obey when it passes an edge in E every time, which behaves like the refraction of a light ray when passing the boundary of two different media. Figure 2 shows an example of the weighted shortest path in 3D that follows Snell's law from s to t which passes faces $f_0, \dots, f_{i-1}, f_i, \dots, f_{l-1}, f_l$ in order. Different faces have different weights $w_0, \dots, w_{i-1}, w_i, \dots, w_{l-1}, w_l$. Let α_i be the acute angle formed by the normal of edge e_i and the out ray (i.e., $\overline{r_i r_{i+1}}$) on f_i, and β_i be the acute angle formed by the normal of edge e_i and the in ray (i.e., $\overline{r_{i-1} r_i}$) on f_{i-1}, respectively. According to Snell's law, we have $w_i \cdot \sin\alpha_i = w_{i-1} \cdot \sin\beta_i$. So the weighted shortest path (the blue line) which follows the Snell's law will bend at r_i when it crosses e_i. As a result, when the weighted shortest path passes the boundary between two faces in F with different weights, it will not result in a straight line. Due to this, in order to calculate the weighted shortest path in the weighted region problem that follows Snell's law, we need to solve two issues: (1) in which order of the sequence of edges in E that the weighted shortest path will pass (i.e., edge sequence finding), and (2) how to calculate the weighted shortest path when an order of the sequence of edges that follows Snell's law is given (i.e., edge sequence based weighted shortest path finding). We will give more details regarding why we cannot directly find the weighted shortest path that follows Snell's law in one step in Section 2.3. According to [13], there is no known algorithm for solving the weighted region problem exactly currently, and most (if not all) existing algorithms aim to calculating the weighted shortest path on the weighted region problem approximately.

Before we introduce the existing algorithms for solving the weighted region problem, we give three criteria of a good algorithm for solving the weighted region problem. Firstly, the algorithm's running time and memory usage should be reasonable (e.g., for a normal size 3D terrain dataset with 50k faces, if the algorithm's running time is less than 15 min and memory usage is less than 200MB, we say the algorithm's running time and memory usage is reasonable). Secondly, the algorithm should calculate the weighted shortest path such that this weighted shortest path needs to follow the Snell's law. This is because Snell's law is a critical optimization measurement of the weighted shortest path in the weighted region problem, and only the weighted path that follows the Snell's law is the weighted shortest path [26]. If the calculated weighted path does not follow the Snell's law, then it must exists a shorter weighted path that follows the Snell's law. Thirdly, the algorithm should provide a $(1+\epsilon)$-approximation of the weighted shortest distance within a given time limit and a given maximum memory, where ϵ is a non-negative real user parameter for controlling the error ratio, named *error parameter*.

The existing algorithms for solving the weighted region problem approximately could be divided into three categories: (1) *Wavefront Propagation plus binary search Snell's Law* (*WPSL*) approach, (2) *Steiner Points* (*SP*) approach, and (3) *Steiner Points plus binary search Snell's Law* (*SPSL*) approach. Firstly, the *WPSL* approach [26] exploits Snell's law and continuous Dijkstra algorithm to calculate the weighted shortest path that follows Snell's law. But, it violets the first criterion of a good algorithm for solving the weighted region problem. Secondly, the *SP* approach [8, 24] places discrete points (i.e., Steiner points) on edges in E, then construct a weighted graph using these Steiner points together with the original vertices to calculate the weighted shortest path (which may not follow Snell's law). But, it violets the second criterion of a good algorithm for solving the weighted region problem. Thirdly, the *SPSL* approach first uses the *SP* approach to find the edge sequence S that the weighted shortest path (which may not follow Snell's law) will pass through, then exploits Snell's law to calculate (or refine) the weighted shortest path (which follows Snell's law) on S, which is the best one among these four methods because it could get a reasonable running time and also guarantee that the calculated the weighted shortest path follow Snell's law at the passing points. But, the only one and the best-known existing work [28] violets the third criterion of a good algorithm for solving the weighted region problem. In addition, there is a large space for reducing their algorithm's running time and memory usage.

1.3 Contribution & Organization

Motivated by the drawbacks of the existing methods, in our paper, we aim to calculate the weighted shortest path in 3D weighted region problem and make our algorithm fulfill the three criteria of a good algorithm in solving the weighted region problem. We summarize our major contributions as follows.

Firstly, we propose a two-step algorithm for calculating the weighted shortest path in 3D weighted region problem using algorithm *Divide-and-conquer step plus Logarithmic scheme Steiner Point placement* (*DLSP*) and algorithm *Effective Weight pruning technique plus binary search Snell's Law* (*EWSL*), such that for a given source point s and destination point t on T, our algorithm returns a $(1+\epsilon)$-approximation of the weighted shortest distance between s and t which follows Snell's law without unfolding any face in the given terrain surface. We categorise our algorithm as the third approach as mentioned in Section 1.2 (i.e., the *SPSL* approach).

Secondly, our algorithm perform much better than the only one and the best-known existing work [28] in terms of running time (both the preprocessing time and query time) and memory usage.

Thirdly, our experimental results show that for the first step (i.e., algorithm *DLSP*) and the second step (i.e., algorithm *EWSL*), our algorithm runs up to 500 times and 20% faster than the best-known algorithm [28] on benchmark real datasets with the same error ratio (with ablation study), respectively. Specifically, for a terrain dataset with 50k faces, our algorithm's total query time is 534s ($\approx$ 9 min) and total memory usage is 130MB, but best-known existing work's [28] total query time is 119,000s ($\approx$ 1.5 day) and total memory usage is 2.9GB.

The remainder of the paper is organized as follows. Section 2 provides the preliminary. Section 3 reviews the related work. Section 4 presents our two-step algorithm for finding the weighted shortest path on 3D weighted region problem. Section 5 shows the baseline algorithms, and the comparison among our algorithm and these baseline algorithms. Section 6 presents the experimental results and Section 7 concludes the paper.

2 PRELIMINARY

2.1 Problem Definition

Consider a terrain T. Let V, E, and F be the set of vertices, edges, and faces of the terrain, respectively. Let n be the number of vertices of T (i.e., $n = |V|$) and N be the smallest integer value which is larger than or equal to the coordinate value of any vertex. Each vertex $v \in V$ has three coordinate values, denoted by x_v, y_v and z_v. Each face in F is represented by three edges in E and three vertices in V. Each face $f_i \in F$ is assigned a weight w_i, which is a positive real number. Given a face f_i, and two points p and q on f_i, we define $d(p, q)$ to be the Euclidean distance between point 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 two points s and t in V, the weighted region problem aims to find the weighted shortest path $\Pi^*(s,t) = (s, r_1, \dots, r_l, t)$, with $l \geq 0$, on the surface of T such that the weighted distance $\sum_{i=0}^{l} D(r_i, r_{i+1})$ is minimum, where each r_i for $i \in \{1, \dots, l\}$ is named as a *passing point* (or *intersection point*) in $\Pi^*(s,t)$, and it is a point on an edge in E. For simplicity, we define $|\cdot|$ to be the weighted distance of a path (e.g., $|\Pi^*(s,t)|$ is the weighted distance of the path $\Pi^*(s,t)$). The blue line in Figure 2 shows an example of $\Pi^*(s,t)$ in a 3D TIN model. Let S^* be a sequence of edges that $\Pi^*(s,t)$ from s to t need to pass in order.

Let $\Pi(s,t)$ be the final calculated weighted shortest path of our algorithm, and ϵ be a user-defined error parameter, where $\epsilon > 0$, then our algorithm guarantee that $|\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$ for any s and t in V.

2.2 Snell's law

Let $S = (e_1, \dots, e_l)$ be a sequence of edges. Given two points s and t in V, we define $\Pi^*(s,t|S) = (s, \psi_1, \dots, \psi_l, t)$ to be the optimal weighted shortest path between s and t that follows the edge sequence S. According to Lemma 3.6 in [26], $\Pi^*(s,t|S)$ is the unique weighted shortest path on S, and it obeys Snell's law at each intersection point $\psi_1, \dots, \psi_l$. Let $F(S) = (f_0, f_1, \dots, f_{l-1}, f_l)$ be a sequence of adjacent faces with respect to S such that for every f_i with $i \in \{1, \dots l-1\}$, f_i is the face containing e_i and e_{i+1} in S, while f_0 is the adjacent face of f_1 at e_1 and f_l is the adjacent face of f_{l-1} at e_l. Note that s and t are two vertices of f_0 and f_l. Let $W(S) = (w_0, w_1, \dots, w_{l-1}, w_l)$ be a weight list with respect to $F(S)$ such that for every w_i with $i \in \{0, \dots l\}$, w_i is the face weight of f_i in $F(S)$. Let $n(f_i, e_i, \psi_i)$ be the normal on face f_i that perpendicular to edge e_i at ψ_i with $i \in \{1, \dots l\}$. Given four lines $n(f_i, e_i, \psi_i)$, (ψ_i, ψ_{i+1}), $n(f_i, e_i, \psi_i)$ and (ψ_i, ψ_{i-1}), we define α_i and β_i to be two acute angles formed by the former two lines, and the latter two lines, respectively. In Figure 2, we have an edge sequence $S = (e_1, \dots, e_l)$ with the corresponding face sequence $F(S) = (f_0, \dots, f_{i-1}, f_i, \dots, f_{l-1}, f_l)$. We assume that $S = S^*$, where S^* is the edge sequence that $\Pi^*(s,t)$ passes. Thus, $\Pi^*(s,t)$ is exactly the same as $\Pi^*(s,t|S)$, and $r_i = \psi_i$ for $i \in \{1, \dots, l\}$. So the blue line represents $\Pi^*(s,t|S)$ in this example. Two acute angles α_i and β_i are denoted as dark blue curves. The Snell's law of refraction is illustrated in Proposition 2.1.

Proposition 2.1. *$\Pi^*(s,t|S)$ must obey that $w_i \cdot \sin \alpha_i = w_{i-1} \cdot \sin \beta_i$ with $i \in \{1, \dots, l\}$.*

If a point ψ_i on $\Pi^*(s,t|S)$ has $\alpha_i = 90°$ after applying Snell's law, then ψ_i is called as a *critical point* (r_l in Figure 2 is a critical point). This case happens if (ψ_i, ψ_{i-1}) comes to ψ_i with $\sin \beta_i = \frac{w_i}{w_{i-1}}$ and $w_i < w_{i-1}$. The angle $\beta_i = \sin^{-1} \frac{w_i}{w_{i-1}}$ is defined as the *critical angle* of e_i (β_l in Figure 2 is a critical angle of e_l).

2.3 Challenges

Solving 3D weighted region problem is very challenging due to the following two reasons.

Firstly, solving 3D weighted region problem is very different from calculating the unweighted shortest path in 3D. When calculating the unweighted shortest path in 3D, a popular exact solution is to unfold the 3D terrain surface into a 2D terrain, and connect the source and destination using a line segment on this 2D terrain [12]. But, in 3D weighted region problem, the weighted shortest path will bend at each crossing point to follow Snell's law. Thus, we cannot use the similar idea as in the unweighted case in solving the weighted region problem.

Secondly, we cannot directly find the weighted shortest path that follows Snell's law without knowing the edge sequence S that this path follows. It seems that given two points s and t, we can simply find the position of c (where c is on the edge opposite to s which is in the same face of s), such that the result path will follow Snell's law which starts from s, then passing c, and finally go through t, and pick the shortest one in one step without knowing S [26]. However, according to [26], it ignores the effect of critical angles and paths through vertices. When a path that follows Snell's law hits an edge at the critical angle, or hits a vertex, we no longer have complete information about where it goes next. It can travel along part of an edge and then get off at the critical angle, or it can pass through a vertex in many possible ways. However, with the given S, and two vertices s and t, by exploiting Snell's law, we can find a unique weighted shortest path such that it obeys Snell's law. So this is the reason why there are two steps in our algorithm, i.e., (1) finding S, and (2) find the weighted shortest path on S.

3 RELATED WORK

As mentioned in Section 1.2, there are three categories of algorithms for solving the weighted region problem approximately: (1) *WPSL* approach, (2) *SP* approach, and (3) *SPSL* approach.

(1) For the *WPSL* approach, it aims to calculate the weighted shortest path (which follows Snell's law) on a continuous surface by exploiting Snell's law using continuous Dijkstra [26] algorithm. The algorithm will return a $(1+\epsilon)$-approximation weighted shortest distance in $O(n^8 \log(\frac{nNW}{w\epsilon}))$ time, where n is the number of vertices in V, N is the smallest integer value which is larger than or equal to the coordinate value of any vertex, W and w are the maximum and minimum weights of T, respectively. But, the *WPSL* approach violets the first criterion of a good algorithm for solving the weighted region problem. To the best of our knowledge, there is no implementation of the *SPSL* approach so far, even the work [26] that proposes this approach does not provide an implementation of their algorithm [23].

(2) For the *SP* approach, it places discrete points (i.e., Steiner points) on edges in E. Then, these Steiner points together with the original vertices will connect with each other and a weighted graph is constructed for calculating the weighted shortest path (which may not follow Snell's law) using Dijkstra algorithm. There are different schemes for placing Steiner points, which results in different running time. Some different Steiner point placement schemes runs in $O(n^3 \log n)$ [24] and $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$ [8] time, where L is the length of the longest edge in T. A comprehensive running time analysis of different Steiner point placement schemes could be found in [11]. But, the *SP* approach violets the second criterion of a good algorithm for solving the weighted region problem. In addition, our experiment shows that for a terrain dataset with 50k faces, the algorithm in [24] runs in 118,000s ($\approx$ 1.5 day) with 2.9GB memory usage, which is very large.

(3) For the *SPSL* approach, it first uses the *SP* approach to find the edge sequence S that the weighted shortest path (which may not follow Snell's law) will pass, and then exploits Snell's law to calculate (or refine) the weighted shortest path (which follows Snell's law) on S, [28] is the only one and the best-known existing work that uses this approach. The running time for these two steps are $O(n^3 \log n)$, and $O(n^4 \log \frac{L}{\delta})$, where δ is a user-defined parameter for controlling the error (but different from the common error parameter ϵ). When δ is larger, then the refinement step which exploits Snell's law will terminate earlier, and thus, the error will become larger. Even though the work [28] is the fastest algorithm among all four types of algorithms mentioned before (based on the requirement that the weighted shortest path need follow Snell's law) and it is regarded as the best-known algorithm among all four types of algorithms for solving the weighted region problem, it violets the third criterion of a good algorithm for solving the weighted region problem. In addition, there is a large space for reducing their algorithm's running time and memory usage. Our experiment shows that for a terrain dataset with 50k faces, our algorithm's total query time is 534s ($\approx$ 9 min) and total memory usage is 130MB, but the algorithm's in [28] has total query time 119,000s ($\approx$ 1.5 day) and total memory usage 2.9GB.

Since our two-step algorithm is categorised as the *SPSL* approach, we give a major idea of the best-known existing work [28] that also belongs to the *SPSL* approach, which also involves two stpes.

In the first step, it uses algorithm *Fixed scheme Steiner Point placement* (*FSP*) to find the weighted shortest path (which may not follow Snell's law), and the edge sequence S that this calculated weighted shortest path passes will be returned, and used as the input for the second step. Specifically, it places m_f Steiner points on every edge e_i in E such that the m_f Steiner points are evenly distributed across the length of e_i. Figure 4 shows an example of algorithm *FSP* where 18 Steiner points are evenly distributed across the length of each edge on face f_i. Then, a weighted graph is constructed using the newly added Steiner points together with the original vertices in V for calculating the weighted shortest path (which may not follow Snell's law) using the Dijkstra algorithm [16] between the source point s and the destination point t. After that, the edge sequence S that this calculated weighted shortest path passes is returned, and used as the input for the second step.

In the second step, it uses algorithm *Binary Search Snell's Law* (*BSSL*) to calculate the weighted shortest path (which follows Snell's law) on S. Specifically, on the first edge e_1 in S that opposite to s, it selects the midpoint m_1 on e_1, and trace a light ray that follows Snell's law from s to m_1, then this light ray will follow S, and bend at each passing point between the ray itself and each edge in S. Next, it checks whether t is on left or right of this ray, and modify the position of m_1 to be the new midpoint of the previous m_1 and the left or right endpoint of e_1 accordingly (i.e., updating the checking interval on e_1). It iterates this procedure until (1) the light ray hits t or (2) the distance between new m_1 and previous m_1 is smaller than a user-defined parameter δ. After the processing on e_1, it continues on other edges in S until all the edges in S has been processed. The detailed description and a running example of algorithm *BSSL* will be presented in Section 4.2.1.

4 METHODOLOGY

In our two-step algorithm, given a terrain $T = (V, E, F)$, and two vertices, namely s and t, in V, we first use algorithm *DLSP* to find an edge sequence S that the weighted shortest path (which may not follow Snell's law) will pass, and then use algorithm *EWSL* to calculate the weighted shortest path that follows Snell's law on S.

Before we introduce the main idea of algorithm *DLSP* and algorithm *EWSL*, we need two more concepts. Given an edge sequence $S = (e_1, \ldots, e_l)$, a *full edge sequence*, is defined that all the length of edges in S are larger than 0. The edge sequence passed by path $(s, \sigma_1, \sigma_2, \ldots, \sigma_8, \sigma_9, t)$ in Figure 6 (a) is an example of a full edge

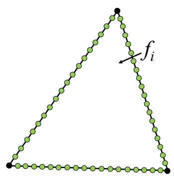

Figure 4: An example of algorithm *FSP* on f_i

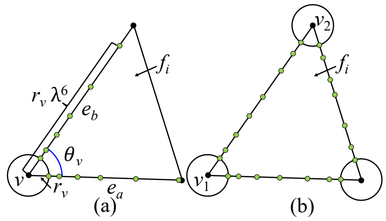

Figure 5: An example of algorithm *LSP* with Steiner points (a) on e_a and e_b, and (b) on three edges of f_i

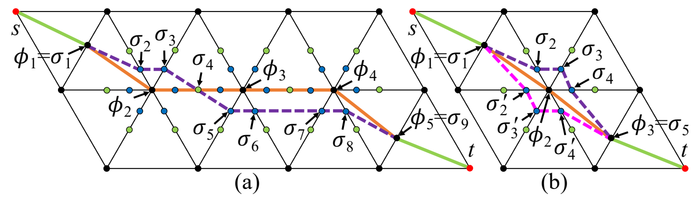

Figure 6: An example of (a) successive endpoint case, and (b) single endpoint case in algorithm *DLSP*

sequence. Similarly, given an edge sequence S, a *non-full edge sequence*, is defined that there exists at least one edge whose length is 0. The edge sequence passed by path $(s, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, t)$ in Figure 6 (a) is an example of a non-full edge sequence since the edge length at $(\overline{\phi_2\phi_2}, \overline{\phi_3\phi_3}, \overline{\phi_4\phi_4})$ is 0. With these two concepts, we are ready to introduce the main idea of algorithm *DLSP* and algorithm *EWSL*, respectively.

Main idea of algorithm *DLSP*: For algorithm *DLSP*, we place Steiner points on each edge e_i in E, then use Dijkstra algorithm [16] to find the weighted shortest path (which may not follow Snell's law) on a weighted graph constructed using the newly added Steiner points together with the original vertices in V, and retrieve an edge sequence S that this weighted shortest path (which may not follow Snell's law) passes in order. But, instead of using algorithm *FSP* [24] used in the best-known existing work [28], we use algorithm *Logarithmic scheme Steiner Point placement* (*LSP*) [8]. Compared with algorithm *FSP*, algorithm *LSP* could significantly reduce the number of Steiner points on each edge, which will reduce the algorithm's running time and memory usage significantly under the same error ratio. This is because for algorithm *FSP*, when placing Steiner points near the vertices of faces, the weighted shortest possible segment that connects two such Steiner points would become infinitesimal in length, which will require an infinite number of Steiner points. But, for algorithm *LSP*, it could have a lower bound on the length of the smallest possible edge that passes between two adjacent Steiner points, and hence, could add a finite number of Steiner points. In our experiment, for a terrain dataset with 10k faces, algorithm *FSP* needs 274 Steiner points per edge, but algorithm *LSP* only needs 40 Steiner points per edge under the same error ratio.

Nevertheless, algorithm *LSP* will cause a major problem, i.e., the edge sequence S that we retrieved from the calculated weighted shortest path (which may not follow Snell's law) of algorithm *LSP* will be a non-full edge sequence (we will discuss the reason in Section 4.1.2). In a non-full edge sequence, if an edge has length 0, it actually is a vertex. As mentioned before, when a path that follows Snell's law hits a vertex, we no longer have complete information about where it goes next because it can pass through a vertex in many possible ways. In this case, we cannot exploit Snell's law on this S to find the weighted shortest path that follows Snell's law.

Thus, in order to use algorithm *LSP* to reduce the algorithm's running time and memory usage, and to convert a non-full edge sequence to a full edge sequence (such that we can exploit Snell's law on S), we apply the divide-and-conquer step on algorithm *LSP*. Specifically, along the calculated weighted shortest path of algorithm *LSP* (which may not follow Snell's law) from source point to destination point, when we meet an edge e_i (resp. multiple edges $e_i, \ldots, e_j$) in S with length 0, i.e., when we meet a vertex v_i (resp. multiple vertices $v_i, \ldots, v_j$), we first divide the whole path into a smaller path segment that only contains v_i (resp. $v_i, \ldots, v_j$). Then, we use algorithm *LSP* again only on this path segment with more Steiner points on the edges only adjacent to e_i (resp. $e_i, \ldots, e_j$) until the edge sequence passes by this path segment is a full edge sequence. Finally, we replace the new path segment with the original path segment. We combine algorithm *LSP* and the divide-and-conquer, and thus, we have algorithm *DLSP*.

In Figure 6 (a), $(s, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, t)$ is the path result of algorithm *LSP* before the divide-and-conquer step. $(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5)$ is the smaller segment. In divide-and-conquer step, we add more Steiner points and use algorithm *LSP* again to find a new path segment $(\sigma_1, \sigma_2, \ldots, \sigma_8, \sigma_9)$ such that the edge sequence passes by this path segment is a full edge sequence. We replace the new path segment with the original path segment, so $(s, \sigma_1, \sigma_2, \ldots, \sigma_8, \sigma_9, t)$ is the result path of algorithm *DLSP*.

Main idea of algorithm *EWSL*: For algorithm *EWSL*, we follow the idea of algorithm *BSSL* as mentioned in Section 3 to calculate the weighted shortest path (which follows Snell's law) on S. Instead of directly using algorithm *BSSL*, we introduce an efficient pruning technique, such that by considering one useful information in T, named *effective weight*, we could reduce the total number of iterations for finding the optimal position of the first passing point on the first edge of S in algorithm *BSSL*, and reduce the algorithm's running time and memory usage. Specifically, in algorithm *EWSL*, we can regard all the faces except the first face in $F(S)$ as one *effective face* (where $F(S)$ is a sequence of adjacent faces with respect to S and has been defined in Section 2.2), and calculate the corresponding weight (i.e., effective weight) of this effective face. By using the effective weight and the weight of the first face, we could set up a simple quartic equation to find the position of intersection point on the first edge in S, such that this position is very close to the optimal position. If we use algorithm *BSSL*, we need to use binary search to run several iterations to find this optimal position.

In Figure 8 (a), we regard f_1 and f_2 as one effective face $\triangle u_p p_1 q_1$, and shot a initial ray (i.e., the blue line) starts from s and passes m_1 for calculating effective weight for $\triangle u_p p_1 q_1$. In Figure 8 (b), we calculate the position of m_{ef} in one simple quartic equation using the weight of f_0 and the effective weight of $\triangle u_p p_1 q_1$ (or $\triangle t_p p_1 q_1$). In Figure 8 (c), we find the final ray (i.e., the pruple line) starts from s and passes m_{ef}. Note that this ray is very close to t, which implies that m_{ef} is very close to the optimal position. In our experiment, for a terrain dataset with 150k faces, algorithm *BSSL* needs 3432 iterations to find the position of m_{ef}, but algorithm *EWSL* only needs 2853 iterations under the same error ratio. We combine algorithm *BSSL* and effective weight pruning technique, and thus, we have algorithm *EWSL*.

In the following, we present the first step of our algorithm for edge sequence finding by algorithm *DLSP* in Section 4.1, the second step of our algorithm for edge sequence based weighted shortest path finding by algorithm *EWSL* in Section 4.2, and a summary in Section 4.3.

4.1 Edge Sequence Finding

4.1.1 Algorithm LSP. In algorithm *LSP*, given a 3D terrain surface T, we aim to find an edge sequence that the weighted shortest path (which may not follow Snell's law) will pass. Specifically, we will find the weighted shortest path that may not follow Snell's law, and retrieve the edge sequence that this weighted shortest path passes. There are two steps in algorithm *LSP*, (1) placing Steiner points, and (2) applying Dijkstra algorithm on a weighted graph constructed using the newly added Steiner points together with the original vertices to calculate the weighted shortest path that may not follow Snell's law, and retrieve its edge sequence.

We give some notations first. Given two points s and t in V, we define $\Pi_{LSP}(s,t) = (s, \phi_1, \dots, \phi_l, t)$ to be the calculated weighted shortest path (which may not follow Snell's law) between s and t using algorithm *LSP*. We further define ϵ_{SP} to be the error parameter for algorithm *LSP*, where $\epsilon_{SP} > 0$. For ease of analysis [8], we define ϵ'_{SP} to be the another error parameter for algorithm *LSP*, where $(2 + \frac{2W}{(1-2\epsilon'_{SP})\cdot w})\epsilon'_{SP} = \epsilon_{SP}$, and W and w are the maximum and minimum weight of all faces in F, respectively. We let L be the length of the longest edge of T, and N be the smallest integer value which is larger than or equal to the coordinate value of any vertex in V. Given a vertex v in V, we define h_v to be the minimum distance from v to the boundary of one of its incident faces, and define h to be the minimum v_v for all $v \in V$ (i.e., h actually is the minimum height of any face in F). Given a vertex v in V, we define a polygonal cap around v, denoted as C_v, to be a *sphere* with center at v. Let $r_v = \epsilon'_{SP} h_v$ be the radius of C_v with $0 < \epsilon'_{SP} < \frac{1}{2}$, and let r be the minimum r_v for all $v \in V$. Let θ_v to be the angle (measured in 3D) between any two edges of T that are incident to v, and let θ be the minimum θ_v for all $v \in V$. Figure 5 (a) shows an example of the polygonal cap C_v around v, with radius r_v, and the angle θ_v of v.

We then introduce the two steps in algorithm *LSP* as follows.

In the first step, we place Steiner points on each edge as follows. Let $\lambda = (1 + \epsilon'_{SP} \cdot \sin\theta_v)$ if $\theta_v < \frac{\pi}{2}$, and $\lambda = (1 + \epsilon'_{SP})$ otherwise. For each vertex v of face f_i, let e_a and e_b be the edges of f_i incident to v. We place Steiner points $a_1, a_2, \dots, a_{\tau_a - 1}$ (resp. $b_1, b_2, \dots b_{\tau_b - 1}$) along e_a (resp. e_b) such that $|\overline{va_j}| = r_v \lambda^{j-1}$ (resp. $|\overline{vb_k}| = r_v \lambda^{k-1}$) where $\tau_a = \log_\lambda \frac{|e_a|}{r_v}$ (resp. $\tau_b = \log_\lambda \frac{|e_b|}{r_v}$) for every integer $2 \le j \le \tau_a - 1$ (resp. $2 \le k \le \tau_b - 1$). We repeat it on each edge of f_i. Figure 5 (a) and Figure 5 (b) show an example of algorithm *LSP* with of Steiner points on e_a and e_b, and on three edges of f_i, respectively.

In the second step, we apply Dijkstra algorithm on a weighted graph constructed using the newly added Steiner points together with the original vertices to calculate the weighted shortest path $\Pi_{LSP}(s,t)$ between s and t that may not follow Snell's law, and retrieve the edge sequence that $\Pi_{LSP}(s,t)$ passes in order.

Theoretical analysis of algorithm *LSP*: We show the running time, memory usage, and bound the error of algorithm *LSP* in Theorem 4.1.

Theorem 4.1. *The running time for algorithm LSP is $O(n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}))$ and the memory usage is $O(n \log \frac{LN}{\epsilon_{SP}})$. This algorithm guarantees that $|\Pi_{LSP}(s,t)| \le (1 + \epsilon_{SP})|\Pi^*(s,t)|$.*

Proof Sketch. For the running time and memory usage, we show that the number of Steiner points k_{SP} on each edge is $O(\log \frac{LN}{\epsilon_{SP}})$. Following Dijkstra algorithm, we get the running time and memory usage for algorithm *LSP*. For the error bound, a proof sketch could be found in Theorem 1 of [8] and a detailed proof could be found in Theorem 3.1 of [23]. For the sake of space, all the detailed proof in this paper could be found in the appendix. □

4.1.2 Algorithm DLSP. Recall that in algorithm *LSP*, we create a polygonal cap C_v for each vertex v in V, and we will not place Steiner point inside C_v (i.e., we will not place Steiner point near the original vertices in V) in order to reduce the total number of Steiner points. So the Steiner points that $\Pi_{LSP}(s,t)$ pass will either be far away from the original vertices in V, or be the original vertices in V exactly. For the latter case, the retrieved edge sequence from $\Pi_{LSP}(s,t)$ will contain some vertices, and this edge sequence will be a non-full edge sequence.

So, in algorithm *DLSP*, given an edge sequence (which may be a non-full edge sequence) that $\Pi_{LSP}(s,t)$ passes in order, we aim to modify this edge sequence such that we could convert it to a full edge sequence. There are two steps in algorithm *DLSP*, (1) applying algorithm *LSP*, and (2) modifying $\Pi_{LSP}(s,t)$ to calculate the weighted shortest path (which may still not follow Snell's law) to convert its corresponding edge sequence from a non-full edge sequence to a full edge sequence.

We give some notations first. Given two points s and t in V, we define $\Pi_{DLSP}(s,t) = (s, \sigma_1, \dots, \sigma_l, t)$ to be the calculated weighted shortest path (which corresponding edge sequence is converted to from a non-full edge sequence to a full edge sequence, but the path still may not follow Snell's law) between s and t using algorithm *DLSP*. The path $(s, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, t)$ in Figure 6 (a), and path $(s, \phi_1, \phi_2, \phi_3, t)$ in Figure 6 (b) are two examples of $\Pi_{LSP}(s,t)$ that their corresponding edge sequences are non-full edge sequences since they passes vertex at (ϕ_2, ϕ_3, ϕ_4), and ϕ_2, respectively. The path $(s, \sigma_1, \sigma_2, \dots, \sigma_8, \sigma_9, t)$ in Figure 6 (a), and path $(s, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, t)$ (or $(s, \sigma_1, \sigma'_2, \sigma'_3, \sigma'_4, \sigma_5, t)$) in Figure 6 (b) are two examples of $\Pi_{DLSP}(s,t)$ since their corresponding edge sequences are converted to a full edge sequence.

We then introduce the two steps in algorithm *DLSP* as follows. In the first step, we run algorithm *LSP* and get $\Pi_{LSP}(s,t)$. In the second step, we modify $\Pi_{LSP}(s,t)$ to calculate the weighted shortest path (which may not follow Snell's law) to convert its corresponding edge sequence from a non-full edge sequence to a full edge sequence. Specifically, we check the path $\Pi_{LSP}(s,t)$ vertex by vertex from the source point s to the destination point t, and let the current checking point, the next checking point and the previous checking point in $\Pi_{LSP}(s,t)$ be v_c, v_n and v_p, respectively. Given a checking point v, v is on the edge means that the corresponding point in $\Pi_{LSP}(s,t)$ lies in the internal of an edge in E, v is on the original vertex in V means that the corresponding point in $\Pi_{LSP}(s,t)$ lies on the vertex in V. Depending on the type of v_c, there are two cases:

- If v_c is on the edge, we will not process it (e.g., $v_c = \phi_1$ in Figure 6 (a) or 6 (b)).
- If v_c is on the original vertex in V, there are two more cases:
  - Successive endpoint (refer to Figure 6 (a))
    - If v_n is on the vertex and v_p is on the edge (e.g., $v_c = \phi_2$, $v_n = \phi_3$ and $v_p = \phi_1$), it means that there are at least two successive points in $\Pi_{LSP}(s,t)$ that is on the vertex. This is called *successive endpoint*, and we store v_p as v_s (e.g., $v_s = \phi_1$), i.e., the start vertex of successive endpoint case.
    - If both v_n and v_p (e.g., $v_c = \phi_3$, $v_n = \phi_4$ and $v_p = \phi_2$) are on the vertex, it is a continuous step in successive endpoint case and we do nothing.
    - If v_n is on the edge and v_p is on the vertex (e.g., $v_c = \phi_4$, $v_n = \phi_5$ and $v_p = \phi_3$), is means we have finished finding the successive endpoints. We store v_n as v_e (e.g., $v_n = \phi_5$), i.e., the end vertex of successive endpoint case. Then, from v_s to v_e, we set ϵ_{SP} to be $\frac{\epsilon_{SP}}{2}$ (to double the number of Steiner points), and use the divide-and-conquer idea by calling algorithm *DLSP* itself, and denote the new path as $\Pi_{DLSP}(v_s,v_e) = (v_s = \phi_1 = \sigma_1, \sigma_2, \ldots, \sigma_8, \sigma_9 = \phi_5 = v_e)$ (i.e., the purple dashed line). We substitute $\Pi_{LSP}(v_s,v_e) = (v_s = \phi_1, \phi_2, \phi_3, \phi_4, \phi_5 = v_e)$ (i.e., the orange line) as $\Pi_{DLSP}(v_s,v_e)$ if $|\Pi_{DLSP}(v_s,v_e)| < |\Pi_{LSP}(v_s,v_e)|$.
  - Single endpoint (refer to Figure 6 (b))
    - If both v_n and v_p are on the edge, it means only v_c is on the vertex (e.g., $v_c = \phi_2$, $v_n = \phi_3$ and $v_p = \phi_1$). This is called *single endpoint*, and we 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. There are three possible ways to go v_p to v_n, which are (1) passes the original path $\Pi_{LSP}(v_p,v_n) = (v_p = \phi_1, \phi_2, \phi_3 = v_n)$ (i.e., the orange line), (2) passes the set of newly added Steiner points on the left side of the path (v_p, v_c, v_n), which is $\Pi_l(v_p,v_n) = (v_p = \phi_1 = \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5 = \phi_3 = v_n)$ (i.e., the purple dashed line), and (3) passes the set of newly added Steiner points on the right side of the path (v_p, v_c, v_n), which is $\Pi_r(v_p,v_n) = (v_p = \phi_1 = \sigma_1, \sigma_2', \sigma_3', \sigma_4', \sigma_5 = \phi_3 = v_n)$ (i.e., the pink dashed line). We compare the weighted distance among $\Pi_{LSP}(v_p,v_n)$, $\Pi_l(v_p,v_n)$, and $\Pi_l(v_p,v_n)$, and substitute $\Pi_{LSP}(v_p,v_n)$ as the path with the shortest weighted distance. We run this step for maximum ζ times (i.e., keep adding 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), if $\Pi_{LSP}(v_p,v_n)$ is still the longest path, where ζ is a constant and normally could be set as 10.

Then, we move forward in $\Pi_{LSP}(s,t)$ by setting v_c to be v_n, and updating v_n to be the next point of v_c and v_p to be the previous point of v_c in $\Pi_{LSP}(s,t)$ accordingly. After we process all the vertices in $\Pi_{LSP}(s,t)$, we return the result path as $\Pi_{DLSP}(s,t)$ and retrieve the edge sequence S that $\Pi_{DLSP}(s,t)$ passes in order.

Theoretical analysis of algorithm *DLSP*: We show the running time, memory usage and bound the error of algorithm *DLSP* in Theorem 4.2. Note that algorithm *DLSP* is a refinement step of algorithm *LSP*, so we still use the same error parameter of algorithm *LSP*, i.e., ϵ_{SP}, for algorithm *DLSP*.

Theorem 4.2. *The average case running time for algorithm DLSP is $O(n\log\frac{LN}{\epsilon_{SP}}\log(n\log\frac{LN}{\epsilon_{SP}})+\zeta n)$, the worst case running time is $O(n^2\log\frac{LN}{\epsilon_{SP}}\log(n\log\frac{LN}{\epsilon_{SP}}))$, and the memory usage is $O(n\log\frac{LN}{\epsilon_{SP}})$. This algorithm guarantees that $|\Pi_{DLSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)|$.*

Proof Sketch. For the average case running time, we have $\frac{1}{3}$ of $\Pi_{LSP}(s,t)$ passes on the edge, $\frac{1}{3}$ of $\Pi_{LSP}(s,t)$ belongs to single endpoint case, and the remaining $\frac{1}{3}$ of $\Pi_{LSP}(s,t)$ belongs to successive endpoint case. For the worst case running time, we have all the points in $\Pi_{LSP}(s,t)$ passes the original vertices in V. For the memory usage, we consider the memory for Dijkstra algorithm, handling one single endpoint case and handling successive endpoint case. For the error bound, we only use the refinement path $\Pi_{DLSP}(s,t)$ if its weighted distance is shorter than $\Pi_{LSP}(s,t)$. □

4.2 Edge Sequence Based Weighted Shortest Path Finding

4.2.1 Algorithm BSSL. In algorithm *BSSL*, given the edge sequence S that $\Pi_{DLSP}(s,t)$ passes, we aim to find the weighted shortest path that follows the Snell's law on S. There is only one step in algorithm *BSSL*.

We give some notations first. Given two points s and t in V, we define $\Pi_{SL}(s,t|S) = (s, \rho_1, \ldots, \rho_l, t)$ to be the calculated weighted shortest path (which follows Snell's law) between s and t using algorithm *BSSL* on the edge sequence S. Let ϵ_{SL} to be the error parameter for algorithm *BSSL*, where $\epsilon_{SL} > 0$. In Figure 3, $\Pi_{DLSP}(s,t)$ passes the edge sequence $S = (e_1, e_2)$ (i.e., the edges highlighted in red) with the corresponding face sequence $F(S) = (f_0, f_1, f_2)$. The green line in this figure shows an example of $\Pi_{SL}(s,t|S)$. In this example, the edge sequence that $\Pi^*(s,t)$ passes (i.e., S^*) is the same as the edge sequence S, and this is also the most common case, so $\Pi^*(s,t|S) = \Pi^*(s,t)$ (i.e., the blue line). But, it could happen that they are different. We will show how to bound the error if S is different from S^* in Section 4.3. Furthermore, given two sequence of points X and X', we define $X \oplus X'$ to be a new sequence of points X by appending X' to the end of X. For example, in Figure 2, we have $\Pi^*(s,t) = \Pi^*(s,r_i) \oplus \Pi^*(r_i,t)$.

Following the Snell's law, when given an edge sequence S, and a point c_1 on $e_1 \in S$, we can apply the Snell's law from the source point s that passes e_1 at c_1, and let it be the *out-ray* R_1^c of f_1. Suppose that R_1^c intersects e_2 at a point c_2, then we can continue the calculation to obtain the path $\Pi_c = (s, c_1, c_2, \ldots, c_g, R_g^c)$, where $1 \leq g \leq l$ and R_g^c is the last out-ray of the path at $e_g \in S$. This calculation will stop when (1) R_g^c intersects $e_g \in S$ with $g = l$, or (2) R_g^c intersects $e_g \in S$ but does not intersect $e_{g+1} \in S$ with $g < l$. We call Π_c as a *3D surface Snell ray*, which simulates a light ray from s and passing S by starting at c_1 on e_1, and ending at c_g on e_g. Figure 7 shows an example of $\Pi_m = (s, m_1, m_2, R_2^m)$ (i.e., the blue line) that does not pass the whole $S = (e_1, e_2, e_3)$ because R_2^m intersects e_2 but does not intersect e_3, and $\Pi_{m'} = (s, m_1', m_2', m_3', R_3^{m'})$ (i.e., the purple line) that passes the whole S because $R_3^{m'}$ intersects e_3. Therefore, if we could find the point ρ_1 on e_1 such that after we calculate the 3D surface Snell ray $\Pi_\rho = (s, \rho_1, \ldots, \rho_l, R_l^\rho)$, t is on the last ray R_l^ρ, then Π_ρ will be the result of $\Pi_{SL}(s,t|S)$.

We then introduce algorithm *BSSL* as follows (refer to Figure 7). For $i \in \{1, \ldots, l\}$, we let $[a_i, b_i]$ be a candidate interval on $e_i \in S$, let m_i be the midpoint of $[a_i, b_i]$, and initial set $a_i = p_i$ and $b_i = q_i$, where p_i and q_i are the left and right endpoint of e_i, respectively. Then, we can find the 3D surface Snell ray $\Pi_m = (s, m_1, \ldots, m_g, R_g^m)$ from s and passing S by starting at m_1 on e_1, and to c_g on e_g, with $g \leq l$ (e.g., we have the blue line $\Pi_m = (s, m_1, m_2, R_2^m)$ when $i = 1$). Depending on whether Π_m leave the edge sequence S or not, there are two cases as follows:

- Π_m does not pass the whole S, i.e., R_g^c intersects $e_g \in S$ but does not intersect $e_{g+1} \in S$ with $g < l$ (e.g., the blue line $\Pi_m = (s, m_1, m_2, R_2^m)$ when $i = 1$ does not pass the whole S): If e_{g+1} is on the left side of R_g^m, then we need to search in $[a_i, m_i]$, so we set $b_i = m_i$. If e_{g+1} is on the right side of R_g^m, then we need to search in $[m_i, b_i]$, so we set $a_i = m_i$. (E.g., e_3 is on the left side of the blue line R_2^m, we set $b_1 = m_1$, and we have $[a_1, b_1] = [p_1, m_1]$ when $i = 1$.)
- Π_m passes the whole S, i.e., R_g^c intersects $e_g \in S$ with $g = l$ (e.g., the purple line $\Pi_{m'} = (s, m_1', m_2', m_3', R_3^{m'})$ when $i = 1$ passes the whole S): If t is on R_g^m, then we could just return Π_m as the result. If t is on the left side of R_g^m, then we need to search in $[a_i, m_i]$, so we set $b_i = m_i$. If t is on the right side of R_g^m, then we need to search in $[m_i, b_i]$, so we set $a_i = m_i$. (E.g., t is on the right side of the purple line R_2^m, we set $a_1 = m_1'$, and we have $[a_1, b_1] = [m_1', m_1]$ when $i = 1$.)

We then perform this step until $|a_i b_i| < \delta$ (e.g., $|a_1 b_1| = |m_1' m_1| < \delta$ when $i = 1$). Note that $\delta = \frac{h\epsilon_{SL} w}{6lW}$ is an error parameter depending on ϵ_{SL}, where h is the minimum height of any face in F, W and w are the maximum and minimum weights of face in F, and l is the number of edges in S, respectively. We calculate the midpoint of $[a_i, b_i]$ as ρ_i (e.g., ρ_1 is the midpoint of $[a_1, b_1] = [m_1', m_1]$ when $i = 1$), and store ρ_i in $\Pi_{SL}(s,t|S)$ using $\Pi_{SL}(s,t|S) \oplus (\rho_i)$ where $\Pi_{SL}(s,t|S)$ is initialized to be (s) (e.g., we have the pink dashed line $\Pi_{SL}(s,t|S) = (s, \rho_1)$ when $i = 1$). Then, we move forward (i.e., $i = i + 1$) and let ρ_i be the starting point of new Π_m that passing S by starting at m_{i+1} on e_{i+1}, and to m_g on e_g (e.g., we have the green line $\Pi_{m''} = (\rho_1, m_2'', R_2^{m''})$ and the yellow line $\Pi_{m'''} = (\rho_1, m_2''', m_3''', R_3^{m'''})$ when $i = 2$). We iterate this step until we process all the edges in S (e.g., until we process all the edges in $S = (e_1, e_2, e_3)$, we get result path $\Pi_{SL}(s,t|S) = (s, \rho_1, \rho_2, \rho_3, t)$).

Theoretical analysis of algorithm *BSSL*: We show the running time, memory usage and bound the error of algorithm *BSSL* in Theorem 4.3.

Theorem 4.3. *The running time for algorithm BSSL is $O(n^4 \log \frac{nWL}{h\epsilon_{SL} w})$, and the memory usage is $O(n^2)$. This algorithm guarantees that $|\Pi_{SL}(s,t|S)| \leq (1 + \epsilon_{SL})|\Pi^*(s,t|S)|$.*

Proof Sketch. For the running time and memory usage, in the binary search step on each edge, we proof that the running time is $O(n^2 \log \frac{nWL}{h\epsilon_{SL} w})$. Since there are at most n^2 edges in S, we get the running time and memory usage for algorithm *BSSL*. For the error bound, for $i \in \{0, 1, 2, \ldots, l\}$, we use induction to prove $|\Pi_{SL}(\rho_i, t|S)| \leq (1 + \frac{\epsilon}{2})|\Pi^*(\rho_i, t|S)| + 3(l - i)\delta W$, then by setting $k = 0$ and since we set $\delta = \frac{h\epsilon_{SL} w}{6lW}$, we finish the proof. □

4.2.2 *Algorithm EWSL.*

Even though algorithm *BSSL* is the fastest algorithm for finding the weighted shortest path that follows Snell's law with the given S, we could still use effective weight pruning technique to make it run even faster.

So, in algorithm *EWSL*, given the edge sequence S that $\Pi_{DLSP}(s,t)$ passes, we still aim to find the weighted shortest path that follows the Snell's law on S, but with a shorter time. There are two steps in algorithm *EWSL*, (1) applying algorithm *BSSL*, and (2) applying effective weight pruning technique. Since algorithm *EWSL* will not affect the result of the calculated weighted shortest path (which follows Snell's law), we still let $\Pi_{SL}(s,t|S)$ be the calculated weighted shortest path (which follows Snell's law) between s and t using algorithm *EWSL*, which is the same as algorithm *BSSL*.

We introduce the two steps in algorithm *EWSL* as follows (we use Figure 8 for illustration).

In the first step, we use algorithm *BSSL* to find a 3D surface Snell ray $\Pi_m = (s, m_1, \ldots, m_l, R_l^m)$ (e.g., the blue line $\Pi_m = (s, m_1, m_2, R_2^m)$) from s with the initial ray through m_1 that Π_m passes the whole S for the first time. Note that if Π_m does not pass the whole S, we repeat the iteration in algorithm *BSSL* until it does.

In the second step, we apply effective weight pruning technique to reduce the number of iterations.

- Firstly (refer to Figure 8 (a)), given two edges that adjacent to t in the last face f_l in $F(S)$, we calculate the intersection point between R_l^m and these two edges (either one of these two edges), and denote it as u (e.g., the blue line R_l^m intersect with the left edge of f_l that adjacent to t, i.e., $\overline{p_1 t}$).
- Secondly (refer to Figure 8 (a)), we project u onto f_0 (i.e., the first face in $F(S)$) into two-dimensional (2D), and denote the projection point as u_p. Now, the whole $F(S)$ could be divided into two parts using e_1, which are (1) f_0, and (2) all the faces in $F(S)$ except f_0. For the latter one, we regard them as one *effective face* and denote it as f_{ef}, where the weight of f_{ef} is called *effective weight* and we denote it as w_{ef} (e.g., $f_{ef} = \triangle u_p p_1 q_1$ is an effective face for f_1 and f_2).
- Thirdly (refer to Figure 8 (a)), by using $\overline{sm_1}$ (e.g., the blue line), $\overline{m_1 u_p}$ (e.g., the orange dashed line), and the weight for f_0 (i.e., w_0), we could use Snell's law to calculate w_{ef}, i.e., the effective weight for f_{ef}. Note that $\overline{sm_1}$ and $\overline{m_1 u_p}$ are on the same plane, since they are on f_0 and f_{ef}, where f_0 and f_{ef} are coplanar.
- Fourthly (refer to Figure 8 (b)), we project t onto f_0 (or f_{ef}, since they are coplanar) into 2D, and denote the projection point as t_p. We apply the Snell's law again to find the effective passing point m_{ef} on e_1 using the w_0, w_{ef}, s and t_p in a quartic equation (note that only two faces f_0 and f_{ef} are involved, so the equation will have the unknown at power of four). Specifically, we set m_{ef} to be unknown and use the Snell's law in vector form [7], we could build a quartic equation using w_0, w_{ef}, $\overline{sm_{ef}}$ (e.g., the purple line $\overline{sm_{ef}}$) and $\overline{m_{ef} t_p}$ (e.g., the green dashed line $\overline{m_{ef} t_p}$). Then, we could use root formula [6] to solve m_{ef}.
- Fifthly (refer to Figure 8 (c)), we compute the 3D surface Snell ray Π_m from s with the initial ray through m_{ef} (e.g., the purple

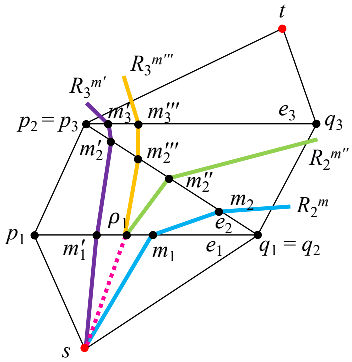

Figure 7: An example of algorithm ***BSSL***

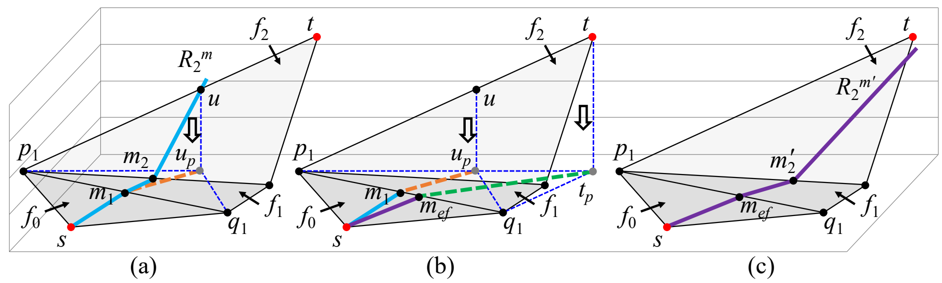

Figure 8: An example of algorithm *EWSL* (a) with initial ray for calculating effective weight on the effective face $\triangle u_p p_1 q_1$, (b) for calculating m_{ef} using the weight of f_0 and the effective weight of $\triangle u_p p_1 q_1$ (or $\triangle t_p p_1 q_1$), and (c) with final ray passing through m_{ef}

line $\Pi_{m'} = (s, m_{ef}, m'_2, R_2^{m'})$), then follow the remaining steps in algorithm *BSSL*.

An illustration of the reasons in 1D example on why algorithm *EWSL* could prune out unnecessary checking in algorithm *BSSL* could be found in the appendix.

Theoretical analysis of algorithm *EWSL*: Note that algorithm *EWSL* is a pruning step based on algorithm *BSSL*, the theoretical running time, the theoretical memory usage, the error ratio and the actual calculated weighted shortest path (which follows Snell's law) of algorithm *EWSL* is the same as algorithm *BSSL*. So we still use the same error parameter of algorithm *BSSL*, i.e., ϵ_{SL}, for algorithm *EWSL*. Algorithm *EWSL* will only improve the experimental running time and memory usage compared with algorithm *BSSL*.

4.3 Summary

We provide a summary of the relationship among $\Pi(s,t)$, $\Pi_{DLSP}(s,t)$, and $\Pi_{SL}(s,t|S)$, and also ϵ, ϵ_{SP} and ϵ_{SL}, respectively. Recall that the calculated weighted shortest path (which may not follow Snell's law) using algorithm *DLSP* is $\Pi_{DLSP}(s,t)$, the calculated weighted shortest path (which follows Snell's law) using algorithm *EWSL* is $\Pi_{SL}(s,t|S)$, the final calculated weighted shortest path (which follows Snell's law) of algorithm *DLSP-EWSL* (i.e., our two-step algorithm) is $|\Pi(s,t)|$, and the optimal weighted shortest path is $\Pi^*(s,t)$. Usually, $\Pi_{SL}(s,t|S)$ is a refinement of $\Pi_{DLSP}(s,t)$ that follows Snell's law, so $|\Pi_{DLSP}(s,t)| \geq |\Pi_{SL}(s,t|S)|$. But, it could happen that edge sequence S found using algorithm *DLSP* may not be the optimal edge sequence S^* that $\Pi^*(s,t)$ pass, which will make $|\Pi_{DLSP}(s,t)| < |\Pi_{SL}(s,t|S)|$. In order to guarantee a general error bound, we set $|\Pi(s,t)| = \min(|\Pi_{DLSP}(s,t)|, |\Pi_{SL}(s,t|S)|)$. In addition, the error parameter for algorithm *DLSP* is ϵ_{SP}, the error parameter for algorithm *EWSL* is ϵ_{SL}, and the error parameter of algorithm *DLSP-EWSL* is ϵ. Since only ϵ is the user-defined error parameter, ϵ_{SP} and ϵ_{SL} are depended on ϵ, we let $\epsilon = \epsilon_{SP} = \epsilon_{SL}$. But, it is sufficient to bound the error of algorithm *DLSP-EWSL* by simply having $\epsilon = \max[\epsilon_{SP}, \epsilon_{SL}]$.

Theoretical analysis of algorithm *DLSP-EWSL*: We combine algorithm *DLSP* and algorithm *EWSL* to get the total running time and memory usage, and bound the error for algorithm *DLSP-EWSL* in Theorem 4.4.

THEOREM 4.4. *The average case total running time for algorithm DLSP-EWSL is* $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + \zeta n + n^4 \log \frac{nWL}{h\epsilon w})$, *the worst case total running time is* $O(n^2 \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + n^4 \log \frac{nWL}{h\epsilon w})$, *the total memory usage is* $O(n \log \frac{LN}{\epsilon} + n^2)$. *This algorithm guarantees that* $|\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$.

PROOF SKETCH. For the running time and memory usage, we could use Theorem 4.2 and Theorem 4.3 to proof. For the error bound, depending on whether the edge sequence S found by $\Pi_{DLSP}(s,t)$ is the same as the optimal edge sequence S^* that $\Pi^*(s,t)$ passes, and whether the path $\Pi_{DLSP}(s,t)$ found by algorithm *DLSP* is longer or the path $\Pi_{SL}(s,t|S)$ found by SL is longer, there are four cases. Then we could use Theorem 4.2 and Theorem 4.3 to proof. □

5 BASELINE & COMPARISON

As mentioned in Section 1.2, the existing algorithms for solving the weighted region problem approximately could be divided into four categories: (1) *WPSL* approach, (2) *SP* approach, and (3) *SPSL* approach.

The *SP* approach proposed by [24] (i.e., algorithm *FSP*) is a widely-known algorithm and a standard approach for solving the weighted region problem [29], we set it as one of our baseline algorithms. The *SP* approach proposed by [8] (i.e., algorithm *LSP*) is regarded as the best-known algorithm in the *SP* approach type due to its shortest running time, we also set it as one of our baseline algorithms. But, note that the calculated weighted shortest path for algorithm *FSP* and algorithm *LSP* does not follow Snell's law, which violets the second criterion of a good algorithm for solving the weighted region problem. In addition, the *SPSL* approach proposed by [28] (i.e., algorithm *FSP-BSSL*) is the fastest algorithm among all four types of algorithms for solving the weighted region problem (based on the requirement that the weighted shortest path need follow Snell's law) and it is regarded as the best-known algorithm, we also set it as one of our baseline algorithms. Thus, we compared three baseline algorithms, i.e., algorithm *FSP*, algorithm *LSP* and algorithm *FSP-BSSL*.

We do not use the *WPSL* approach as the baseline algorithm. This is because its running time is very large, which violets the first criterion of a good algorithm for solving the weighted region problem. In addition, the *SPSL* approach actually is a modification of the *WPSL* approach, which could let the weighted shortest path follows Snell's law and also reduce the running time. Furthermore, to the best of our knowledge, there is no implementation of the *SPSL* approach so far, even the work [26] that proposes this approach does not provide an implementation of their algorithm [23].

We then compare the baseline algorithms and our algorithm, i.e., algorithm *DLSP-EWSL* in two separate steps. Table 1 (resp. Table 2) shows the comparison of *FSP*, *LSP* and *DLSP* (resp. *BSSL* and *EWSL*) in terms of the running time and memory usage, respectively. *DLSP* and *LSP* could perform much better than *FSP* in terms of theoretical running time (both average case and worst case) and memory usage. This is because the dependence on n for *FSP* (resp. *DLSP* and *LSP*) is very large (resp. small), where n is usually quite large (larger than 10^5) for real dataset. Furthermore, even though *DLSP* and *LSP* also depend on $\log \frac{LN}{\epsilon}$, this term actually is a constant, which could be ignored in the analysis of big-O. In our experiment on the real dataset, the maximum value for $\log \frac{LN}{\epsilon} \approx 25$ with the real values $L_{max} = 156$, $N_{max} = 10,000$ and $\epsilon_{min} = 0.05$, but the maximum value for n is about $150,000$. In addition, ζ is also a constant and is usually set to be 10. Regarding *EWSL* and *BSSL*, even though both of them have the same theoretical running time and memory usage, *EWSL* uses pruning technique compared with *BSSL*, so the experimental running time and memory usage of *EWSL* will outperform *BSSL* (which will be discussed in Section 6).

6 EMPIRICAL STUDIES

6.1 Experimental Setup

We conducted our experiments on a Linux machine with 2.67 GHz CPU and 48GB memory. All algorithms were implemented in C++. For the following experiment setup, we mainly follow the experiment setup in the work [20, 21, 25, 30].

Datasets: Following some existing studies on terrain data [15, 25, 27], we conducted our experiment based on two real terrain datasets, namely BearHead (in short, *BH*, with 280k faces), and EaglePeak (in short, *EP*, with 300k faces) [3, 30]. Following the work [17, 28], we set the weight of a triangle in terrain datasets to be the slope of that face. Besides, a small-version of *BH* and *EP* datasets (in short, *BH-small* and *EP-small* dataset) which corresponds to a small sub-region of the *BH* and *EP* datasets containing 3k faces (for both *BH-small* and *EP-small* datasets) were also used since one of the baseline algorithms, i.e., algorithm *FSP* [24], is not feasible on any of the full datasets due to its expensive running time.

In addition, we also have the two set of datasets with different dataset sizes (one set of large-version datasets and one set of small-version datasets) for testing the scalability of our algorithm (since algorithm *FSP* is not feasible on any of the full datasets, so we have this small-version datasets). We generate these two set of datasets using *EP* and *EP-small* following the procedure in the work [25, 30] (which creates a set of terrains with different resolutions). This procedure could be found in the appendix.

Algorithms: Our algorithm *DLSP-EWSL*, and the baseline algorithms, i.e., *FSP* [24], *LSP* [8] and *FSP-BSSL* [28], are studied in the experiments. But, note that the calculated weighted shortest path of *FSP* and *LSP* do not follow Snell's law. In order to conduct the ablation study, i.e., show the superior performance of our algorithms in both edge sequence finding step and edge sequence based weighted shortest path finding step, we also interchanged two steps of *DLSP-EWSL* and *FSP-BSSL*. That is, we also studied *FSP-EWSL* and *DLSP-BSSL* in the experiments. In total, we compared six algorithms, namely, *FSP*, *LSP*, *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL*. Since *FSP* is not feasible on large datasets due to its expensive running time, so we (1) compared these six algorithms on *BH-small* and *EP-small* datasets, and the set of small-version datasets, and (2) compared *LSP*, *DLSP-BSSL* and *DLSP-EWSL* on *BH* and *EP* datasets, and the set of large-version datasets.

Query Generation: We randomly generate two vertices on the surface of a terrain, one as a source and the other as a destination. For each measurement, 100 queries were answered and the average result was returned.

Factors & Measurements: We studied four factors in the experiments, namely (1) ϵ, (2) ϵ_{SP}, (3) ϵ_{SL}, and (4) dataset size *DS* (i.e., the number of faces in a terrain model). Note that only ϵ is the user-defined parameter with $\epsilon = \epsilon_{SP} = \epsilon_{SL}$. But, in order to see how varying ϵ_{SP} (resp. ϵ_{SL}) could affect the performance of the whole algorithm, we also changed the value of ϵ_{SP} (resp. ϵ_{SL}) and kept the other one, i.e., ϵ_{SL} (resp. ϵ_{SP}) to be unchanged. In this case, we have $\epsilon = \max[\epsilon_{SP}, \epsilon_{SL}]$, as mentioned in Section 4.3.

In addition, we used five measurements to evaluate the performance of our algorithms, namely (1) *preprocessing time* (i.e., the time for constructing the weighted graph using Steiner points), (2a) *query time for the first step* (i.e., the time in edge sequence finding step), (2b) *query time for the second step* (i.e., the time in edge sequence based weighted shortest path finding step), (2c) *improvement ratio of query time for the second step* (i.e., the improvement ratio in percentage of query time for the second step from algorithm *BSSL* to algorithm *EWSL*, since the improvement from algorithm *BSSL* to algorithm *EWSL* is no obvious due to the large scale in the experiment figure), (2d) *total query time* (i.e., the time for finding the weighted shortest path that follows Snell's law), (3a) *memory usage for the first step* (i.e., the space consumption in edge sequence finding step), (3b) *memory usage for the second step* (i.e., the space consumption in edge sequence based weighted shortest path finding step), (3c) *total memory usage* (i.e., the space consumption for finding the weighted shortest path that follows Snell's law), (4) *Snell's law iteration count* (i.e., the total number of iterations in edge sequence based weighted shortest path finding step), and (5) *distance error* (i.e., the error of the distance returned based on the algorithm compared with the exact weighted shortest path).

Note that there is no known algorithm for solving the weighted region problem exactly currently, so we use algorithm *FSP-BSSL* and set $\epsilon = 0.05$ (by setting $\epsilon_{SP} = \epsilon_{SL} = 0.05$) to simulate the exact weighted shortest path on a small-version of datasets for measuring distance error. Since algorithm *FSP* is not feasible on any of the full datasets due to its expensive running time, we cannot simulate the exact weighted shortest path on the full datasets, so the distance error is omitted for the full datasets.

6.2 Experimental Results

Figure 9, Figure 10, and Figure 11 show the result on the *EP-small* dataset when varying ϵ, ϵ_{SP}, and ϵ_{SL}, respectively. Figure 12 (resp.

Algorithm	Average Time	Worst Time	Memory
FSP [28]	$O(n^3 \log n)$	$O(n^4 \log n)$	$O(n^3)$
LSP [8]	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	$O(n^2 \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	$O(n \log \frac{LN}{\epsilon})$
DLSP	$O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + \zeta n)$	$O(n^2 \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	$O(n \log \frac{LN}{\epsilon})$

Table 1: Comparison of algorithm *FSP*, algorithm *LSP* and algorithm *DLSP*

Algorithm	Time	Memory
BSSL [28]	$O(n^4 \log(\frac{nNW}{w\epsilon}))$	$O(n^2)$
EWSL	$O(n^4 \log(\frac{nNW}{w\epsilon}))$	$O(n^2)$

Table 2: Comparison of algorithm *BSSL* and algorithm *EWSL*

Remark: Even though algorithm *LSP* and algorithm *DLSP* have similar running time and memory usage, the latter one allows us to exploit Snell's law on the result of algorithm *DLSP*, such that the final path result of algorithm *DLSP-EWSL* could follow Snell's law. Even though algorithm *BSSL* and algorithm *EWSL* have the same theoretical running time and memory usage, the latter one's experimental running time and memory usage outperform the former one's.

Figure 13) shows the result on the *EP* dataset (resp. a set of large-version datasets) when varying ϵ (resp. *DS*). All the five measurements are included in Figure 9, and only the preprocessing time, query time and memory usage are included for the rest of Figures (for sake of space). The results on other combinations of dataset and the variation of ϵ, ϵ_{SP} and ϵ_{SL} could be found in the appendix. Note that the magnitude difference between the query time (resp. memory usage) on the first step and the second step is very large, so it seems that only one step (out of the two steps) is shown in the the experiment figures of total query time (resp. total memory usage). The separation of two steps in query time (resp. memory usage) could also be found in the appendix.

Effect of ϵ. In Figure 9 and Figure 12, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on *EP-small* and *EP* datasets by setting $\epsilon_{SP} = \epsilon_{SL} = \epsilon$. Algorithm *DLSP-EWSL* superior performance of all the remaining algorithms (except algorithm *LSP*) in terms of preprocessing time, query time, memory usage and iteration count. Even though algorithm *LSP* seems to run faster than algorithm *DLSP-EWSL*, the former one doesn't follow Snell's law, which violets the second criterion of a good algorithm for solving the weighted region problem. The errors of all the six algorithms are very small (close to 0%) and much smaller than the theoretical bound.

Effect of ϵ_{SP}. In Figure 10, we tested 6 values of ϵ_{SP} from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on *EP-small* dataset by setting ϵ_{SL} to be 0.1 as default value for all the cases. The preprocessing time, query time and memory usage of algorithm *DLSP* (i.e., *DLSP-BSSL* and *DLSP-EWSL*) and *LSP* are 2-3 orders of magnitude smaller than *FSP* (i.e., *FSP*, *FSP-BSSL* and *FSP-EWSL*).

Effect of ϵ_{SL}. In Figure 11, we tested 6 values of ϵ_{SL} from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on *EP-small* dataset by setting ϵ_{SP} to be 0.1 as default value for all the cases. The preprocessing time, query time for the first step and memory usage for the first step will not be affected by ϵ_{SL}. The query time, memory usage and iteration count of algorithm *EWSL* (both *FSP-EWSL* and *DLSP-EWSL*) could always perform 4% to 20% better than *BSSL* (both *FSP-BSSL* and *DLSP-BSSL*), and they will decrease when ϵ_{SL} is increasing.

Effect of *DS* **(scalability test)**. In Figure 13, we tested 5 values of *DS* from {200k, 400k, 600k, 800k, 1000k} on the set of large-version datasets (by setting ϵ to be 0.25) for scalability test. When the dataset size is 1000k, algorithm *DLSP-EWSL* could beat other two algorithms (given that the path needs to follow Snell's law). We also tested 5 values of *DS* from {10k, 20k, 30k, 40k, 50k} on the set of small-version datasets (by setting ϵ to be 0.1). The figure could be found in the appendix. When the dataset size is 50k, the state-of-the-art algorithm's (i.e., algorithm *FSP-BSSL*) total query time is 119,000s ($\approx$ 1.5 day) and total memory usage is 2.9GB, while our algorithm's (i.e., algorithm *DLSP-EWSL*) total query time is 534s ($\approx$ 9 min) and total memory usage is 130MB, which shows the excellent performance of our algorithm.

6.3 Case Study

6.3.1 User Study. We conducted a user study on a web-based campus map weighted shortest path finding tool, which allows users to find and visualize the weighted shortest path between any two rooms (or buildings) in university campus, namely *Path Advisor* [33]. They hope (1) the path should not be too close to the obstacle (e.g., the distance between the path to the obstacle should be at least 0.2 meter), and (2) the path should not have sudden direction changes. Based on this, when a face is closer to the boundary of aisle in a building (resp. the aisle center), they set the face with a larger (resp. smaller) weight. We obtained the code from the authors of [33] and adopted our six algorithms to their tool. We selected two places in Path Advisor, namely atrium and lift 25/26 (with distance 105.8m), as source and destination, and repeated it for 100 times to calculate the weighted shortest path. Figure 14 shows an example of different paths in Path Advisor. The blue, pink and yellow path are the weighted shortest path that follows Snell's law (calculated using algorothm *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL*), the weighted shortest path that does not follow Snell's law (calculated using *FSP* and *LSP*), and the unweighted shortest path. The blue path is shorter than the pink path since it follows the Snell's law. The yellow path is not realistic since it is very close to the obstacle and it has sudden direction changes. The average query time (with $\epsilon = 0.05$) for *FSP*, *LSP*, *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL* are 90.3s, 3s, 92.1s, 90.7s, 3.4s and 3.1s, respectively. We then collected 30 users' response on whether they think query time of these six algorithms on Path Advisor are reasonable. 96.7% of users cannot accept the query time larger than 60s, and 100% of users think 3.1s is the most acceptable query time given that the path need to follow Snell's law.

6.3.2 Motivation study. We also conducted a motivation study on the placement of undersea optical fibre cable on the seabed, which is one of the motivations for the weighted region problem in Section 1.1. We first obtained the seabed terrain height map from [9], then used Blender [1] to generate the seabed 3D terrain model. For a face with deeper sea, the hydraulic pressure is higher, and it is more expensive to build the cable, so the face will have a larger weight. We randomly selected two points as source and destination, and repeated it for 100 times to calculate the weighted shortest path (with distance 954.9km). Figure 15 shows an example of different paths on seabed. The green, blue and red path are the weighted shortest path that follows Snell's law, the weighted shortest path that does not follow Snell's law, and the unweighted shortest path. The red path is not realistic since it passes sea valley where the

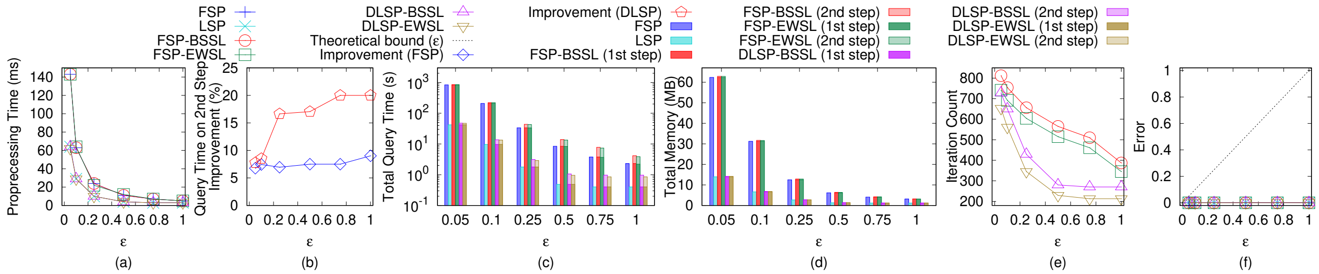

Figure 9: Effect of ϵ on *EP-small* dataset

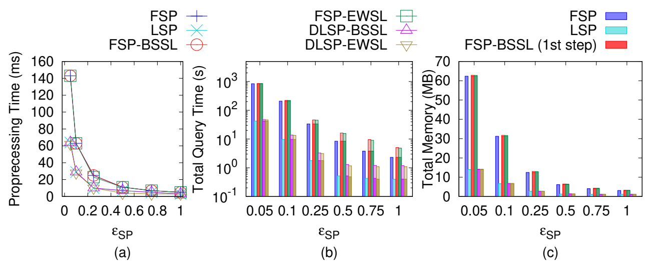

Figure 10: Effect of ϵ_{SP} on *EP-small* dataset

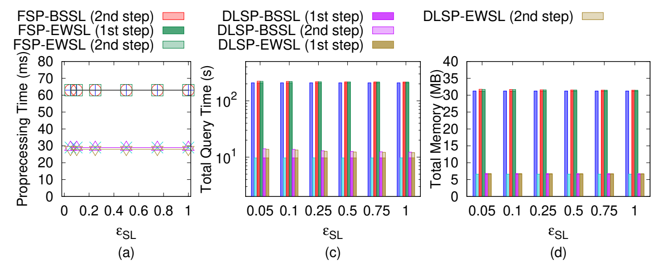

Figure 11: Effect of ϵ_{SL} on *EP-small* dataset

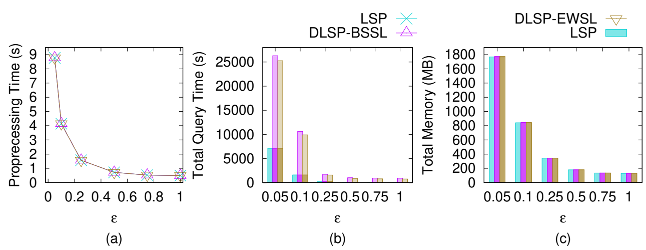

Figure 12: Effect of ϵ on *EP* dataset

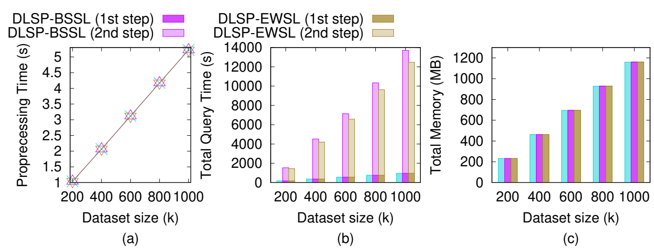

Figure 13: Effect of dataset size on a set of large-version datasets

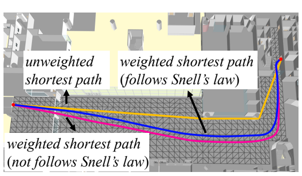

Figure 14: An example of paths in Path Advisor

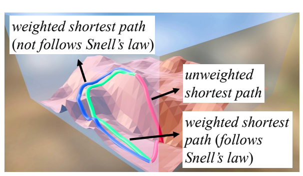

Figure 15: An example of paths on seabed

hydraulic pressure is higher, so the cost is more expensive. The average query time (with $\epsilon = 0.05$) for algorithm *FSP*, *LSP*, *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL* are 2787s, 41s, 2883s, 2880s, 64s and 62s, respectively. It still shows that *DLSP-EWSL* is the best given that the path need to follow Snell's law.

6.4 Experimental Results Summary

Our algorithm *DLSP-EWSL* consistently outperforms the state-of-the-art algorithm, i.e., algorithm *FSP-BSSL*, in terms of all measurements (i.e., preprocessing time, query time, memory usage and iteration count). Specifically, the first step and the second step of our algorithm runs up to 500 times and 20% faster than the state-of-the-art algorithm. When the dataset size is 50k, our algorithm's total query time is 534s ($\approx$ 9 min) and total memory usage is 130MB, but the state-of-the-art algorithm's total query time is 119,000s ($\approx$ 1.5 day) and total memory usage is 2.9GB. The case study also shows that algorithm *DLSP-EWSL* is the best algorithm given that the path need to follow Snell's law.

7 CONCLUSION

In our paper, we propose a two-step approximation algorithm for calculating the weighted shortest path that follows Snell's law in 3D weighted region problem using algorithm *DLSP-EWSL*. Our algorithm could bound the error ratio, and the experimental results show that algorithm *DLSP* and algorithm *EWSL* runs up to 500 times and 20% faster than the state-of-the-art algorithm, respectively. The future work could be that proposing a new pruning technique based on effective weight to reduce the algorithm running time further.

REFERENCES

[1] 2022. *Blender*. https://www.blender.org
[2] 2022. *Cyberpunk 2077*. https://www.cyberpunk.net
[3] 2022. *Data Geocomm*. http://data.geocomm.com/
[4] 2022. *Google Earth*. https://earth.google.com/web
[5] 2022. *Metaverse*. https://about.facebook.com/meta
[6] 2022. *Quartic function*. https://en.wikipedia.org/wiki/Quartic_function
[7] 2022. *Snell's law in vector form*. https://physics.stackexchange.com/questions/435512/snells-law-in-vector-form
[8] 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.
[9] Christopher Amante and Barry W Eakins. 2009. ETOPO1 arc-minute global relief model: procedures, data sources and analysis. (2009).
[10] Harri Antikainen. 2013. Comparison of Different Strategies for Determining Raster-Based Least-Cost Paths with a Minimum Amount of Distortion. *Transactions in GIS* 17, 1 (2013), 96–108.
[11] Prosenjit Bose, Anil Maheshwari, Chang Shu, and Stefanie Wuhrer. 2011. A survey of geodesic paths on 3D surfaces. *Computational Geometry* 44, 9 (2011), 486–498.
[12] Jindong Chen and Yijie Han. 1990. Shortest Paths on a Polyhedron. In *SOCG*. New York, NY, USA, 360–369.
[13] 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.
[14] 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.
[15] Ke Deng and Xiaofang Zhou. 2004. Expansion-based algorithms for finding single pair shortest path on surface. In *International Workshop on Web and Wireless Geographical Information Systems*. Springer, 151–166.
[16] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.
[17] Amin Gheibi, Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. 2018. Path refinement in weighted regions. *Algorithmica* 80, 12 (2018), 3766–3802.
[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] Norman Jaklin, Mark Tibboel, and Roland Geraerts. 2014. Computing high-quality paths in weighted regions. In *Proceedings of the Seventh International Conference on Motion in Games*. 77–86.
[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] Mark R Kindl and Neil C Rowe. 2012. Evaluating simulated annealing for the weighted-region path-planning problem. In *2012 26th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 926–931.
[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] 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.
[26] 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.
[27] 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.
[28] 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.
[29] Nguyet Tran, Michael J Dinneen, and Simone Linz. 2020. Computing Close to Optimal Weighted Shortest Paths in Practice. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 30. 291–299.
[30] 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.
[31] 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.
[32] Da Yan, Zhou Zhao, and Wilfred Ng. 2012. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. 942–951.
[33] 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.
[34] 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 REMARK ON ALGORITHM *DLSP*

Recall that in algorithm *DLSP*, only if the weighted distance of the path segment after applying the algorithm *DLSP* is shorter than the weighted distance of the original path segment, we will substitute the original path segment with the new path segment. In Figure 6 (a), we substitute $\Pi_{LSP}(v_s, v_e) = (v_s = \phi_1, \phi_2, \phi_3, \phi_4, \phi_5 = v_e)$ (i.e., the orange line) as $\Pi_{DLSP}(v_s, v_e)$ if $|\Pi_{DLSP}(v_s, v_e)| < |\Pi_{LSP}(v_s, v_e)|$. In Figure 6 (b), we compare the weighted distance among $\Pi_{LSP}(v_p, v_n)$, $\Pi_l(v_p, v_n)$, and $\Pi_l(v_p, v_n)$, and substitute $\Pi_{LSP}(v_p, v_n)$ as the path with the shortest weighted distance.

So it could happen that after algorithm *DLSP* is used, $\Pi_{DLSP}(s, t)$ still passes the original vertices in V. For example, in Figure 6 (b), if $\Pi_{DLSP}(s, t)$ is still $(s, \phi_1, \phi_2, \phi_3, t)$, we need to divide $\Pi_{DLSP}(s, t)$ into two parts, i.e., $\Pi_{DLSP}(s, \phi_2)$ and $\Pi_{DLSP}(\phi_2, t)$, such that both the edge sequences S_1 and S_2 corresponding to $\Pi_{DLSP}(s, \phi_2)$ and $\Pi_{DLSP}(\phi_2, t)$ are full edge sequences, respectively. Then, we use S_1 and S_2 in algorithm *EWSL*, respectively. After using algorithm *EWSL*, we combine two result paths into one path by using ϕ_2 as the connecting point.

B REASON ON THE GOOD PERFORMANCE OF ALGORITHM *EWSL*

We use a 1D example to illustrate why algorithm *EWSL* could prune out unnecessary checking in algorithm *BSSL*. 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 ψ_1 is 87.32. Then, using algorithm *BSSL*, the searching interval will be $[50, 100]$, $[75, 100]$, $[75, 87.5]$, $[81.25, 87.5]$, $[84.375, 87.5]$, $[85.9375, 87.5]$, $[86.71875, 87.5]$, $[87.109375, 87.5] \cdots$. In algorithm *EWSL*, assume that we still need to use several iterations of algorithm *BSSL* to let Π_m pass the whole S, 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. Assume we calculate m_{ef} as 87 using effective weight pruning technique. As a result, in the next checking, we could directly limit the searching interval to be $[87, 87.5]$, which could 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, algorithm *EWSL* could save the running time and memory usage.

C EMPIRICAL STUDIES

C.1 Experimental Results

(1) Figure 16 and Figure 17, (2) Figure 18 and Figure 19, (3) Figure 20 and Figure 21 show the result on the *BH-small* dataset when varying ϵ, ϵ_{SP}, and ϵ_{SL}, respectively. (4) Figure 22 and Figure 23, (5) Figure 24 and Figure 25, (6) Figure 26 and Figure 27 show the result on the *BH* dataset when varying ϵ, ϵ_{SP}, and ϵ_{SL}, respectively. (7) Figure 9 and Figure 28, (8) Figure 29 and Figure 30, (9) Figure 31 and Figure 32 show the result on the *EP-small* dataset when varying ϵ, ϵ_{SP}, and ϵ_{SL}, respectively. (10) Figure 33 and Figure 34, (11) Figure

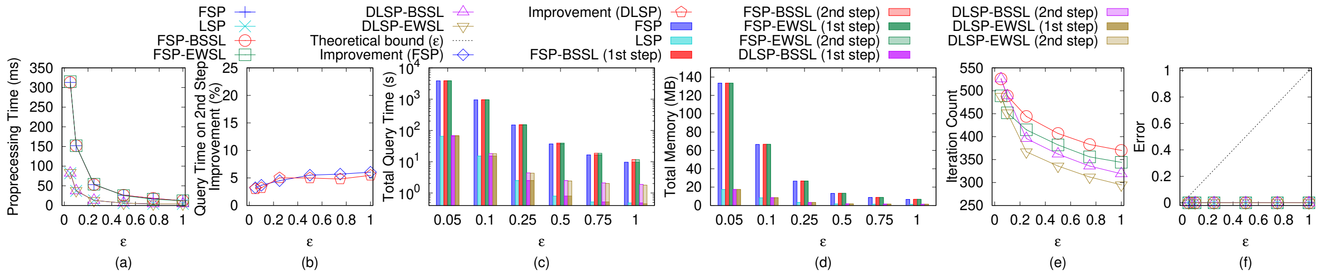

Figure 16: Effect of ϵ on *BH-small* dataset

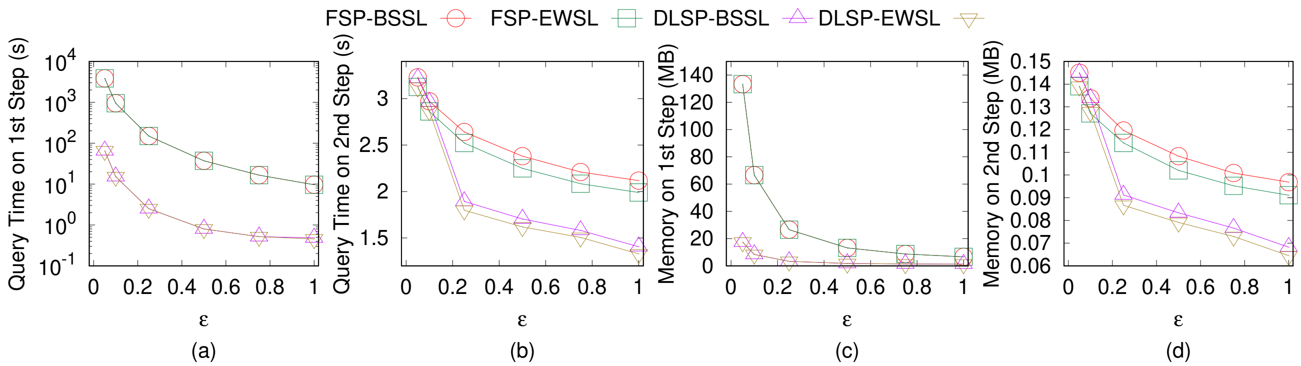

Figure 17: Effect of ϵ on *BH-small* dataset with separated query time and memory usage in two steps

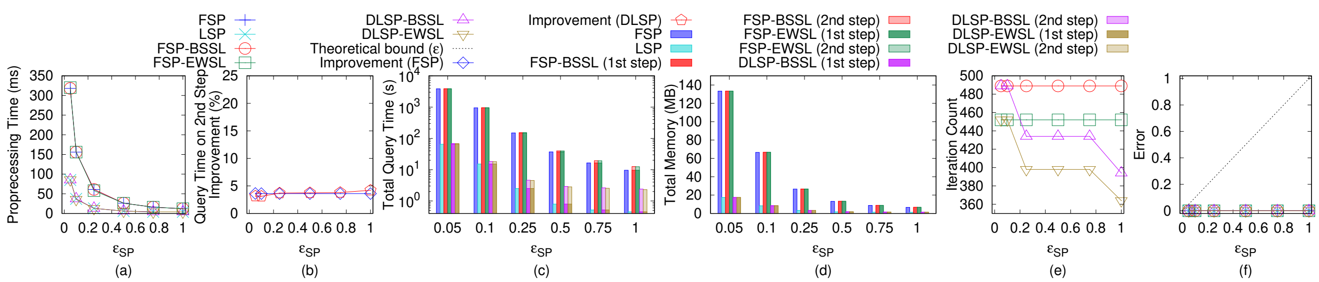

Figure 18: Effect of ϵ_{SP} on *BH-small* dataset

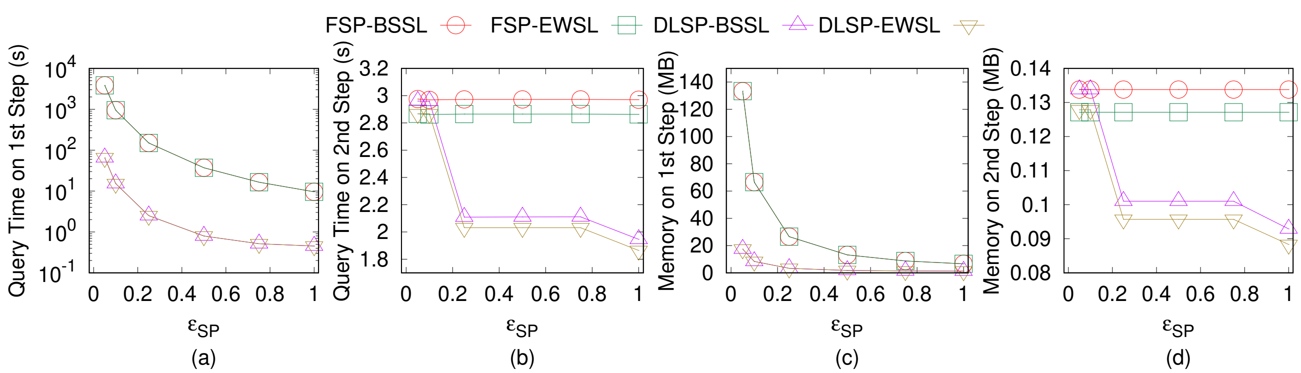

Figure 19: Effect of ϵ_{SP} on *BH-small* dataset with separated query time and memory usage in two steps

35 and Figure 36, (12) Figure 37 and Figure 38 show the result on the *EP-small* dataset when varying ϵ, ϵ_{SP}, and ϵ_{SL}, respectively. (13) Figure 39 and Figure 40 show the result on a set of small-version

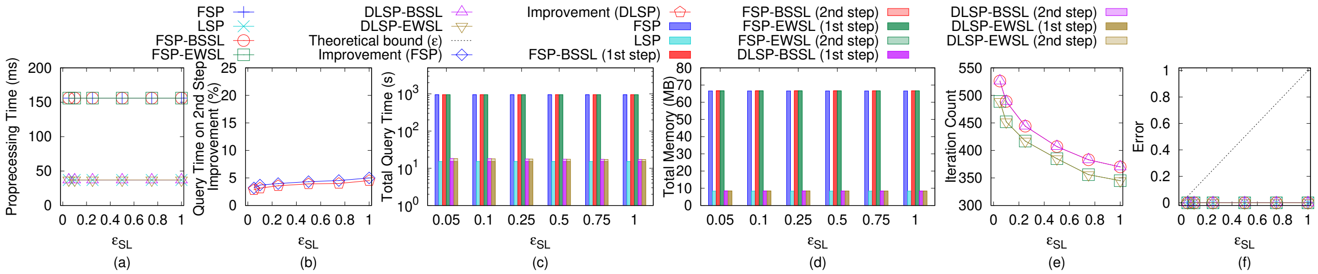

Figure 20: Effect of ϵ_{SL} on *BH-small* dataset

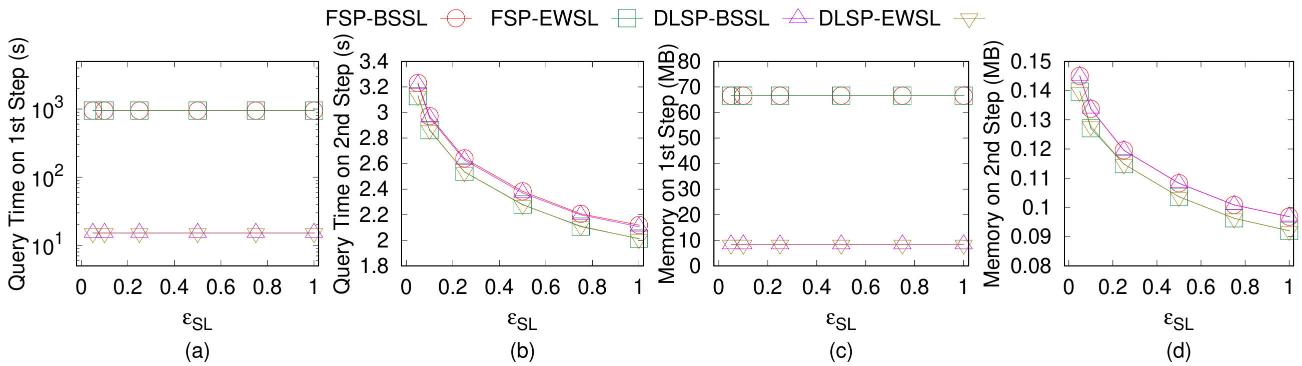

Figure 21: Effect of ϵ_{SL} on *BH-small* dataset with separated query time and memory usage in two steps

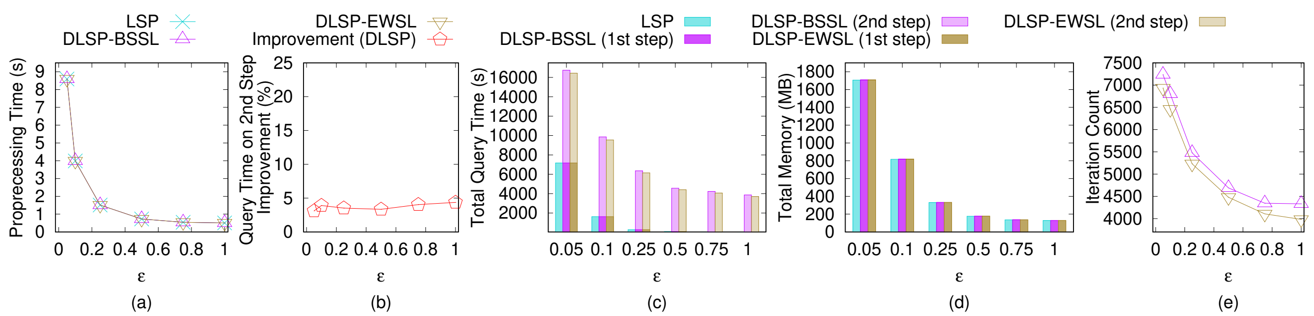

Figure 22: Effect of ϵ on *BH* dataset

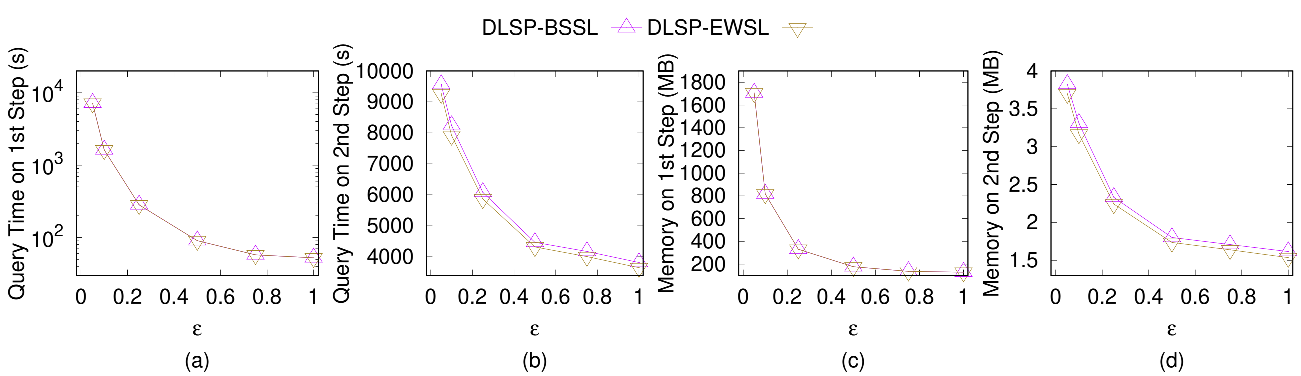

Figure 23: Effect of ϵ on *BH* dataset with separated query time and memory usage in two steps

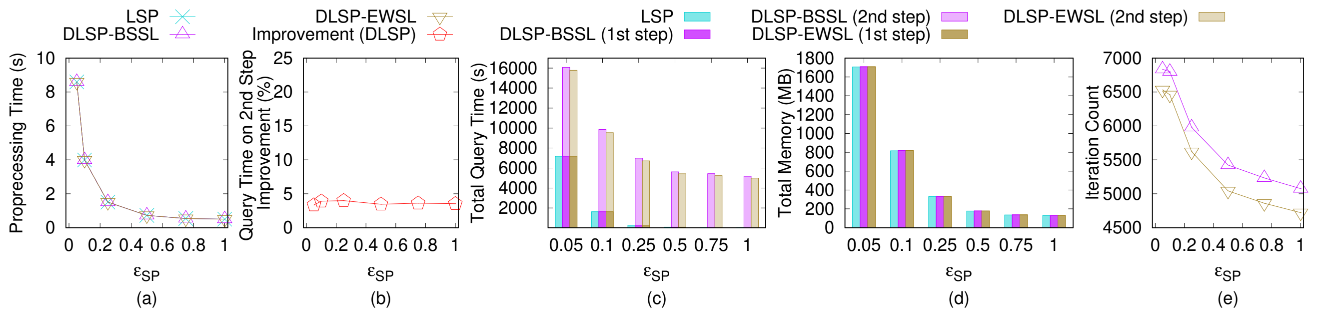

Figure 24: Effect of ϵ_{SP} on *BH* dataset

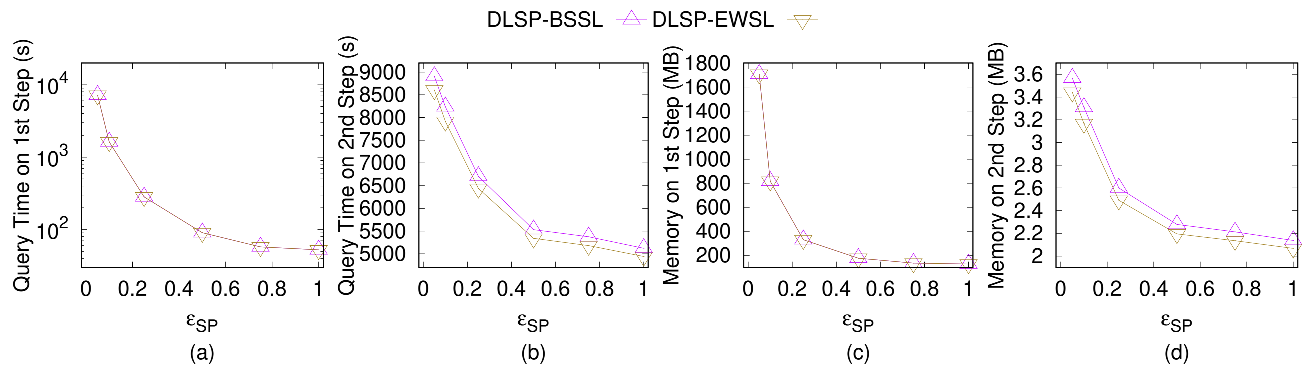

Figure 25: Effect of ϵ_{SP} on *BH* dataset with separated query time and memory usage in two steps

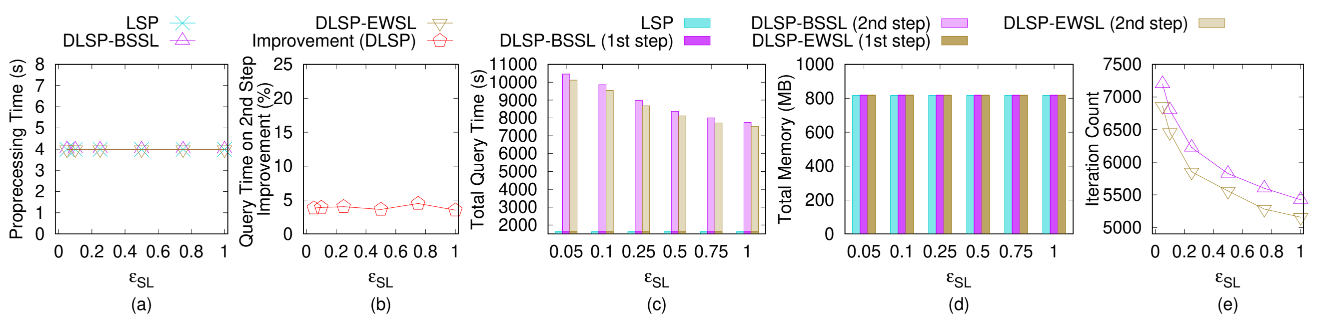

Figure 26: Effect of ϵ_{SL} on *BH* dataset

datasets when varying *DS*. (14) Figure 41 and Figure 42 show the result on a set of large-version datasets when varying *DS*.

Effect of ϵ. In Figure 16, Figure 22, Figure 9 and Figure 33, 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 $\epsilon_{SP} = \epsilon_{SL} = \epsilon$. Figure 17, Figure 23, Figure 28 and Figure 34 are the separated query time and memory usage in two steps for these results. In terms of preprocessing time, (the first step, the second step and the total) query time, (the first step, the second step and the total) memory usage and iteration count, algorithm *DLSP-EWSL* is the best one given that the path need to follow Snell's law. The errors of all the six algorithms are very small (close to 0%) and much smaller than the theoretical bound.

Effect of ϵ_{SP}. In Figure 18, Figure 24, Figure 29 and Figure 35, we tested 6 values of ϵ_{SP} from $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting ϵ_{SL} to be 0.1 as default value for all the cases. Figure 19, Figure 25, Figure 30 and Figure 36 are the separated query time and memory usage in two steps for these results. The preprocessing time, (the first step and the total) query time, and (the first step and the total) memory usage of algorithm *DLSP* (i.e., *DLSP-BSSL* and *DLSP-EWSL*) and *LSP* are 2-3 orders of magnitude smaller than *FSP* (i.e., *FSP*, *FSP-BSSL* and *FSP-EWSL*). Theoretically, the second step query time, the second step memory usage and iteration count should not change since ϵ_{SP} will not affect the second step. But, with a larger ϵ_{SP}, the edge sequence found by algorithm *FSP* and algorithm *DLSP* will become simpler, thus these term will reduce.

Effect of ϵ_{SL}. In Figure 20, Figure 26, Figure 31 and Figure 37, we tested 6 values of ϵ_{SL} from $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on *BH-small*, *BH*, *EP-small* and *EP* datasets by setting ϵ_{SP} to be 0.1 as default

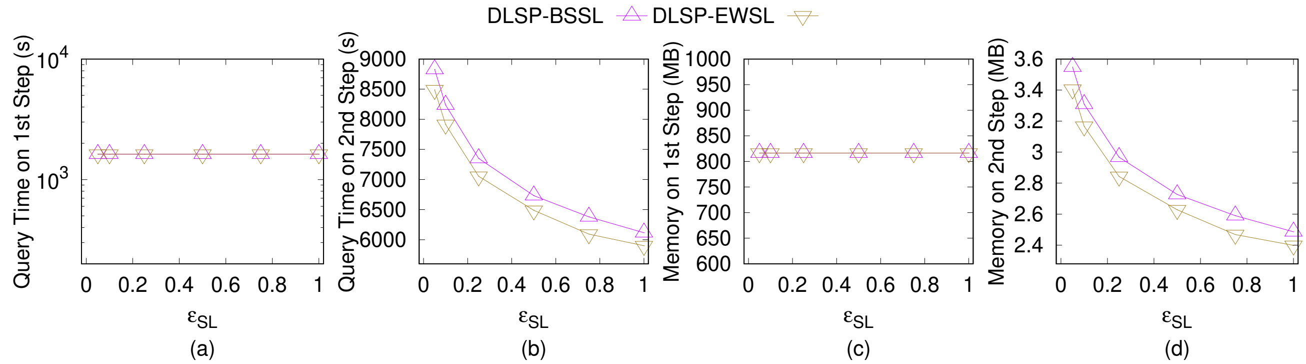

Figure 27: Effect of ϵ_{SL} on *BH* dataset with separated query time and memory usage in two steps

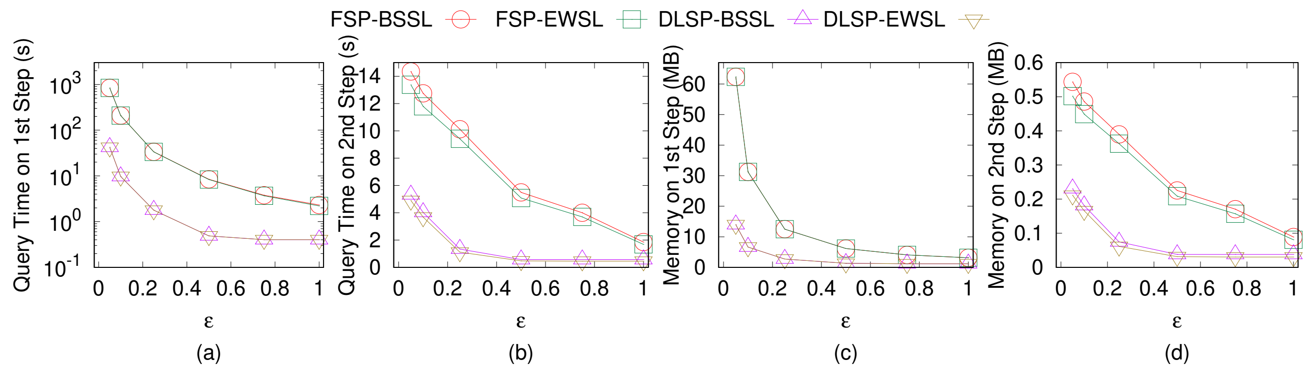

Figure 28: Effect of ϵ on *EP-small* dataset with separated query time and memory usage in two steps

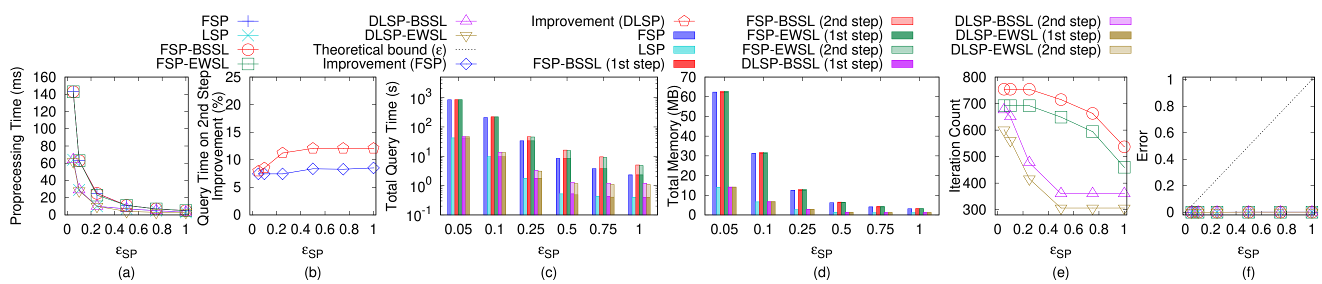

Figure 29: Effect of ϵ_{SP} on *EP-small* dataset

value for all the cases. Figure 21, Figure 27, Figure 32 and Figure 38 are the separated query time and memory usage in two steps for these results. The preprocessing time, the first step query time, and the first step memory usage remain unchanged since ϵ_{SL} will not affect these terms. ϵ_{SL} will only affect the second step query time, the second step memory usage and the iteration count, and they will decrease when ϵ_{SL} is increasing.

Effect of *DS* (scalability test). In Figure 39 and Figure 41, we tested 5 values of *DS* from {10k, 20k, 30k, 40k, 50k} on the a of small-version datasets (by setting ϵ to be 0.1) and {200k, 400k, 600k, 800k, 1000k} on a set of large-version datasets (by setting ϵ to be 0.25) for scalability test. Figure 40 and Figure 42 are the separated query time and memory usage in two steps for these results. On the set of small-version datasets, algorithm *DLSP-EWSL* could still superior perform the remaining algorithms in terms of the preprocessing time, (the first step, the second step and the total) query time and (the first step, the second step and the total) memory usage given that the path need to follow Snell's law. When the dataset size is 50k, the state-of-the-art algorithm's (i.e., algorithm *FSP-BSSL*) total query time is 119,000s (≈ 1.5 day) and total memory usage is 2.9GB, while our algorithm's (i.e., algorithm *DLSP-EWSL*) total query time is 534s (≈ 9 min) and total memory usage is 130MB, which shows the excellent performance of our algorithm. On the set of large-version datasets, algorithm *DLSP-EWSL* could still return a path that follows Snell's law in reasonable time.

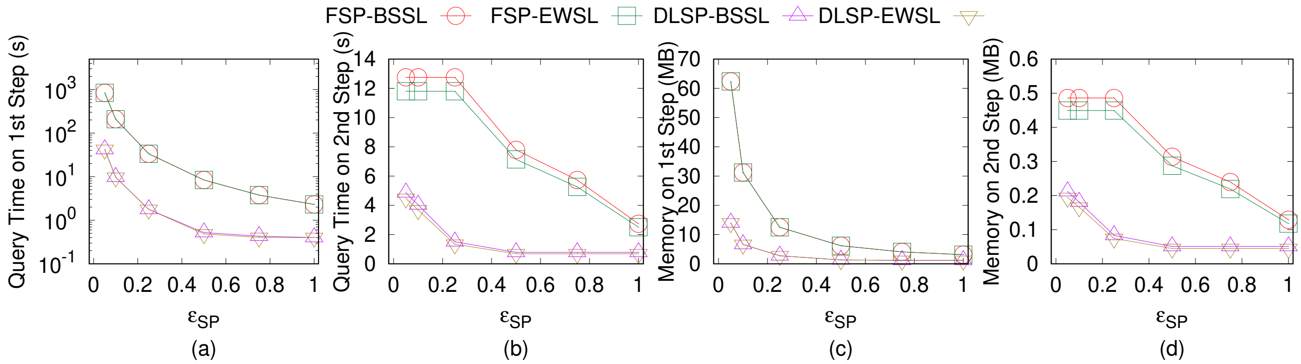

Figure 30: Effect of ϵ_{SP} on *EP-small* dataset with separated query time and memory usage in two steps

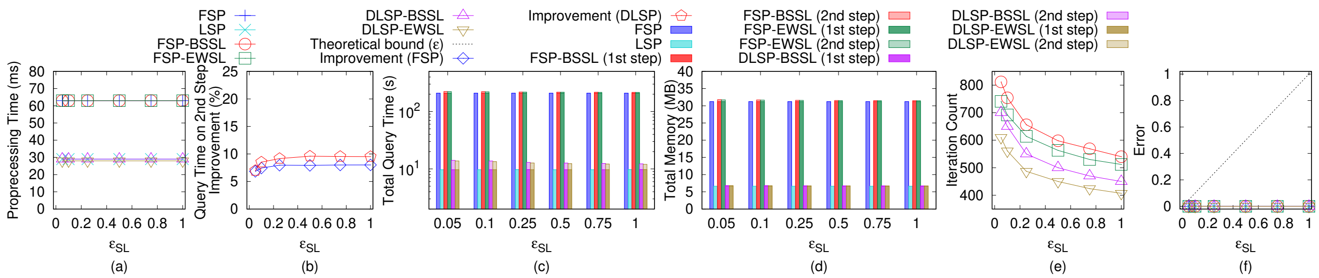

Figure 31: Effect of ϵ_{SL} on *EP-small* dataset

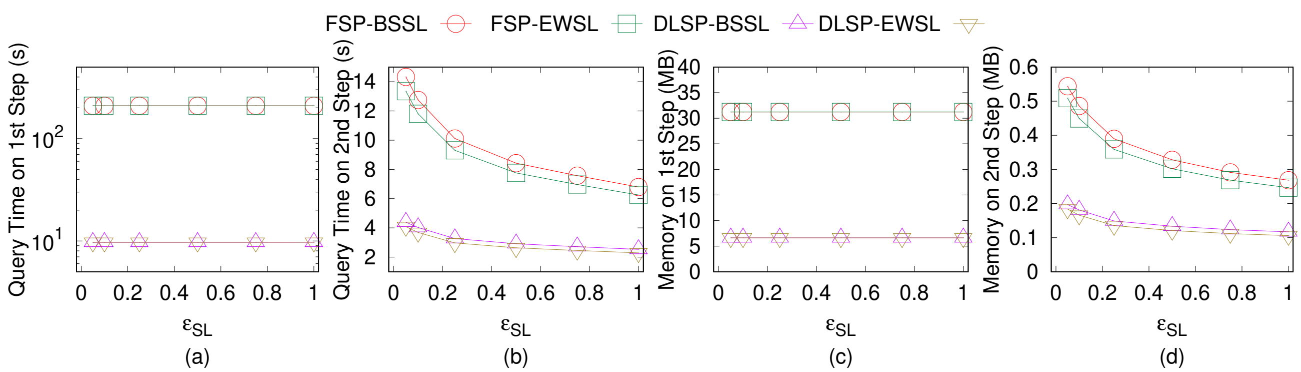

Figure 32: Effect of ϵ_{SL} on *EP-small* dataset with separated query time and memory usage in two steps

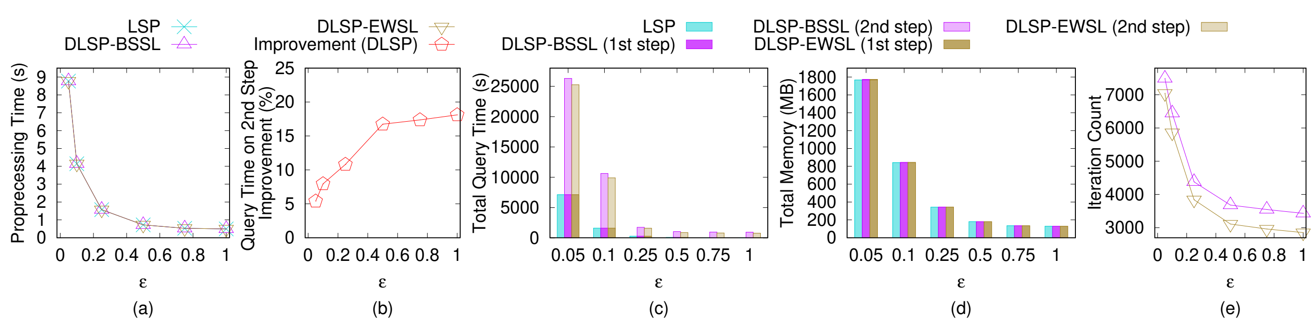

Figure 33: Effect of ϵ on *EP* dataset

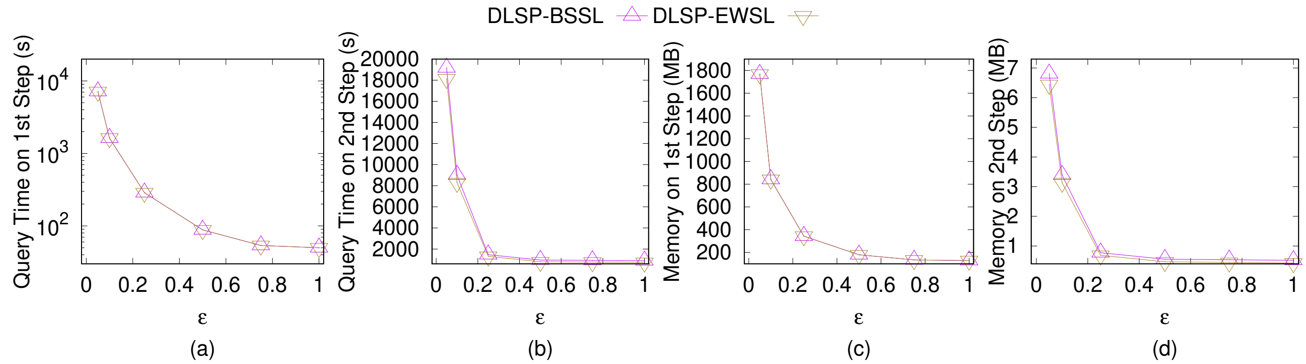

Figure 34: Effect of ϵ on *EP* dataset with separated query time and memory usage in two steps

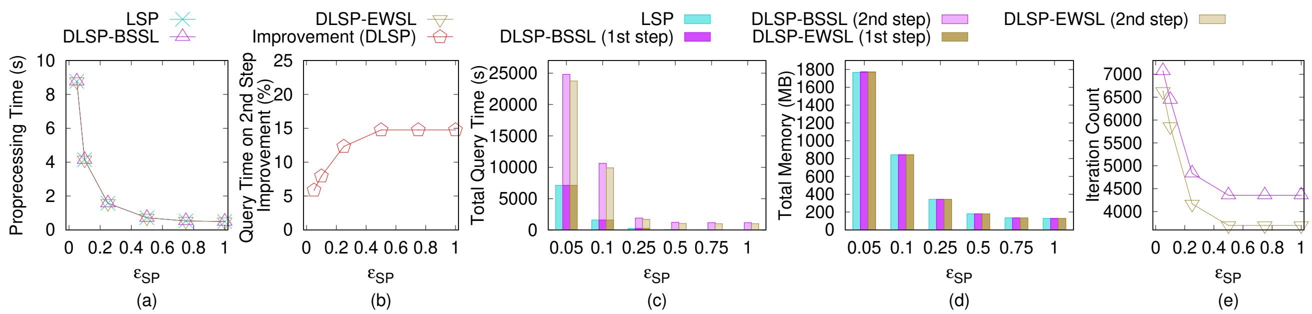

Figure 35: Effect of ϵ_{SP} on *EP* dataset

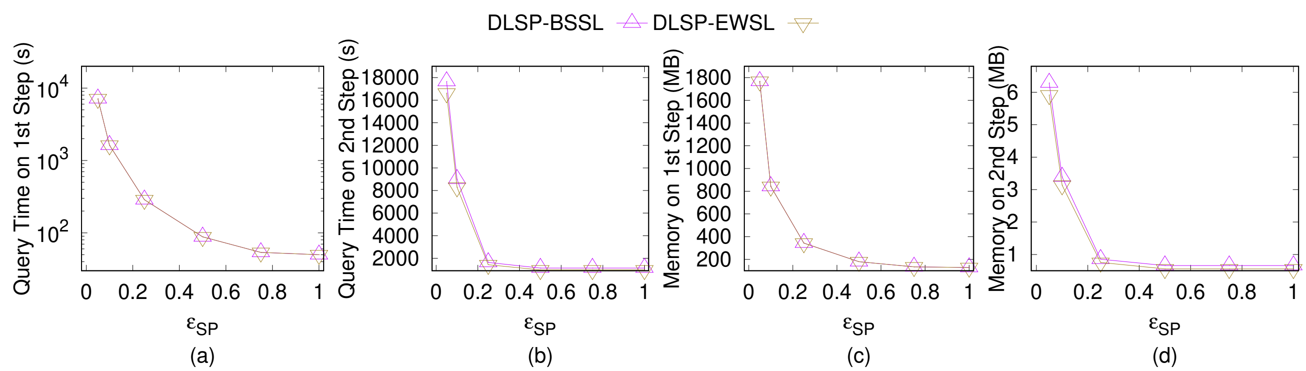

Figure 36: Effect of ϵ_{SP} on *EP* dataset with separated query time and memory usage in two steps

C.2 Generating datasets with different dataset sizes

The procedure for generating the datasets with different dataset sizes is as follows. We mainly follow the procedure for generating datasets with different dataset sizes in the work [25, 30]. 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 $\sigma_N^2 = (\frac{\sum_{v_o \in V_o}(x_{v_o}-\overline{x})^2}{(ex_t+1)\cdot(ey_t+1)}, \frac{\sum_{v_o \in V_o}(y_{v_o}-\overline{y})^2}{(ex_t+1)\cdot(ey_t+1)})$. In the end, we project each generated point (x, y) to the surface of T_o and take the projected point as the newly generate T_t.

D PROOFS

Lemma D.1. *There are at most $k_{SP} \leq 2(1 + \log_\lambda \frac{L}{r})$ Steiner points on each edge in E using algorithm LSP.*

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

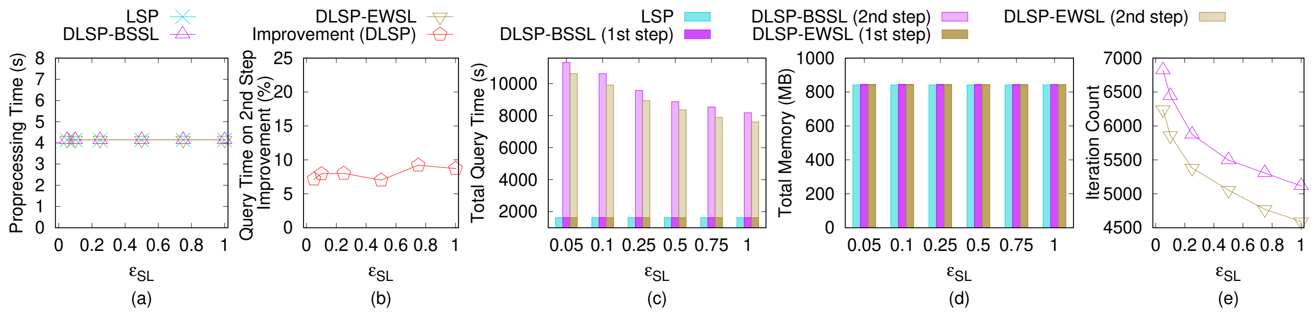

Figure 37: Effect of ϵ_{SL} on *EP* dataset

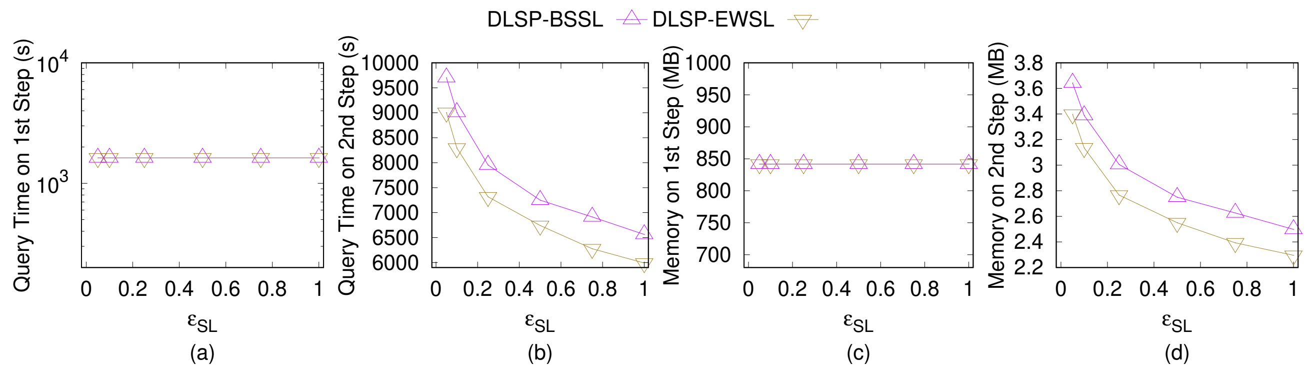

Figure 38: Effect of ϵ_{SL} on *EP* dataset with separated query time and memory usage in two steps

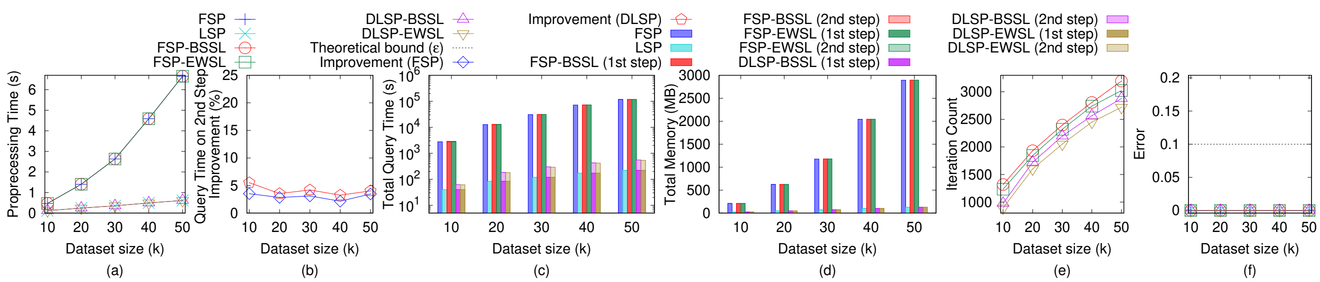

Figure 39: Effect of dataset size on a set of small-version datasets

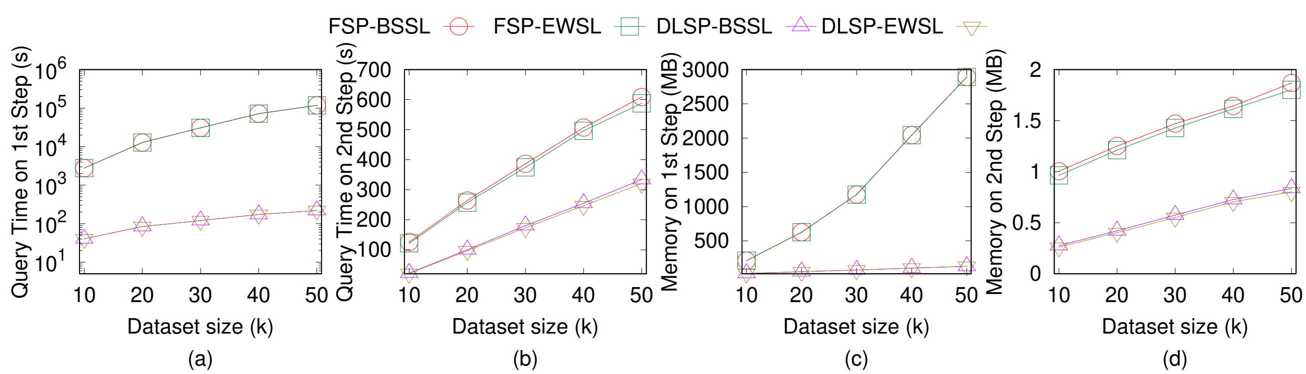

Figure 40: Effect of dataset size on a set of small-version datasets with separated query time and memory usage in two steps

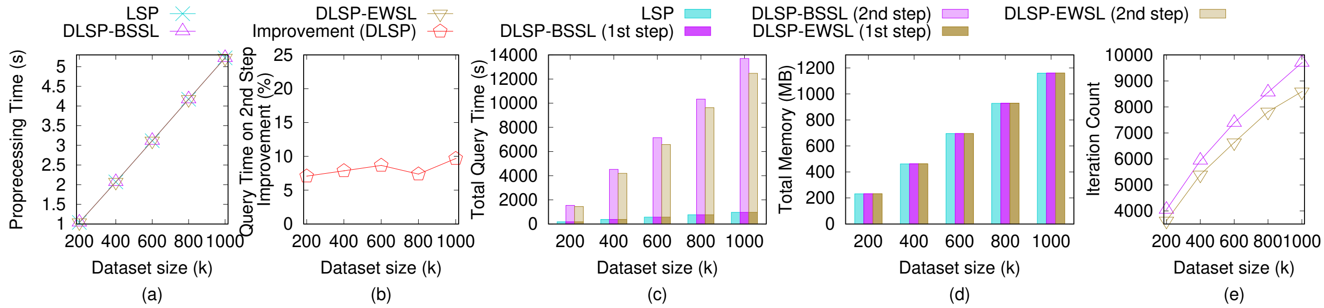

Figure 41: Effect of dataset size on a set of large-version datasets

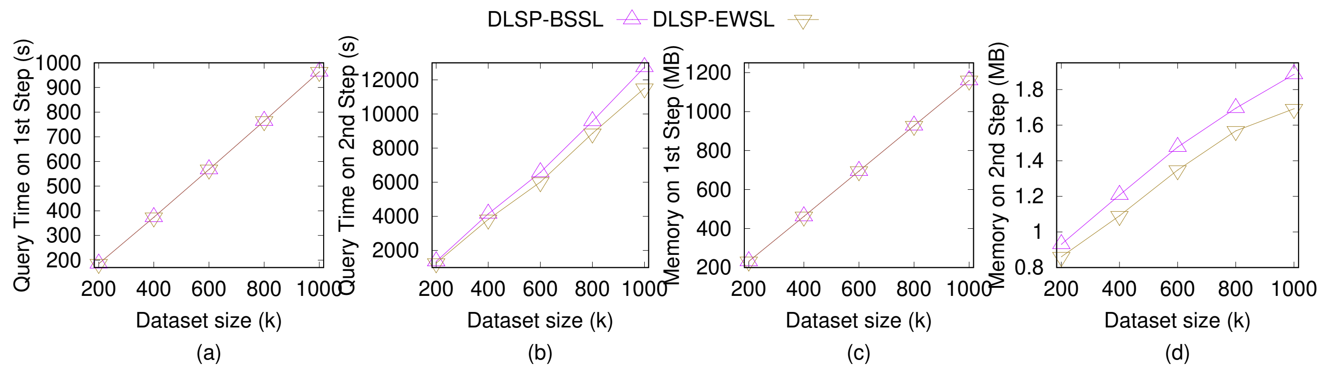

Figure 42: Effect of dataset size on a set of large-version datasets with separated query time and memory usage in two steps

endpoints, we have two sets of Steiner points from both endpoints, and we have the factor 2. From algorithm *LSP*, each set of Steiner points contains at most $(1+\log_\lambda \frac{L}{r})$ Steiner points, where the 1 comes from the first Steiner point that is the nearest one from the endpoint. Therefore, we have $k_{SP} \leq 2(1+\log_\lambda \frac{L}{r})$. □

Lemma D.2. *In algorithm LSP, $\epsilon'_{SP} = \frac{1+\epsilon_{SP}+\frac{W}{w}-\sqrt{(1+\epsilon_{SP}+\frac{W}{w})^2-4\epsilon_{SP}}}{4}$ with $0 < \epsilon'_{SP} < \frac{1}{2}$ and $\epsilon_{SP} > 0$ after we express ϵ'_{SP} in terms of ϵ_{SP}.*

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

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

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

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

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

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

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

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

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

Lemma D.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 $h = \min_{a,b} \frac{|a\times b|}{|b|} = \min_{a,b} \frac{\sqrt{\eta}}{\sqrt{x_a^2+y_a^2+z_a^2}} \geq \frac{1}{N\sqrt{3}} \min_{a,b} \sqrt{\eta} \geq \frac{1}{N\sqrt{3}}$, where $\eta = (y_a z_b - z_a y_b)^2 + (z_a x_b - x_a z_b)^2 + (x_a y_b - y_a x_b)^2$. □

Proof of Theorem 4.1. Firstly, we proof the running time and memory usage. Following Lemma D.1, the number of Steiner points k_{SP} on each edge is $O(\log_\lambda \frac{L}{r})$, where $\lambda = (1+\epsilon'_{SP}\cdot\sin\theta)$ and $r = \epsilon'_{SP} h$. Following Lemma D.2 and Lemma D.3, $r = O(\frac{\epsilon_{SP}}{N})$, and thus $k_{SP} = O(\log \frac{LN}{\epsilon_{SP}})$. So $|V_k| = O(n \log \frac{LN}{\epsilon_{SP}})$. Since we know for a graph with n' vertices, the running time and memory usage of Dijkstra algorithm on this graph are $O(n' \log n')$ and n', so the running time of our algorithm is $O(n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}))$ and the memory usage of our algorithm is $O(n \log \frac{LN}{\epsilon_{SP}})$.

Secondly, we proof the error bound. A proof sketch of this could be found in Theorem 1 of [8] and a detailed proof could be found in Theorem 3.1 of [23]. But, in [8, 23], they have $|\Pi_{LSP}(s,t)| \leq (1+(2+\frac{2W}{(1-2\epsilon'_{SP})\cdot w})\epsilon'_{SP})|\Pi^*(s,t)|$ where $0 < \epsilon'_{SP} < \frac{1}{2}$. After substituting $(2+\frac{2W}{(1-2\epsilon'_{SP})\cdot w})\epsilon'_{SP} = \epsilon_{SP}$ with $0 < \epsilon'_{SP} < \frac{1}{2}$ and $\epsilon_{SP} > 0$, we have $|\Pi_{LSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)|$ where $\epsilon_{SP} > 0$. □

Proof of Theorem 4.2. Firstly, we proof the average case running time. For the average case, we assume $\frac{1}{3}$ of $\Pi_{LSP}(s,t)$ passes on the edge (i.e., no need use divide-and-conquer step for refinement), $\frac{1}{3}$ of $\Pi_{LSP}(s,t)$ belongs to single endpoint case, and the remaining $\frac{1}{3}$ of $\Pi_{LSP}(s,t)$ belongs to successive endpoint case. Let the number of path segments in $\Pi_{LSP}(s,t)$ be l and we know $l = O(n^2)$. Let $T_{SLP} = O(n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}))$ be the running time of algorithm *LSP*, whose running time will not affected by l. Let $T_{avg}(l)$ be the average case running time for the divide-and-conquer step in terms of l and $T_{avg}(n)$ be the average case running time for the divide-and-conquer step in terms of n. Then, $T_{avg}(l) = T_{SLP} + T_{avg}(\frac{l}{3}) + O(\frac{\zeta l}{3}) = T_{SLP} + O(\zeta l)$, and $T_{avg}(n^2) = T_{SLP} + O(\zeta n^2)$. Therefore, $T_{avg}(n) = T_{SLP} + O(\zeta n) = O(n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}) + \zeta n)$.

Secondly, we then proof the worst case running time. The worst case would be that all the points in $\Pi_{LSP}(s,t)$ passes the original vertices in V (i.e., all the vertices are successive endpoints), and there will be $O(n)$ of these points. Then, the divide-and-conquer step is applied, and only one point is refined (i.e., only one point in $\Pi_{LSP}(s,t)$ is refined on the edge). Let $T_{worst}(l)$ be the worst case running time for the divide-and-conquer step in terms of l and $T_{worst}(n)$ be the average case running time for the divide-and-conquer step in terms of n. Then, $T_{worst}(l) = T_{SLP} + T_{worst}(l-1) = O(l) \cdot T_{SLP}$, and $T_{worst}(n^2) = O(n^2) \cdot T_{SLP}$. Therefore, $T_{worst}(n) = O(n) \cdot T_{SLP} = O(n^2 \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}))$.

Thirdly, we proof the memory usage. Algorithm *LSP* needs $O(n \log \frac{LN}{\epsilon_{SP}})$ memory since it is a common Dijkstra algorithm, whose memory usage is $O(|V_k|)$, where $|V_k|$ is size of vertices in the Dijkstra algorithm. Handling one single endpoint case needs $O(1)$ memory. Since there could be at most n single endpoint cases, the memory usage is $O(n)$. Handling successive endpoint cases needs $O(n)$ memory since divide-and-conquer step needs $O(n)$ memory. Therefore, the total memory usage is $O(n \log \frac{LN}{\epsilon_{SP}})$.

Finally, we proof the error bound. In algorithm *DLSP*, we only use the refinement path $\Pi_{DLSP}(s,t)$ if its weighted distance is shorter than $\Pi_{LSP}(s,t)$ (i.e., $|\Pi_{DLSP}(s,t)| \leq |\Pi_{LSP}(s,t)|$). So, using Theorem 4.1, we know $|\Pi_{DLSP}(s,t)| \leq |\Pi_{LSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)|$. In other words, even if $\Pi_{DLSP}(s,t)$ could avoid lying on the original vertices in V (and actually lying on the edges in E), but $|\Pi_{DLSP}(s,t)| > |\Pi_{LSP}(s,t)|$, then we will not use $\Pi_{DLSP}(s,t)$ for calculating the edge sequence S, we still use $\Pi_{LSP}(s,t)$. In this case, the error ratio is still the error ratio of algorithm *LSP*, i.e., $|\Pi_{LSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)|$. □

Proof of Theorem 4.3. Firstly, we proof the running time and memory usage. Let l be the number of edges in S. For the running time, it first takes $O(l)$ time for computing the 3D surface Snell ray Π_m since there are l edges in S and we need to use Snell's law l times to calculate the passing point on each edge. It then takes $O(\log \frac{L_i}{\delta})$ time for deciding the position of m_i because we stop the iteration when $|a_i b_i| < \delta$, and it is a binary search approach, where L_i is the length of e_i. Since $\delta = \frac{h\epsilon_{SL} w}{6lW}$ and $L_i \leq L$ for $\forall i \in \{1,\ldots,l\}$, the running time for this step is $O(\log \frac{lWL}{h\epsilon_{SL} w})$. Since we run the above two nested loop l times, so the total running time is $O(l^2 \log \frac{lWL}{h\epsilon_{SL} w})$. According to Lemma 7.1 in [26], $l = O(n^2)$, so the running time of algorithm *BSSL* is $O(n^4 \log \frac{nWL}{h\epsilon_{SL} w})$. For the memory usage, since the weighted shortest path that follows Snell's law will pass l edges, so the memory usage is $O(l) = O(n^2)$.

Secondly, we proof the error bound. Since $s = \rho_0$, proving $|\Pi_{SL}(s,t|S)| \leq (1+\epsilon_{SL})|\Pi^*(s,t|S)|$ is equivalent to prove $|\Pi_{SL}(\rho_0,t|S)| \leq (1+\epsilon_{SL})|\Pi^*(\rho_0,t|S)|$. We will convert it in terms of δ, and prove it by induction for $i \in \{0,1,2,\ldots,l\}$, there are three steps:

$$|\Pi_{SL}(\rho_i,t|S)| \leq (1+\frac{\epsilon}{2})|\Pi^*(\rho_i,t|S)| + 3(l-i)\delta W \quad (1)$$

Step one: we have $|\Pi_{SL}(\rho_l,t|S)| = w_l|\rho_l t| = |\Pi^*(\rho_l,t|S)| \leq (1+\frac{\epsilon_{SL}}{2})|\Pi^*(\rho_l,t|S)|$ when $i = l$. So the Equation 1 holds for $i = l$.

Step two: we assume that the Equation 1 holds for $i = k+1$, that is, we assume that $|\Pi_{SL}(\rho_{k+1},t|S)| \leq (1+\frac{\epsilon_{SL}}{2})|\Pi^*(\rho_{k+1},t|S)| + 3(l-k-1)\delta W$, and we hope to prove that the inequality holds for $i = k$. So for $i = k$, we have the following equation:

$$\begin{aligned}
&|\Pi_{SL}(\rho_k,t|S)| \\
&= w_k|\rho_k\rho_{k+1}| + |\Pi_{SL}(\rho_{k+1},t|S)| \\
&\leq w_k|\rho_k\rho_{k+1}| + (1+\frac{\epsilon_{SL}}{2})|\Pi^*(\rho_{k+1},t|S)| + 3(l-k-1)\delta W \\
&\leq w_k|\rho_k\rho_{k+1}| + (1+\frac{\epsilon_{SL}}{2})[|\Pi^*(\psi_{k+1},t|S)| + w_k|\rho_{k+1}\psi_{k+1}|] \\
&+ 3(l-k-1)\delta W \\
&\leq [w_k|\rho_k\psi_{k+1}| + w_k|\rho_{k+1}\psi_{k+1}|] + (1+\frac{\epsilon_{SL}}{2})[|\Pi^*(\psi_{k+1},t|S)| \\
&+ w_k|\rho_{k+1}\psi_{k+1}|] + 3(l-k-1)\delta W \\
&= w_k|\rho_k\psi_{k+1}| + (1+\frac{\epsilon_{SL}}{2})|\Pi^*(\psi_{k+1},t|S)| \\
&+ (2+\frac{\epsilon_{SL}}{2})w_k|\rho_{k+1}\psi_{k+1}| + 3(l-k-1)\delta W \\
&\leq w_k(1+\frac{\epsilon_{SL}}{2})|\rho_k\psi_{k+1}| + (1+\frac{\epsilon_{SL}}{2})|\Pi^*(\psi_{k+1},t|S)| \\
&+ 3w_k|\rho_{k+1}\psi_{k+1}| + 3(l-k-1)\delta W \\
&\leq w_k(1+\frac{\epsilon_{SL}}{2})|\rho_k\psi_{k+1}| + (1+\frac{\epsilon_{SL}}{2})|\Pi^*(\psi_{k+1},t|S)| \\
&+ 3W\delta + 3(l-k-1)\delta W \\
&= (1+\frac{\epsilon_{SL}}{2})[w_k|\rho_k\psi_{k+1}| + |\Pi^*(\psi_{k+1},t|S)|] \\
&+ [3W\delta + 3(l-k-1)\delta W] \\
&= (1+\frac{\epsilon_{SL}}{2})|\Pi^*(\rho_k,t|S)| + 3(l-k)\delta W
\end{aligned}$$

Step three: by induction, we have finished proving Equation 1. By setting $k = 0$ and since we set $\delta = \frac{h\epsilon_{SL} w}{6lW}$, we have $|\Pi_{SL}(s,t|S)| \leq (1+\frac{\epsilon_{SL}}{2})|\Pi^*(s,t|S)| + 3l\delta W = (1+\frac{\epsilon_{SL}}{2})|\Pi^*(s,t|S)| + wh\frac{\epsilon_{SL}}{2} \leq (1+\epsilon_{SL})|\Pi^*(s,t|S)|$, where $\epsilon_{SL} > 0$. Note that the last inequality comes from the fact that wh must certainly be a lower bound on $|\Pi^*(s,t|S)|$. This is because for a path that pass a triangle (start from one vertex of the triangle), its length should be at least the minimum height in this triangle. Since the face has a weight, so $wh \leq |\Pi^*(s,t|S)|$. And we finish the proof. □

Proof of Theorem 4.4. Firstly, we proof the running time and memory usage. Since algorithm *DLSP* and algorithm *EWSL* are two independent steps, so the total running time and total memory usage is the sum of the running time and the memory usage for these two steps using Theorem 4.2 and Theorem 4.3. Since we let $\epsilon = \epsilon_{SP} = \epsilon_{SL}$, we could get the running time and the memory usage in terms of ϵ.

Secondly, we proof the error bound. Following Theorem 4.2 and Theorem 4.3, we have proved that $|\Pi_{LSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)|$ and $|\Pi_{SL}(s,t|S)| \leq (1+\epsilon_{SL})|\Pi^*(s,t|S)|$, where $\epsilon_{SP} > 0$ and $\epsilon_{SL} > 0$. Depending on whether the edge sequence S found by $\Pi_{DLSP}(s,t)$ is the same as the optimal edge sequence S^* that $\Pi^*(s,t)$ passes, and whether the path $\Pi_{DLSP}(s,t)$ found by algorithm *DLSP* is longer or the path $\Pi_{SL}(s,t|S)$ found by SL is longer, there are four cases as follows.

Case one: $S = S^*$ and $|\Pi_{SL}(s,t|S)| \leq |\Pi_{DLSP}(s,t)|$ (which is the most common case): $|\Pi(s,t)| = \min(|\Pi_{DLSP}(s,t)|, |\Pi_{SL}(s,t|S)|) = |\Pi_{SL}(s,t|S)| \leq (1+\epsilon_{SL})|\Pi^*(s,t|S)| = (1+\epsilon)|\Pi^*(s,t|S)| = (1+\epsilon)|\Pi^*(s,t|S^*)| = (1+\epsilon)|\Pi^*(s,t)|$. Note that the last equality (i.e., $|\Pi^*(s,t|S^*)| = |\Pi^*(s,t)|$) comes from the fact that $\Pi^*(s,t|S^*) = \Pi^*(s,t)$, since S^* is a sequence of edges that $\Pi^*(s,t)$ from s to t need to pass in order, and these two terms are actually the same thing.

Case two: $S = S^*$ and $|\Pi_{SL}(s,t|S)| > |\Pi_{DLSP}(s,t)|$: $|\Pi(s,t)| = \min(|\Pi_{DLSP}(s,t)|, |\Pi_{SL}(s,t|S)|) = |\Pi_{DLSP}(s,t)| \leq |\Pi_{LSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)| = (1+\epsilon)|\Pi^*(s,t)|$.

Case three: $S \neq S^*$ and $|\Pi_{SL}(s,t|S)| \leq |\Pi_{DLSP}(s,t)|$: $|\Pi(s,t)| = \min(|\Pi_{DLSP}(s,t)|, |\Pi_{SL}(s,t|S)|) \leq |\Pi_{DLSP}(s,t)| \leq |\Pi_{LSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)| = (1+\epsilon)|\Pi^*(s,t)|$.

Case four: $S \neq S^*$ and $|\Pi_{SL}(s,t|S)| > |\Pi_{DLSP}(s,t)|$: $|\Pi(s,t)| = \min(|\Pi_{DLSP}(s,t)|, |\Pi_{SL}(s,t|S)|) = |\Pi_{DLSP}(s,t)| \leq |\Pi_{LSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)| = (1+\epsilon)|\Pi^*(s,t)|$.

For all of these four cases, we have $|\Pi(s,t)| \leq (1+\epsilon)|\Pi^*(s,t)|$, and this concludes our proof. □