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 the shortest distance query on terrain data has aroused widespread concern in industry and academia. In this paper, we propose an efficient method for the *weighted region problem* on a three-dimensional (3D) weighted terrain surface. 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 solving the weighted region problem exactly when the number of faces in the terrain is larger than two, 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 on the weighted terrain surface that the shortest path passes, 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 step plus binary search Snell's Law* (*EWSL*), respectively. In both the theoretical 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. 2023. Finding Shortest Path on 3D Weighted Terrain Surface using Divide-and-Conquer and Effective Weight. In *Proceedings of 2024 International Conference on Management of Data (SIGMOD '24).* ACM, New York, NY, USA, 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In recent years, the digital terrain data becomes increasingly widespread in industry and academia [39]. In industry, many existing commercial companies/applications, such as Metaverse [6, 29, 30], Cyberpunk 2077 (a popular three-dimensional (3D) computer game) [2] and Google Earth [5], are using terrain data of objects such as mountains, valleys, and hills with different features (e.g., water and grassland) to help users reach the destination faster. In academia, researchers paid considerable attention to studying shortest path queries on terrain datasets [16, 23, 24, 32, 35, 37, 38, 40, 41]. A terrain surface is represented by a set of *faces* each of which corresponds to a triangle. Each face (or triangle) has three line segments called *edges* connected with each other at three *vertices*. Figure 1 shows an example of a terrain surface. The *weighted shortest path* 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 is assigned a *weight*, and the *unweighted shortest path* refers to the shortest path between s and t where each face weight is set to a fixed value (e.g. 1). In Figure 1, the blue (resp. yellow) line is the weighted (resp. unweighted) shortest path from s to t in this terrain surface. 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 terrain surface between s and t with different meanings of the face weights is involved in obstacle avoidance in path planning for autonomous vehicles, overland route-recommendation systems for human, laying pipelines or electrical cables [11, 21, 22, 26, 35, 41, 42], etc. In Figure 1, a robot wants to move on a 3D terrain surface from s to t which consists water (the faces with blue color) and grassland (the faces with green color), and avoid passing through the water. We could set the terrain faces corresponding to water (resp. grassland) with a larger (resp. smaller) weight. So, the weighted length of the path that passes water is larger, and the robot will choose the path that doesn't pass water (i.e., the weighted length is smaller). In addition, consider a real-life example for placement of undersea optical fiber cable on the seabed (i.e., a terrain). Nowadays, over 1.35×10^5km of undersea cables have been constructed [25]. We aim to minimize the weighted length of the cable for cost saving. For some regions with a deeper sea level, the hydraulic pressure is higher, and the cable's lifespan is reduced, so it is more expensive to repair and maintain the cable [13]. We set the terrain faces for this type of regions with a larger weight. So, we could avoid placing the cable on these regions, and reduce the cost. Our motivation study in Section 5.3.2 shows that for a cable that will be used for 100 years, the total estimated cost of the cable for following the weighted shortest path and the unweighted shortest path are USD \$366B and \$438B, respectively, which shows the usefulness of the weighted shortest path.

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

SIGMOD '24, June 11–16, 2024, Santiago, Chile

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

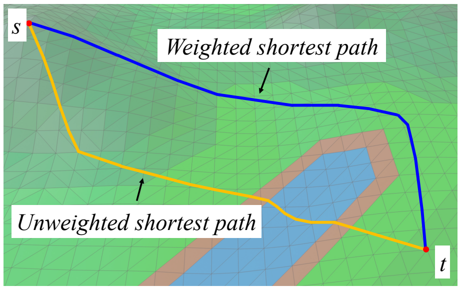

Figure 1: An example of terrain surface, unweighted shortest path and weighted shortest path

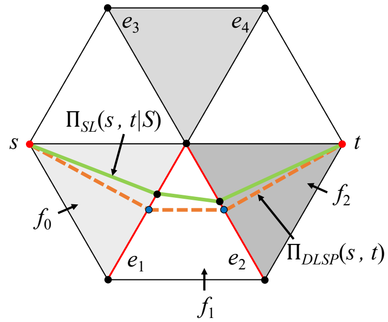

Figure 2: An example of $\Pi_{DLSP}(s,t)$ and $\Pi_{SL}(s,t|S)$ that passes $S=(e_1,e_2)$

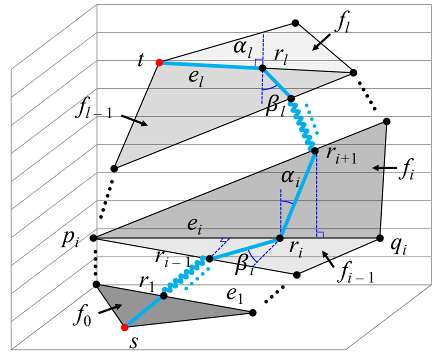

Figure 3: An example of $\Pi^*(s,t)$ in 3D that follows Snell's law

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 [33], which behaves like the refraction of a light ray when passing the boundary of two different media. In the weighted region problem, the weighted shortest path should follow Snell's law. This is because Snell's law is a critical optimization measurement of the weighted shortest path [33]. If the calculated weighted path does not follow Snell's law, then it must exist a shorter weighted path that follows Snell's law. In Figure 2, the green line is the weighted shortest path which follows Snell's law from s to t that passes edges e_1, e_2 in order, and the orange dashed line is the path that does not follow Snell's law. Clearly, the green line is shorter than the orange dashed line. But, if the weighted shortest path passes the boundary between two faces with different weights, it will not result in a straight line, and will bend when it crosses an edge. Due to this, in order to calculate the weighted shortest path 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 based on T, 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 [15], there is no exact solution for solving the weighted region problem exactly when the number of faces in the terrain is larger than two, and most (if not all) existing algorithms aim to calculating the weighted shortest path on the weighted region problem approximately. We then give two criteria of a good algorithm for solving the weighted region problem:

- **Snell's law criterion**: The algorithm should calculate the weighted shortest path which follows Snell's law. The weighted path that follows Snell's law will have a shorter distance (i.e., a smaller error) compared with a path that does not follow Snell's law [33]. Our motivation study in Section 5.3.2 shows that for the placement of undersea optical fiber cable on the seabed, the path that follows Snell's law could save USD $1 billion compared with the path does not follow Snell's law (even though the former one's length is only 0.8km shorter than the latter one's).
- **Error bound criterion**: 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, called the *error parameter*. If the error is not guaranteed, our motivation study in Section 5.3.2 shows that the cost of cable will differ by USD $1 billion per 0.8km path length difference.

There are three categories of existing algorithms for solving the weighted region problem approximately: (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 [33] exploits Snell's law and continuous Dijkstra algorithm to calculate the weighted shortest path that follows Snell's law. But, its running time is $O(n^8 \log c_1)$, which is very large, where n is the number of vertices of the terrain and c_1 is a constant depends on the error and some geometry information of T. Secondly, the *SP* approach [9, 23, 28] places discrete points (i.e., Steiner points) on edges in E, and then constructs 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 violates Snell's law criterion of a good algorithm for solving the weighted region problem. Thirdly, the *SPSL* approach first uses the *SP* approach to find a weighted path which may not follow Snell's law and retrieve an edge sequence S that this path passes based on T, and then exploits Snell's law to calculate the weighted shortest path which follows Snell's law based on the edge sequence S. It is the best one among these three methods because it could obtain a reasonable running time and also guarantee that the calculated the weighted shortest path follow Snell's law. But, the best-known existing work [35] violates the error bound 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 these, we propose a two-step algorithm for calculating the weighted shortest path in the 3D weighted region problem using algorithm *Divide-and-conquer step plus Logarithmic scheme Steiner*

Point placement (*DLSP*) and algorithm *Effective Weight pruning step 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 categorize our algorithm as the third approach as mentioned in Section 1.2 (i.e., the *SPSL* approach).

Compared with the best-known existing work [35], (1) we use a different but efficient Steiner point placement scheme in the first step, and an efficient pruning out technique in the second step, for reducing the algorithm's running time and memory usage, (2) we also use divide-and-conquer step to handle some cases that the output of the first step is not applicable for the input in the second step, but [35] does not consider these cases, and (3) we also provide detailed theoretical analysis on the error bound of our algorithm, but [35] does not provide error bound on both two steps. We then summarize our major **contributions**.

Firstly, to the best of our knowledge, we are the first to propose the efficient algorithm *DLSP-EWSL* for calculating the weighted shortest path in weighted region problem which fulfills the two criteria of a good algorithm in solving the weighted region problem.

Secondly, we provide thorough theoretical analysis on the running time, memory usage, and error bound of our algorithm.

Thirdly, our algorithm perform much better than the best-known existing work [35] in terms of running time and memory usage, and our algorithm is suitable for real-life applications that require real-time responses. Our experimental results show that algorithm *DLSP* and algorithm *EWSL* runs up to 500 times and 20% faster than the best-known algorithm [35] on benchmark real datasets with the same error ratio, respectively. In addition, for a terrain 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 [35] total query time is 119,000s ($\approx$ 1.5 day) and total memory usage is 2.9GB. For a real-time map application in our user study in Section 5.3.1, our algorithm just needs 0.38s to calculate the result.

The remainder of the paper is organized as follows. Section 2 provides the preliminary. Section 3 presents our algorithm. Section 4 shows the related work and baseline algorithms. Section 5 presents the experimental results and Section 6 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., $n = |V|$). Each vertex $v \in V$ has three coordinate values, denoted by x_v, y_v and z_v. If two faces share a common edge, they are said to be adjacent. Each face $f_i \in F$ is assigned a weight w_i, which is a positive real number, and the weight of an edge is equal to the smaller weight of the face that contains that edge. Given a face f_i, and two points p and q on f_i, we define $d(p, q)$ to be the Euclidean distance between 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 optimal weighted shortest path $\Pi^*(s, t) = (s, r_1, \ldots, r_l, t)$, 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 $r_0 = s$, $r_{l+1} = t$, each r_i for $i \in \{1, \ldots, l\}$ is named as a *intersection point* in $\Pi^*(s, t)$, and it is a point on an edge in E. The blue line in Figure 3 shows an example of $\Pi^*(s, t)$ on a terrain surface. We define $|\cdot|$ to be the weighted distance of a path (e.g., $|\Pi^*(s, t)|$ is the weighted distance of $\Pi^*(s, t)$). Given a face f_i, and two points p and q on f_i, we define $\overline{pq}$ to be a line on f_i.

Let S^* be a sequence of edges that $\Pi^*(s, t)$ connects from s to t in order based on T, and S^* is said to be *passed by* $\Pi^*(s, t)$. Let $\Pi(s, t)$ be the final calculated weighted shortest path of our algorithm, and ϵ be a user-defined error parameter, where $\epsilon > 0$. Our algorithm guarantee that $|\Pi(s, t)| \leq (1 + \epsilon)|\Pi^*(s, t)|$ for any s and t in V. A notation table could be found in the appendix of Table 3.

2.2 Snell's law

Let $S = ((v_1, v_1'), \ldots, (v_l, v_l')) = (e_1, \ldots, e_l)$ be a sequence of edges based on T. Given two points s and t in V, we define $\Pi^*(s, t|S) = (s, \psi_1, \ldots, \psi_l, t)$ to be the optimal weighted shortest path between s and t that passes the edge sequence S, where each ψ_i for $i \in \{1, \ldots, l\}$ is an intersection point in $\Pi^*(s, t|S)$, and it is a point on an edge in E. According to Lemma 3.6 in [33], $\Pi^*(s, t|S)$ is the unique weighted shortest path on S, and it obeys Snell's law at each $\psi_1, \ldots, \psi_l$. Let $F(S) = (f_0, f_1, \ldots, f_{l-1}, f_l)$ be a sequence of adjacent faces with respect to S such that for every f_i with $i \in \{1, \ldots l-1\}$, 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 $W(S) = (w_0, w_1, \ldots, w_{l-1}, w_l)$ be a weight list with respect to $F(S)$ such that for every w_i with $i \in \{0, \ldots l\}$, w_i is the face weight of f_i in $F(S)$. Let $\overline{n(f_i, e_i, \psi_i)}$ be the normal on face f_i that perpendicular to edge e_i at ψ_i with $i \in \{1, \ldots l\}$. Given four lines $\overline{n(f_i, e_i, \psi_i)}$, $\overline{\psi_i \psi_{i+1}}$, $\overline{n(f_i, e_i, \psi_i)}$ and $\overline{\psi_i \psi_{i-1}}$, 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 3, we have an edge sequence $S = (e_1, \ldots, e_l)$ with the corresponding face sequence $F(S) = (f_0, \ldots, f_{i-1}, f_i, \ldots, f_{l-1}, f_l)$. We assume that $S = S^*$, where S^* is the edge sequence that $\Pi^*(s, t)$ passes based on T. Thus, $\Pi^*(s, t)$ is exactly the same as $\Pi^*(s, t|S)$, and $r_i = \psi_i$ for $i \in \{1, \ldots, l\}$. 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. 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 $i \in \{1, \ldots, l\}$.*

After applying Snell's law, if $\alpha_i = 90^\circ$ at ψ_i on $\Pi^*(s, t|S)$, ψ_i is called as a *critical point* (r_l in Figure 3 is a critical point). It will happens if $\overline{\psi_i \psi_{i-1}}$ comes to ψ_i with $\sin \beta_i = \frac{w_i}{w_{i-1}}$ and $w_i < w_{i-1}$, and $\beta_i = \sin^{-1} \frac{w_i}{w_{i-1}}$ is defined as the *critical angle* of e_i (β_l in Figure 3 is a critical angle of e_l).

2.3 Challenges

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

Firstly, solving the 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 [14]. But, in the 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, and then passing c, and finally go through t, and pick the shortest one in one step without knowing S [33]. However, according to [33], 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. But, 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 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 a weighted shortest path. But, this path may not follow Snell's law, so we use algorithm *EWSL* to calculate the weighted shortest path that follows Snell's law based on the edge sequence S that the path calculated in the first step passes. The calculated path of algorithm *DLSP* is said to be the *candidate weighted shortest path* since this path may not follow Snell's law, and the calculated path in algorithm *EWSL* is said to be the *SL-weighted shortest path* since this path follows Snell's Law. Furthermore, given an edge sequence $S = ((v_1, v_1'), \ldots, (v_l, v_l')) = (e_1, \ldots, e_l)$, S is said to be a *full edge sequence* if the length of each edge in S is larger than 0. Given path $(s, \sigma_1, \sigma_2, \ldots, \sigma_8, \sigma_9, t)$ in Figure 4 (a), the edge sequence $S_a = ((a, b), (b, c), (c, d), (c, e), (e, f), (e, g), (g, h), (h, i), (i, j))$ passed by this path is an example of a full edge sequence since the length of each edge in S_a is larger than 0. Similarly, given an edge sequence S, S is said to be a *non-full edge sequence* if there exists at least one edge whose length is 0. Given path $(s, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, t)$ in Figure 4 (a), the edge sequence $S_b = ((a, b), (c, c), (e, e), (h, h), (i, j))$ passed by this path is an example of a non-full edge sequence since the edge length at (c, c), (e, e) and (h, h) are 0.

Algorithm *DLSP*: In algorithm *DLSP*, given a terrain surface T, we aim to find a full edge sequence that the candidate weighted shortest path will pass based on T. We first use an efficient Steiner point placement scheme (by providing a lower bound of the minimum distance between two adjacent Steiner points on the same edge) to place Steiner points on each edge in E (which needs a small number of Steiner points per edge), then apply Dijkstra algorithm [18] on these Steiner points and the original vertices to calculate the candidate weighted shortest path, and retrieve an edge sequence S that this calculated path passes based on T. Figure 5 (b) shows an example of the placement of Steiner points in our algorithm. We will not place Steiner points inside three polygonal caps, and the distance from Steiner point on the polygonal caps to the nearest polygonal cap center forms a lower bound of the minimum distance between two adjacent Steiner points on the same edge. But, directly using our efficient Steiner point placement scheme in the edge sequence S calculation will make the calculated S to be a non-full edge sequence, and we cannot use Snell's law on this S to find the *SL*-weighted shortest path using our second step (we will discuss the reason in Section 3.1). So, we propose an efficient divide-and-conquer step to modify this path to convert its corresponding edge sequence from a non-full edge sequence to a full edge sequence.

Thus, in algorithm *DLSP*, we first use an efficient Steiner point placement scheme to place Steiner points, construct weighted graph and apply Dijkstra algorithm to calculate the initial candidate weighted shortest path (i.e., **initial path calculation**), and then modify this path to convert its corresponding edge sequence from a non-full edge sequence to a full edge sequence by using divide-and-conquer step (i.e., **full edge sequence conversion**). Specifically, in full edge sequence conversion step, along the calculated path from a source point to a 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 divide the whole path into a smaller path segment that only contains v_i (resp. $v_i, \ldots, v_j$). Next, we use algorithm *DLSP* 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.

In Figure 4 (a), $(s, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, t)$ is the path result of algorithm *DLSP* initial path calculation step and $(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5)$ is the smaller segment (i.e., the orange line) since the edge sequence $((\phi_2 = c, c), (\phi_3 = e, e), (\phi_4 = h, h))$ passed by this path has edge length equal to 0 at (c, c), (e, e) and (h, h). In algorithm *DLSP* full edge sequence conversion step, we add more Steiner points and use algorithm *DLSP* again to find a new path segment $(\sigma_1, \sigma_2, \ldots, \sigma_8, \sigma_9)$ such that the edge sequence $((b, c), (c, d), (c, e), (e, f), (e, g), (g, h), (h, i))$ 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*.

In contract, the Steiner point placement scheme used in the first step of the best-known existing work [35] (which belongs to the *SPSL* approach), called algorithm *Fixed scheme Steiner Point placement* (*FSP*) [23, 28], will result in a a large number of Steiner points per edge, and make their algorithm very slow, even though the edge sequence retrieved from algorithm *FSP* is not likely to be a non-full edge sequence (we will discuss the reason in Section 3.1). Because algorithm *FSP* just places m_f evenly distributed Steiner points on every edge e_i in E based on the length of e_i, it does not consider any lower bound of the minimum distance between two adjacent Steiner points on the same edge when placing Steiner points near the vertices of faces, so it needs a large number of Steiner points per edge. Figure 6 shows an example of the placement of Steiner points in algorithm *FSP* (with the same error ratio as of Figure 5 (b)). Intuitively, algorithm *FSP* needs $O(n^2)$ Steiner points per edge to bound the error, but our Steiner point placement scheme only needs $O(n \log c_2)$ Steiner points per edge, where c_2 is a constant depends on the error and some geometry information of T. Our experiment show that our Steiner point placement scheme

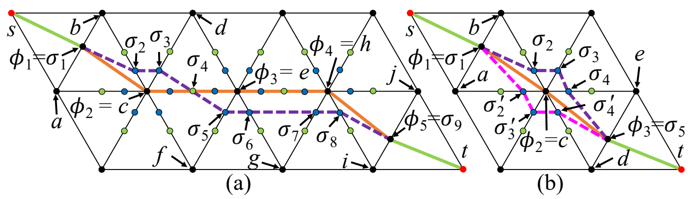

Figure 4: An example of (a) successive endpoint case, and (b) single endpoint case in algorithm *DLSP* full edge sequence conversion step

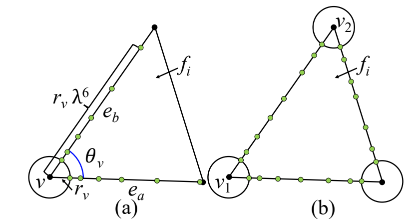

Figure 5: An example of Steiner points on (a) e_a and e_b, and (b) f_i in algorithm *DLSP* initial path calculation step

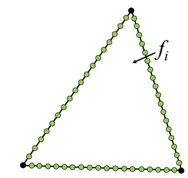

Figure 6: An example of Steiner points on f_i in algorithm *FSP*

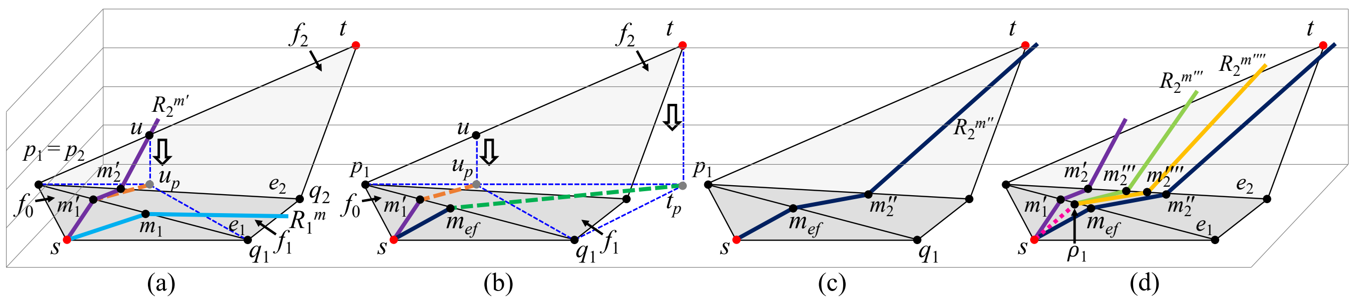

Figure 7: An example of algorithm *EWSL* (a) with initial ray for calculating effective weight on the effective face $\triangle u_p p_1 q_1$, (b) for calculating m_{ef} using the weight of f_0 and the effective weight of $\triangle u_p p_1 q_1$ (or $\triangle t_p p_1 q_1$), (c) with final ray passing through m_{ef}, and (d) processing on the remaining edges

could save 118,000s ($\approx$ 1.4 day) compared with algorithm *FSP* for a terrain with 50k faces. In addition, the first step of [35] does not consider whether their calculated edge sequence is applicable to be used in Snell's law or not, and they do not have the divide-and-conquer conversion step. Furthermore, [35] does not provide any error bound for their first step, but we provide an error bound in terms of ϵ for algorithm *DLSP*.

Algorithm *EWSL*: In algorithm *EWSL*, given the edge sequence S calculated by algorithm *DLSP*, we aim to find the weighted shortest path that follows Snell's law on S. The basic idea is to use binary search to find the optimal position of the intersection points of the weighted shortest path that follows Snell's law and each edge in S, then connect these intersection points to form the result path. But, we introduce an efficient pruning step, such that by considering one useful information in T, called *effective weight*, we could reduce the total number of iterations in the binary search, and reduce the algorithm's running time and memory usage.

Thus, in algorithm *EWSL*, we first use binary search to find the initial path on S that follows Snell's law (i.e., **binary search initial path finding**), and then use effective weight to prune out unnecessary checking in binary search to speed up the algorithm (i.e., **effective weight pruning**), next use binary search to find the *SL*-weighted shortest path with error guarantee on S that follows Snell's law (i.e., **binary search *SL*-weighted shortest path finding**). Specifically, in binary search initial path finding step, for the first edge e_1 in S that opposite to s, we select 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 intersection point between the ray itself and each edge in S. Next, we check 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). We iterate this procedure once the light ray passes the whole S, then we are ready to use effective weight prune out unnecessary checking. In effective weight pruning step, we regard all the faces except the first face in $F(S)$ as one *effective face*, 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. In binary search *SL*-weighted shortest path finding step, we iterate the binary search initial path finding step 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 (used for controlling the error, and proportional depending on ϵ). After the processing on e_1, we continue on other edges in S until all the edges in S has been processed.

In Figure 7 (a), the ray that starts from s and passes m_1 (i.e., the purple line) passes the S for the first time during the binary search initial path finding step, so we can use effective weight pruning step. We regard f_1 and f_2 as one effective face $\triangle u_p p_1 q_1$, and the ray in the purple line for calculating effective weight for $\triangle u_p p_1 q_1$. In Figure 7 (b), we calculate the position of effective intersection point 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 7 (c), we find the ray starts from s and passes m_{ef} (i.e., the dark blue line), which is very close to t, implies that m_{ef} is very close to the optimal position. In Figure 7 (d), we enter binary search *SL*-weighted shortest path finding step, the ray starts from ρ_1 and passes m_2''' (resp. m_2'''') (i.e., the green (resp. yellow) line) are the iterations of binary search initial path finding step on other edges.

In contract, the second step of the best-known existing work [35] does not have this pruning step, so their running time is very large. In our experiment, for a terrain with 5M faces, when calculating *SL*-weighted shortest path, using effective weight pruning step could save 20,000s ($\approx$ 5 hours) compared with purely using binary search. Furthermore, [35] does not provide any error bound for their second step, but we provide an error bound in terms of ϵ for algorithm *EWSL*.

3.1 Edge Sequence Finding by Algorithm *DLSP*

In algorithm *DLSP*, given a 3D terrain surface T, we aim to find a full edge sequence that the candidate weighted shortest path will pass based on T. There are two steps involved:

- **Initial path calculation**: Placing Steiner points, and constructing a weighted graph, and then applying Dijkstra algorithm to calculate the initial candidate weighted shortest path (which corresponding edge sequence may be a non-full edge sequence).
- **Full edge sequence conversion**: Modifying the previous calculated candidate weighted shortest path to convert its corresponding edge sequence from a non-full edge sequence to a full edge sequence by using divide-and-conquer step.

We give some notations first. Given two points s and t in V, we define $\Pi'_{DLSP}(s,t) = (s, \phi_1, \ldots, \phi_l, t)$ to be the calculated candidate weighted shortest path (which corresponding edge sequence may be a non-full edge sequence) between s and t using algorithm *DLSP* initial path calculation step, where each ϕ_i for $i \in \{1, \ldots, l\}$ is an intersection point in $\Pi'_{DLSP}(s,t)$, and it is a point on an edge (including two endpoints of the edge) in E. We further define $\Pi_{DLSP}(s,t) = (s, \sigma_1, \ldots, \sigma_l, t)$ to be the calculated candidate 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*, where each σ_i for $i \in \{1, \ldots, l\}$ is an intersection point in $\Pi_{DLSP}(s,t)$, and it is a point on an edge (including two endpoints of the edge) in E. The path $(s, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, t)$ in Figure 4 (a) (resp. path $(s, \phi_1, \phi_2, \phi_3, t)$ in Figure 4 (b)) is an example of $\Pi'_{DLSP}(s,t)$ that the corresponding edge sequence is non-full edge sequences since the edge length at $(\phi_2 = c, c)$, $(\phi_3 = e, e)$ and $(\phi_4 = h, h)$ (resp. $(\phi_2 = c, c)$) is 0. The path $(s, \sigma_1, \sigma_2, \ldots, \sigma_8, \sigma_9, t)$ in Figure 4 (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 4 (b) are two examples of $\Pi_{DLSP}(s,t)$ since the corresponding edge sequence has been converted to a full edge sequence.

We define ϵ_{SP} to be the error parameter for algorithm *DLSP* initial path calculation step, where $\epsilon_{SP} > 0$. For ease of analysis [9], we define ϵ'_{SP} to be the another error parameter for algorithm *DLSP*, 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. 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. 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 $0 < \epsilon'_{SP} < \frac{1}{2}$. Let θ_v to be the angle (measured in 3D) between any two edges of T that are incident to v. Let h, r and θ be the minimum h_v, r_v and θ_v for all $v \in V$, respectively. 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 *DLSP* as follows.

In the first step, we calculate the initial candidate weighted shortest path (which corresponding edge sequence may be a non-full edge sequence) 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 $a_1, a_2, \ldots, a_{\tau_a - 1}$ (resp. $b_1, b_2, \ldots b_{\tau_b - 1}$) along e_a (resp. e_b) such that $|\overline{va_j}| = r_v \lambda^{j-1}$ (resp. $|\overline{vb_k}| = r_v \lambda^{k-1}$) where $\tau_a = \log_\lambda \frac{|e_a|}{r_v}$ (resp. $\tau_b = \log_\lambda \frac{|e_b|}{r_v}$) for every integer $2 \leq j \leq \tau_a - 1$ (resp. $2 \leq k \leq \tau_b - 1$). We repeat it on each edge of f_i. Figure 5 (a) and Figure 5 (b) show an example of Steiner points on e_a and e_b, and f_i, respectively, in algorithm *DLSP* inititial path calculation step. After that, we construct a weighted graph using the newly added Steiner points together with the original vertices, and then applying Dijkstra algorithm to calculate the candidate weighted shortest path $\Pi'_{DLSP}(s,t)$ between s and t.

Note that in this step, 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). So the Steiner points that $\Pi'_{DLSP}(s,t)$ pass will either (1) be far away from the original vertices in V, or (2) be the original vertices in V exactly. For the latter case, the retrieved edge sequence from $\Pi'_{DLSP}(s,t)$ will contain some vertices, and this edge sequence will be a non-full edge sequence. As discussed before, when a path that follows Snell's law hits a vertex, we no longer have complete information about where it goes next since it can pass through a vertex in many possible ways. In this case, we cannot use Snell's law on this S to find the *SL*-weighted shortest path. So we need the full edge sequence conversion step to convert the corresponding edge sequence of $\Pi'_{DLSP}(s,t)$ to a full edge sequence, then we can use this full edge sequence in algorithm *EWSL* to find the *SL*-weighted shortest path that follows Snell's law. Note that algorithm *DLSP* full edge sequence conversion step is a refinement step of initial path calculation step, so the whole error parameter for algorithm *DLSP* is the same as the error parameter for algorithm *DLSP* full edge sequence conversion step, i.e., ϵ_{SP}. But in algorithm *FSP*, there is no lower bound on the minimum distance between adjacent Steiner points on the same edge, so there are more Steiner points placed near the original vertices in V, and the edge sequence retrieved from algorithm *FSP* is not likely to be a non-full edge sequence.

In the second step, we modify $\Pi'_{DLSP}(s,t)$ in order to convert its corresponding edge sequence from a non-full edge sequence to a full edge sequence. Specifically, we check the path $\Pi'_{DLSP}(s,t)$ 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 $\Pi'_{DLSP}(s,t)$ 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 $\Pi'_{DLSP}(s,t)$ lies in the internal of an edge in E, v is on the original vertex in V means that the corresponding point in

$\Pi'_{DLSP}(s,t)$ 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 4 (a) or 4 (b)).
- If v_c is on the original vertex in V, there are two more cases:
  - **Successive endpoint** (refer to Figure 4 (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 $\Pi'_{DLSP}(s,t)$ 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, 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 $v_p = \phi_3$), it 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* full edge sequence conversion step 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 $\Pi'_{DLSP}(v_s,v_e) = (v_s = \phi_1, \phi_2, \phi_3, \phi_4, \phi_5 = v_e)$ (i.e., the orange line) as $\Pi_{DLSP}(v_s,v_e)$ if $|\Pi_{DLSP}(v_s,v_e)| < |\Pi'_{DLSP}(v_s,v_e)|$.
  - **Single endpoint** (refer to Figure 4 (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 $\Pi'_{DLSP}(v_p,v_n) = (v_p = \phi_1, \phi_2, \phi_3 = v_n)$ (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 $\Pi'_{DLSP}(v_p,v_n)$, $\Pi_l(v_p,v_n)$, and $\Pi_l(v_p,v_n)$, and substitute $\Pi'_{DLSP}(v_p,v_n)$ 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 $\Pi'_{DLSP}(v_p,v_n)$ is still the longest path, where ζ is a constant and normally is set as 10.

Then, we move forward in $\Pi'_{DLSP}(s,t)$ 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 $\Pi'_{DLSP}(s,t)$. After we process all the vertices in $\Pi'_{DLSP}(s,t)$, we return the result path as $\Pi_{DLSP}(s,t)$ and retrieve the edge sequence S that $\Pi_{DLSP}(s,t)$ passes based on T.

Theoretical analysis of algorithm *DLSP*: The running time, memory usage and error of *DLSP* are in Theorem 3.1.

Theorem 3.1. *The running time for algorithm DLSP is $O(\mu_1 n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}) + \zeta n)$, and the memory usage is $O(n \log \frac{LN}{\epsilon_{SP}})$, where μ_1 is a data-dependent variable. In the average case, μ_1 is $O(1)$. In the worst case, μ_1 is $O(n)$. In our experiment, $\mu_1 \in [1,2]$. This algorithm guarantees that $|\Pi_{DLSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)|$.*

Proof Sketch. We first show that the running time for algorithm *DLSP* initial path calculation step is $O(n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}))$ and the memory usage is $O(n \log \frac{LN}{\epsilon_{SP}})$ (because we could show that the number of Steiner points per edge k_{SP} is $O(\log \frac{LN}{\epsilon_{SP}})$, and we get the running time and memory usage using Dijkstra algorithm). This step guarantees that $|\Pi'_{DLSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)|$ (a proof sketch could be found in Theorem 1 of [9] and a detailed proof could be found in Theorem 3.1 of [27]).

We then show that in algorithm *DLSP* full edge sequence conversion step: (1) For the average case running time, we have $\frac{1}{3}$ of $\Pi'_{DLSP}(s,t)$ passes on the edge, $\frac{1}{3}$ of $\Pi'_{DLSP}(s,t)$ belongs to single endpoint case, and the remaining $\frac{1}{3}$ of $\Pi'_{DLSP}(s,t)$ belongs to successive endpoint case. (2) For the worst case running time, we have all the points in $\Pi'_{DLSP}(s,t)$ passes the original vertices in V. (3) For the memory usage, we consider the memory for Dijkstra algorithm, handling one single endpoint case and handling successive endpoint case. (4) For the error bound, we only use the refinement path $\Pi_{DLSP}(s,t)$ if its weighted distance is shorter than $\Pi'_{DLSP}(s,t)$. For the sake of space, all the detailed proof in this paper could be found in the appendix. □

3.2 Edge Sequence Based Weighted Shortest Path Finding by Algorithm *EWSL*

In algorithm *EWSL*, given the edge sequence S that $\Pi_{DLSP}(s,t)$ passes based on T, we aim to find the *SL*-weighted shortest path that follows Snell's law on S. There are three steps involved:

- **Binary search initial path finding**: Using binary search to find the initial path on S that follows Snell's law.
- **Effective weight pruning**: Using effective weight to prune out unnecessary checking in binary search to speed up the algorithm.
- **Binary search *SL*-weighted shortest path finding**: Using binary search to find the *SL*-weighted shortest path with error guarantee on S that follows Snell's law.

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 *SL*-weighted shortest path between s and t using algorithm *EWSL* on the edge sequence S, where each ρ_i for $i \in \{1,\ldots,l\}$ is an intersection point in $\Pi_{SL}(s,t|S)$, and it is a point on an edge in E. Let ϵ_{SL} to be the error parameter for algorithm *EWSL*, where $\epsilon_{SL} > 0$. In Figure 2, $\Pi_{DLSP}(s,t)$ 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 $\Pi_{SL}(s,t|S)$. 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 3, we have $\Pi^*(s,t) = \Pi^*(s,r_i) \oplus \Pi^*(r_i,t)$.

Following Snell's law, when given an edge sequence S, and a point c_1 on $e_1 \in S$, we can apply 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, and then we can continue the calculation to obtain the path $\Pi_c = (s, c_1, c_2, \ldots, c_g, R^c_g)$, where $1 \leq g \leq l$, each c_i for $i \in \{1, \ldots, g\}$ is an intersection point in Π_c, and R^c_g is the last out-ray of the path at $e_g \in S$. This calculation will stop when (1) R^c_g intersects $e_g \in S$ with $g = l$, or (2) R^c_g 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's 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 (a) shows an example of $\Pi_m = (s, m_1, R^m_1)$ (i.e., the blue line) that does not pass the whole $S = (e_1, e_2)$ because R^m_1 intersects e_1 but does not intersect e_2, and $\Pi_{m'} = (s, m'_1, m'_2, R^{m'}_2)$ (i.e., the purple line) that passes the whole S because $R^{m'}_2$ intersects e_2. Therefore, if we could find the point ρ_1 on e_1 such that after we calculate the 3D surface Snell's ray $\Pi_\rho = (s, \rho_1, \ldots, \rho_l, R^\rho_l)$, t is on the last ray R^ρ_l, then Π_ρ will be the result of $\Pi_{SL}(s, t|S)$.

We then introduce the three steps in algorithm *EWSL* as follows.

In the first step, we use binary search to find the initial path on S that follows Snell's law (refer to Figure 7 (a)). For $i = 1$, 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's ray $\Pi_m = (s, m_1, \ldots, m_g, R^m_g)$ from s and passing S based on T 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 $\Pi_m = (s, m_1, R^m_1)$ when $i = 1$). Depending on whether Π_m leave the edge sequence S or not, there are two cases:

- Π_m does not pass the whole S based on T, i.e., R^m_g intersects $e_g \in S$ but does not intersect $e_{g+1} \in S$ with $g < l$ (e.g., the blue line $\Pi_m = (s, m_1, R^m_1)$ when $i = 1$ does not pass the whole S): If e_{g+1} is on the left (resp. right) side of R^m_g, then we need to search in $[a_i, m_i]$ (resp. $[m_i, b_i]$), so we set $b_i = m_i$ (resp. $a_i = m_i$) for $i \in \{1, \ldots, g\}$. (E.g., e_2 is on the left side of the blue line R^m_1, we set $b_1 = m_1$, and we have $[a_1, b_1] = [p_1, m_1]$.) We iterate this step until Π_m passes the whole S based on T for the first time.
- Π_m passes the whole S based on T, i.e., R^m_g intersects $e_g \in S$ with $g = l$ (e.g., the purple line $\Pi_{m'} = (s, m'_1, m'_2, R^{m'}_2)$ when $i = 1$ passes the whole S): If t is on R^m_g, then we could just return Π_m as the result. If t is on the left (resp. right) side of R^m_g, then we need to search in $[a_i, m_i]$ (resp. $[m_i, b_i]$), so we set $b_i = m_i$ (resp. $a_i = m_i$) for $i \in \{1, \ldots, l\}$. (E.g., t is on the right side of the purple line $R^{m'}_2$, we set $a_1 = m'_1$ and $a_2 = m'_2$, and we have $[a_1, b_1] = [m'_1, m_1]$ and $[a_2, b_2] = [m'_2, q_2]$.)

In the second step, we use effective weight to prune out unnecessary checking in binary search to speed up the algorithm (refer to Figure 7 (a)-(c)). For $i = 1$, suppose that we have found a 3D surface Snell's ray $\Pi_m = (s, m_1, \ldots, m_l, R^m_l)$ (e.g., the purple line $\Pi_{m'} = (s, m'_1, m'_2, R^{m'}_2)$) from s with the initial ray through m'_1 that Π_m passes the whole S based on T for the first time.

- Firstly (refer to Figure 7 (a)), given two edges that adjacent to t in the last face f_l in $F(S)$, we calculate the intersection point between R^m_l and these two edges (either one of these two edges), and denote it as u (e.g., the purple line $R^{m'}_2$ intersect with the left edge $\overline{p_1 t}$ of f_2 that adjacent to t).
- Secondly (refer to Figure 7 (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 7 (a)), by using $\overline{sm'_1}$ (e.g., the purple 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 7 (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 Snell's law again to find the effective intersection 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 Snell's law in vector form [8], we could build a quartic equation using w_0, w_{ef}, $\overline{sm_{ef}}$ (e.g., the dark blue 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 [7] to solve m_{ef}.
- Fifthly (refer to Figure 7 (c)), we compute the 3D surface Snell's ray Π_m from s with the initial ray through m_{ef} (e.g., the dark blue line $\Pi_{m''} = (s, m_{ef}, m''_2, R^{m''}_2)$), and update $[a_i, b_i]$ for $i \in \{1, \ldots, l\}$ depending on whether Π_m passes the whole S based on T and whether t (or e_{g+1}) is on the left or right side of R^m_l (or R^m_g) for $i \in \{1, \ldots, l\}$ (or for $i \in \{1, \ldots, g\}$) (e.g., the dark blue line $\Pi_{m''}$ passes the whole S and t is on the left side of the dark blue line $R^{m'}_2$, we set $b_1 = m_{ef}$ and $b_2 = m''_2$, and we have $[a_1, b_1] = [m'_1, m_{ef}]$ and $[a_2, b_2] = [m'_2, m''_2]$).

In the third step, we use binary search to find the *SL*-weighted shortest path with error guarantee on S that follows Snell's law (refer to Figure 7 (d)). We perform algorithm *EWSL* binary search initial path finding step until $|a_i b_i| < \delta$ (e.g., $|a_1 b_1| = |m'_1 m_{ef}| < \delta$ when $i = 1$), where $\delta = \frac{h \epsilon_{SL} w}{6lW}$ is an error parameter proportional depending on ϵ_{SL}, 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 $[a_1, b_1] = [m'_1, m_{ef}]$ 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 based on T by starting at m_{i+1} on e_{i+1}, and to m_g on e_g (e.g., we have the green line $\Pi_{m'''} = (\rho_1, m'''_2, R^{m'''}_2)$ and the yellow line $\Pi_{m''''} = (\rho_1, m''''_2, R^{m''''}_2)$ when $i = 2$). We iterate this step until we process all the edges in S (e.g., until we process all the edges in $S = (e_1, e_2)$, we get result path $\Pi_{SL}(s, t|S) = (s, \rho_1, \rho_2, t)$).

Theoretical analysis of algorithm *EWSL*: The running time, memory usage and error of *EWSL* are in Theorem 3.2.

Theorem 3.2. *The running time for algorithm EWSL is $O(\mu_2 n^2)$, and the memory usage is $O(n^2)$, where μ_2 is a data-dependent variable. In the best case, μ_2 is $O(1)$. In the worst case, μ_2 is $O(n^2 \log(\frac{nNW}{w\epsilon}))$. In our experiment, $\mu_2 \in [1, 105]$. This algorithm guarantees that $|\Pi_{SL}(s, t|S)| \leq (1 + \epsilon_{SL})|\Pi^*(s, t|S)|$.*

Proof Sketch. For the best case running time, the algorithm *EWSL* effective weight pruning step could find the intersection point's optimal position on the first edge in S in $O(1)$ time (note that there are at most n^2 edges in S). For the worst case running time, the algorithm *EWSL* binary search initial path and *SL*-weighted shortest path finding step use binary search on each edge in $O(n^2 \log \frac{nWL}{h\epsilon_{SL} w})$ time (there are at most n^2 edges in S). For the memory usage, since there are at most n^2 edges in S, we obtain the result. For the error bound, for $i \in \{0, 1, 2, \ldots, l\}$, we use induction to prove $|\Pi_{SL}(\rho_i, t|S)| \leq (1+\frac{\epsilon}{2})|\Pi^*(\rho_i, t|S)|+3(l-i)\delta W$, and then by setting $k = 0$ and since we set $\delta = \frac{h\epsilon_{SL} w}{6lW}$, we finish the proof. □

3.3 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 candidate weighted shortest path using algorithm *DLSP* is $\Pi_{DLSP}(s,t)$, the calculated *SL*-weighted shortest path using algorithm *EWSL* is $\Pi_{SL}(s,t|S)$, the final calculated *SL*-weighted shortest path 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 based on T, 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*: The running time, memory usage, and error of *DLSP-EWSL* are in Theorem 3.3.

Theorem 3.3. *The total running time for algorithm DLSP-EWSL is $O(\mu_1 n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + \zeta n + \mu_2 n^2)$, 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 3.1 and Theorem 3.2 to prove. 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 $\Pi_{DLSP}(s,t)$ or $\Pi_{SL}(s,t|S)$ is longer, there are four cases. Then we could use Theorem 3.1 and Theorem 3.2 to prove. □

4 RELATED WORK & BASELINE

4.1 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 [33] algorithm. The algorithm will return a $(1+\epsilon)$-approximation weighted shortest distance in $O(n^8 \log(\frac{nNW}{w\epsilon}))$ time. But, the running time of the *WPSL* approach is very large. To the best of our knowledge, there is no implementation of the *WPSL* approach so far [27].

(2) For the *SP* approach, it uses Dijkstra algorithm on the weighted graph constructed using Steiner points on edges in E and the original vertices to calculate the weighted shortest path which may not follow Snell's law. Some different Steiner point placement schemes, e.g., algorithm *FSP* [23, 28] and algorithm *Logarithmic scheme Steiner Point placement* (*LSP*) [9], run in $O(n^3 \log n)$ and $O(n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$ time, respectively. The Steiner point placement scheme used in *LSP* is similar to *DLSP*, but the divide-and-conquer step in *DLSP* could convert the candidate weighted shortest path's corresponding edge sequence calculated using *LSP* from a non-full edge sequence to a full edge sequence, such that Snell's law is applicable on the full edge sequence to calculate the *SL*-weighted shortest path. A comprehensive running time analysis of different Steiner point placement schemes could be found in [12]. But, the *SP* approach violates Snell's law criterion of a good algorithm for solving the weighted region problem.

(3) For the *SPSL* approach, it first uses the *SP* approach to find a weighted path which may not follow Snell's law, and then exploits Snell's law to refine it to be the weighted shortest path that follows Snell's law. [35] is the best-known existing work that uses this approach, and it uses algorithm *FSP* and algorithm *Binary Search Snell's Law* (*BSSL*) in these two steps, where *BSSL* is a naive version of *EWSL* without effective weight pruning out step (recall that *EWSL* could save 20,000s (≈ 5 hours) compared with *BSSL* for a terrain with 5M faces). The running time for *FSP* and *BSSL* are $O(n^3 \log n)$ and $O(n^4 \log \frac{L}{\delta})$. Even though the work [35] is regarded as the best-known algorithm among all three types of algorithms for solving the weighted region problem (based on the requirement that the weighted shortest path need follow Snell's law), it violates the error bound 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 with 50k faces, our algorithm's total query time is 534s (≈ 9 min) and total memory usage is 130MB, but the algorithm's in [35] has total query time 119,000s (≈ 1.5 day) and total memory usage 2.9GB.

4.2 Baseline

The *SP* approach proposed by [23, 28] (i.e., algorithm *FSP*) is a widely-known algorithm and a standard approach for solving the weighted region problem [36], we set it as one of our baseline algorithms. Note that the algorithm in [23] is for calculating unweighted shortest path, we adapt it by assigning each face a weight for solving the weighted region problem. The *SP* approach proposed by [9] (i.e., *LSP*) is regarded as the best-known algorithm in the *SP* approach due to its shortest running time, we also set it as one of our baseline algorithms. But, the calculated candidate weighted shortest path for *FSP* and *LSP* does not follow Snell's law. In addition, the *SPSL* approach proposed by [35] (i.e., *FSP-BSSL*) is the fastest algorithm among all three types of algorithms for solving the weighted region problem (based on the requirement that the result 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 have three baseline algorithms, i.e., *FSP*, *LSP* and *FSP-BSSL*.

We do not use the *WPSL* approach as the baseline algorithm, due to its large running time. In addition, there is no implementation of the *WPSL* approach so far, even the authors themselves do not provide an implementation of their algorithm [27].

We then compare the baseline algorithms and our algorithm 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 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*, the best case running time of *EWSL* outperforms *BSSL*, since *EWSL* uses pruning step compared with *BSSL*.

5 EMPIRICAL STUDIES

5.1 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 [23, 24, 32, 37, 38].

Datasets: Following some existing studies on terrain data [17, 32, 34], we conducted our experiment based on seven real terrain datasets, namely (1) BearHead (*BH*, with 280k faces) [4, 37, 38], (2) EaglePeak (*EP*, with 300k faces) [4, 37, 38], (3) SeaBed (*SB*, with 2k faces) [10], (4) CyberPunk (*CP*, with 2k faces) [3], (5) PathAdvisor (*PA*, with 1k faces) [41], (6) a small-version of *BH* (*BH-small*, with 3k faces), and (7) a small-version of *EP* (*EP-small*, with 3k faces). For *SB* [10] and *CP* [3] dataset, we first obtained the height / satellite map from [3, 10], then used Blender [1] to generate the 3D terrain model. For *BH-small* and *EP-small* datasets, we generate them using *BH* and *EP* following the procedure in [32, 37, 38] (which creates a terrain with different resolutions), since one of the baseline algorithms, i.e., algorithm *FSP* [23, 28], is not feasible on any of the full datasets due to its expensive running time. The procedure of generating a terrain with different resolution could be found in the appendix. Following the work [20, 35], we set the weight of a triangle in terrain datasets to be the slope of that face.

In addition, we have the two sets of datasets with different number of faces (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). We generate these two sets of datasets using *EP* and *EP-small* following same procedure for generating terrains with different resolutions.

Algorithms: Our algorithm *DLSP-EWSL*, and the baseline algorithms, i.e., *FSP* [23, 28], *LSP* [9] and *FSP-BSSL* [35], are studied in the experiments. But, the calculated candidate weighted shortest path of *FSP* and *LSP* do not follow Snell's law. In order to conduct the ablation study, 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 chose two vertices in V, one as a source and the other as a destination. For each measurement, 100 queries were answered, and the average, minimum and maximum 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 3.3.

In addition, we used five measurements to evaluate the algorithm performance, namely (1) *preprocessing time* (i.e., the time for constructing the weighted graph using Steiner points), (2a) *query time for the first 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*), (2d) *total query time* (i.e., the time for finding the weighted shortest path), (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), (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 by the algorithm compared with the exact weighted shortest path).

Since no algorithm could solve the weighted region problem exactly so far, 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, the distance error is omitted for the full datasets.

5.2 Experimental Results

There are total 36 figures. Figure 8 and Figure 9 show the result on the *EP-small* dataset when varying ϵ and ϵ_{SP}, respectively. Figure 10 (resp. Figure 11) shows the result on a set of large-version datasets (the *EP* dataset) when varying DS (resp. ϵ_{SL} with separated query time and memory usage in two steps). All the five measurements are included in Figure 8, and only the preprocessing time, query time and memory usage are included for the rest of figures for sake of space (but the separated query time and memory usage in two steps are included in Figure 11). Since the magnitude difference between the query time (resp. memory usage) on the first step and

Algorithm	Time	Memory
FSP [23, 28]	$O(\mu_1 n^3 \log n)$	$O(n^3)$
LSP [9]	$O(\mu_1 n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}))$	$O(n \log \frac{LN}{\epsilon})$
DLSP	$O(\mu_1 n \log \frac{LN}{\epsilon} \log(n \log \frac{LN}{\epsilon}) + \zeta n)$	$O(n \log \frac{LN}{\epsilon})$

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

Algorithm	Time	Memory
BSSL [35]	$O(n^4 \log(\frac{nNW}{w\epsilon}))$	$O(n^2)$
EWSL	$O(\mu_2 n^2)$	$O(n^2)$

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

Remark: μ_1 is a data-dependent variable. In the average case, μ_1 is $O(1)$. In the worst case, μ_1 is $O(n)$. In our experiment, $\mu_1 \in [1, 2]$. μ_2 is a data-dependent variable. In the best case, μ_2 is $O(1)$. In the worst case, μ_2 is $O(n^2 \log(\frac{nNW}{w\epsilon}))$. In our experiment, $\mu_2 \in [1, 105]$. 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.

the second step is very large, it seems that only one step is shown in the the experiment figures of total query time (resp. total memory usage). For the (1) total query time, (2) query time for the first step and (3) query time for the second step, the vertical bar means the minimum and maximum (1) total query time, (2) query time for the first step and (3) query time for the second step. The results on other combinations of dataset and the variation of ϵ, ϵ_{SP} and ϵ_{SL}, and the separation of two steps in query time and memory usage could be found in the appendix.

Effect of ϵ. In Figure 8, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on *EP-small* dataset by setting $\epsilon_{SP} = \epsilon_{SL} = \epsilon$. *DLSP-EWSL* superior performance of all the remaining algorithms (except *LSP*) in terms of preprocessing time, query time, memory usage and iteration count. Even though *LSP* seems to run faster than *DLSP-EWSL*, the error of *LSP* error is larger than the error of *DLSP-EWSL*, since *LSP* doesn't follow Snell's law.

Effect of ϵ_{SP}. In Figure 9, 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 *DLSP* (i.e., *DLSP-BSSL* and *DLSP-EWSL*) and *LSP* are much 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* dataset by setting ϵ_{SP} to be 0.1 as default value for all the cases with separated query time and memory usage in two steps. 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 and memory usage of *EWSL* (both *FSP-EWSL* and *DLSP-EWSL*) could always perform 4% to 20% better than *BSSL* (both *FSP-BSSL* and *DLSP-BSSL*), the minimum value of the query time of *EWSL* is also always smaller than *BSSL*.

Effect of *DS* (scalability test). In Figure 10, we tested 5 values of *DS* from {1M, 2M, 3M, 4M, 5M} on the set of large-version datasets with 5 datasets (by setting ϵ to be 0.25) for scalability test. When the dataset size is 5M, *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 with 5 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., *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., *DLSP-EWSL*) total query time is 534s ($\approx$ 9 min) and total memory usage is 130MB.

5.3 Case Study

5.3.1 User Study. We conducted a user study on a campus map weighted shortest path finding tool, which allows users to find the shortest path between any two rooms in a university campus, namely *Path Advisor* [41]. The floor of the building is represented in a terrain surface, and the dataset is *PA* dataset [41] used in our experiment, but we set the face weight in a different way. It is expected that (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 on the floor is closer to the boundary of aisle in a building (resp. the aisle center), the face is assigned with a larger (resp. smaller) weight. We obtained the code from [41] and adopted our six algorithms to their tool. We chose two places in Path Advisor as source and destination, respectively, and repeated it for 100 times to calculate the path (with $\epsilon = 0.5$). Figure 12 shows an example of different paths in Path Advisor. The blue, pink and yellow paths are the weighted shortest path that follows Snell's law (calculated using algorithm *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL*) (with distance 105.8m), the weighted shortest path that does not follow Snell's law (calculated using *FSP* and *LSP*) (with distance 106.0m), and the unweighted shortest path (with distance 98.4m), respectively. We presented Figure 12 (without text descriptions) and the path distance result to 30 users (i.e., university students), and 96.7% of users think the blue path is the most realistic one since it is not close to the obstacle and it does not have sudden direction changes. In addition, the average query time for *FSP*, *LSP*, *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL* are 16.64s, 0.28s, 17.44s, 17.43s, 0.39s and 0.38s, respectively. The result on other measurements could be found in the appendix.

5.3.2 Motivation Study. We also conducted a motivation study on the placement of undersea optical fiber cable on the seabed as mentioned in Section 1.1. The dataset is *SB* dataset [10] used in our experiment, but we set the face weight in a different way. For a face with a deeper sea level, the hydraulic pressure is higher, the cable's lifespan is reduced, and it is more expensive to repair and maintain the cable, so the face will have a larger weight. The average life expectancy of the cable is 25 years [31], and if the cable is in deep waters (e.g., 8.5km or greater), the cable needs to be repaired frequently (e.g., its life expectancy is reduced to 20 years) [31]. We randomly selected two points as source and destination, respectively, and repeated it for 100 times to calculate the path (with $\epsilon = 0.5$). Figure 13 shows an example of different paths on seabed. The green, blue and red paths are the weighted shortest path that follows Snell's law (with distance 457.9km), the weighted shortest path that does not follow Snell's law (with distance 458.7km), and the unweighted shortest path (with distance 438.3km). According to [19], the cost of undersea optical fiber cable is USD \$200M/km. Consider constructing a cable that will be used for 100 years, the total estimated cost for the green, blue and red paths are USD \$366B ($= \frac{100\text{years}}{25\text{years}} \times 457.9\text{km} \times \$200\text{M/km}$), \$367B ($= \frac{100\text{years}}{25\text{years}} \times 458.7\text{km} \times$

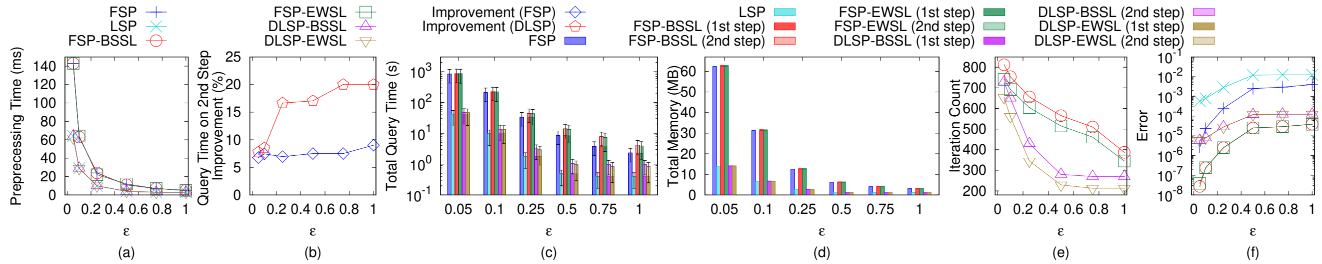

Figure 8: Effect of ϵ on *EP-small* dataset

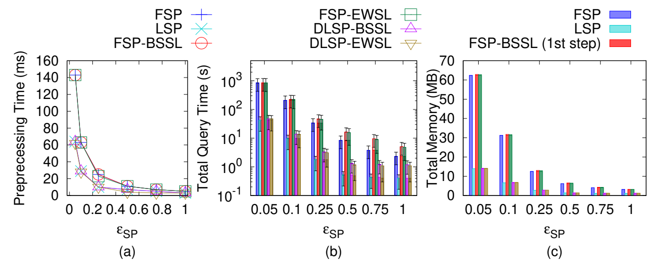

Figure 9: Effect of ϵ_{SP} on *EP-small* dataset

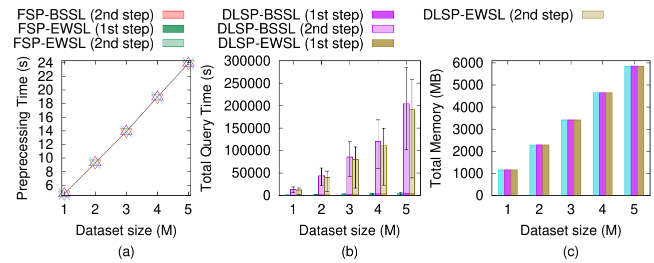

Figure 10: Effect of dataset size on a set of large-version datasets

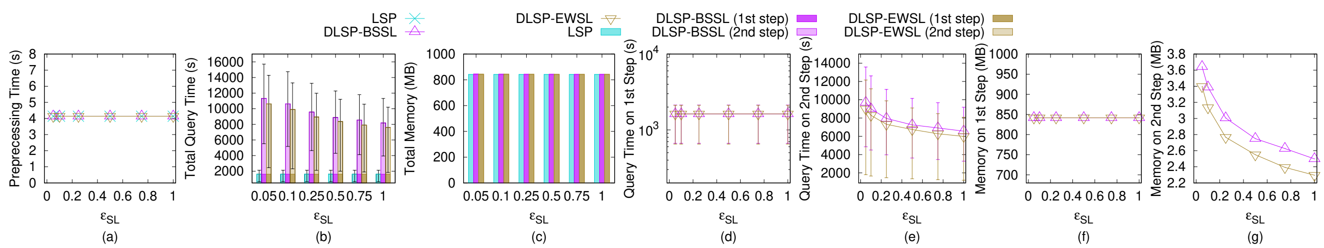

Figure 11: Effect of ϵ_{SL} on *EP* dataset and separated query time and memory usage in two steps

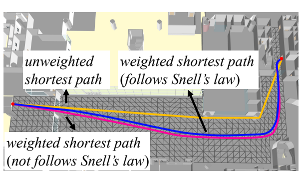

Figure 12: An example of paths in Path Advisor

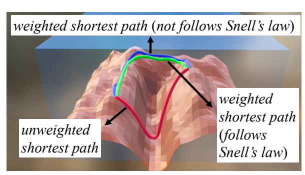

Figure 13: An example of paths on seabed

\$200M/km) and \$438B ($=\frac{100\text{years}}{20\text{years}}\times 438.3\text{km}\times\$200\text{M/km}$), respectively. The total query time for algorithm *FSP*, *LSP*, *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL* are 22.50s, 0.60s, 23.75s, 23.57s, 1.32s and 1.23s, respectively. The result on other measurements could be found in the appendix.

5.4 Experimental Results Summary

Our algorithm *DLSP-EWSL* consistently outperforms the state-of-the-art algorithm, i.e., *FSP-BSSL*, in terms of all measurements. Specifically, *DLSP* and *EWSL* 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 *DLSP-EWSL* is the best algorithm given that the path need to follow Snell's law since it is the fast algorithm that supports real-time responses. In addition, the first step dominates the second step when dataset size is small, and the second step dominates the first step when dataset size is large.

6 CONCLUSION

In our paper, we propose a two-step approximation algorithm for calculating the weighted shortest path that follows Snell's law in the 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 step 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. *Cyberpunk 2077 Map*. https://www.videogamecartography.com/2019/08/cyberpunk-2077-map.html
[4] 2022. *Data Geocomm*. http://data.geocomm.com/

[5] 2022. *Google Earth*. https://earth.google.com/web
[6] 2022. *Metaverse*. https://about.facebook.com/meta
[7] 2022. *Quartic function*. https://en.wikipedia.org/wiki/Quartic_function
[8] 2022. *Snell's law in vector form*. https://physics.stackexchange.com/questions/435512/snells-law-in-vector-form
[9] 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.
[10] Christopher Amante and Barry W Eakins. 2009. ETOPO1 arc-minute global relief model: procedures, data sources and analysis. (2009).
[11] 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.
[12] 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.
[13] Christian Bueger, Tobias Liebetrau, and Jonas Franken. 2022. Security threats to undersea communications cables and infrastructure–consequences for the EU. *Report for SEDE Committee of the European Parliament, PE702* 557 (2022).
[14] Jindong Chen and Yijie Han. 1990. Shortest Paths on a Polyhedron. In *SOCG*. New York, NY, USA, 360–369.
[15] 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.
[16] 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.
[17] 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.
[18] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.
[19] RL Gallawa. 1981. Estimated cost of a submarine fiber cable system. *Fiber & Integrated Optics* 3, 4 (1981), 299–322.
[20] Amin Gheibi, Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. 2018. Path refinement in weighted regions. *Algorithmica* 80, 12 (2018), 3766–3802.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] Jonathan Kim. 2022. *Submarine cables: the invisible fiber link enabling the Internet*. https://dgtlinfra.com/submarine-cables-fiber-link-internet/
[26] 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.
[27] Mark Lanthier. 2000. *Shortest path problems on polyhedral surfaces*. Ph.D. Dissertation. Carleton University.
[28] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica* 30, 4 (2001), 527–562.
[29] Lik-Hang Lee, Tristan Braud, Pengyuan Zhou, Lin Wang, Dianlei Xu, Zijun Lin, Abhishek Kumar, Carlos Bermejo, and Pan Hui. 2021. All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda. *arXiv preprint arXiv:2110.05352* (2021).
[30] Lik-Hang Lee, Zijun Lin, Rui Hu, Zhengya Gong, Abhishek Kumar, Tangyao Li, Sijia Li, and Pan Hui. 2021. When creators meet the metaverse: A survey on computational arts. *arXiv preprint arXiv:2111.13486* (2021).
[31] Jean-François Libert and Gary Waterworth. 2016. 13 - Cable technology. In *Undersea Fiber Communication Systems (Second Edition)*. Academic Press, 465–508.
[32] 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.
[33] 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.
[34] 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.
[35] 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.
[36] 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.
[37] 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.
[38] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, David M Mount, and Hanan Samet. 2022. Proximity queries on terrain surface. *ACM Transactions on Database Systems (TODS)* (2022).
[39] 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.
[40] 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.
[41] 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.
[42] Xiaoming Zheng, Sven Koenig, David Kempe, and Sonal Jain. 2010. Multirobot forest coverage for weighted and unweighted terrain. *IEEE Transactions on Robotics* 26, 6 (2010), 1018–1031.

A SUMMARY OF FREQUENT USED NOTATIONS

Table 3 shows a summary of frequent used notations.

B 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 4 (a), we substitute $\Pi'_{DLSP}(v_s, v_e) = (v_s = \phi_1, \phi_2, \phi_3, \phi_4, \phi_5 = v_e)$ (i.e., the orange line) as $\Pi_{DLSP}(v_s, v_e)$ if $|\Pi_{DLSP}(v_s, v_e)| < |\Pi'_{DLSP}(v_s, v_e)|$. In Figure 4 (b), we compare the weighted distance among $\Pi'_{DLSP}(v_p, v_n)$, $\Pi_l(v_p, v_n)$, and $\Pi_l(v_p, v_n)$, and substitute $\Pi'_{DLSP}(v_p, v_n)$ 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 4 (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 $S_1 = ((a, b))$ and $S_2 = ((d, e))$ 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.

C EXAMPLE 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 based on T, and we need to check $[50, 100]$, $[75, 100]$, $[75, 87.5]$. After checking the interval $[75, 87.5]$, we get a Π_m that passes the whole S based on T. Assume we calculate m_{ef} as 87 using effective weight pruning step. As a result, in the next checking, we could directly limit the

Notation	Meaning
T	The terrain surface
$V/E/F$	The set of vertices / edges / faces of T
n	The number of vertices of T
N	The smallest integer value which is larger than or equal to the coordinate value of any vertex
W/w	The the maximum / minimum weights of T
L	The length of the longest edge in T
h	The minimum height of any face in F
ϵ	The error parameter for the whole algorithm
ϵ_{SP}	The error parameter for algorithm *DLSP*
ϵ_{SL}	The error parameter for algorithm *BSSL* and algorithm *EWSL*
$\Pi^*(s,t)$	The optimal weighted shortest path
r_i	The intersection point of $\Pi^*(s,t)$ and an edge in E
S^*	The edge sequence that $\Pi^*(s,t)$ connects from s to t in order based on T
$\Pi(s,t)$	The final calculated weighted shortest path
$\|\Pi(s,t)\|$	The weighted distance of $\Pi(s,t)$
$\overline{pq}$	A line between two points p and q on a face
$\Pi'_{DLSP}(s,t)$	The calculated candidate weighted shortest path using algorithm *DLSP* initial path calculation step
ϕ_i	The intersection point of $\Pi'_{DLSP}(s,t)$ and an edge in E
$\Pi_{DLSP}(s,t)$	The calculated candidate weighted shortest path using algorithm *DLSP*
σ_i	The intersection point of $\Pi_{DLSP}(s,t)$ and an edge in E
ξ	The iteration counts of single endpoint cases in algorithm *DLSP* full edge sequence conversion step
S	The edge sequence that $\Pi_{DLSP}(s,t)$ connects from s to t in order based on T
l	The number of edges in S
$\Pi^*(s,t\|S)$	The optimal weighted shortest path passes S
ψ_i	The intersection point of $\Pi^*(s,t\|S)$ and an edge in E
$\Pi_{SL}(s,t\|S)$	The calculated *SL*-weighted shortest path using algorithm *EWSL*
ρ_i	The intersection point of $\Pi_{SL}(s,t\|S)$ and an edge in E
Π_c	A 3D surface Snell's ray

Table 3: Summary of frequent used notations

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.

D EMPIRICAL STUDIES

D.1 Experimental Results

(1) Figure 14 and Figure 15, (2) Figure 16 and Figure 17, (3) Figure 18 and Figure 19 show the result on the *BH-small* dataset when varying ϵ, ϵ_{SP}, and ϵ_{SL}, respectively. (4) Figure 20 and Figure 21, (5) Figure 22 and Figure 23, (6) Figure 24 and Figure 25 show the result on the *BH* dataset when varying ϵ, ϵ_{SP}, and ϵ_{SL}, respectively. (7) Figure 8 and Figure 26, (8) Figure 27 and Figure 28, (9) Figure 29 and Figure 30 show the result on the *EP-small* dataset when varying ϵ, ϵ_{SP}, and ϵ_{SL}, respectively. (10) Figure 31 and Figure 32, (11) Figure 33 and Figure 34, (12) Figure 35 and Figure 36 show the result on the *EP-small* dataset when varying ϵ, ϵ_{SP}, and ϵ_{SL}, respectively. (13) Figure 37 and Figure 38 show the result on a set of small-version datasets when varying *DS*. (14) Figure 39 and Figure 40 show the result on a set of large-version datasets when varying *DS*.

Effect of ϵ. In Figure 14, Figure 20, Figure 8 and Figure 31, 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 15, Figure 21, Figure 26 and Figure 32 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 *DLSP-EWSL* is smaller than *FSP* and *LSP*, since *DLSP-EWSL* follows Snell's law.

Effect of ϵ_{SP}. In Figure 16, Figure 22, Figure 27 and Figure 33, 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 17, Figure 23, Figure 28 and Figure 34 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 much 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 18, Figure 24, Figure 29 and Figure 35, 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 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 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. The minimum value of the query time of *EWSL* (both *FSP-EWSL* and *DLSP-EWSL*) is also always smaller than *BSSL* (both *FSP-BSSL* and *DLSP-BSSL*).

Effect of ***DS*** **(scalability test)**. In Figure 37 and Figure 39, 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 {1M, 2M, 3M, 4M, 5M} on a set of large-version datasets (by setting ϵ to be 0.25)

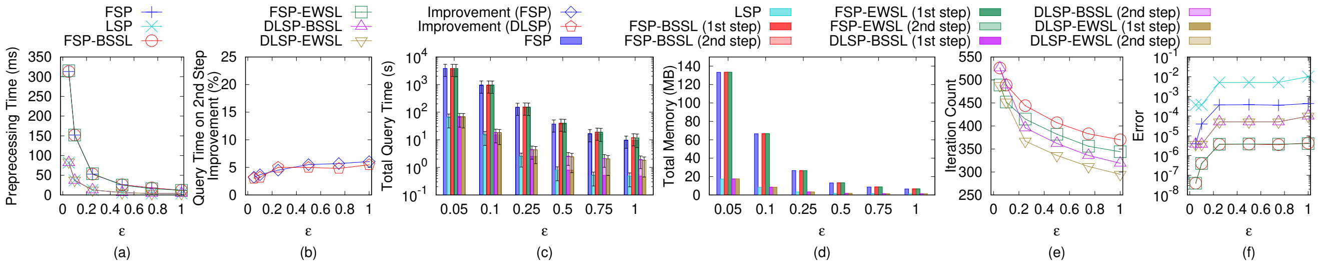

Figure 14: Effect of ϵ on *BH-small* dataset

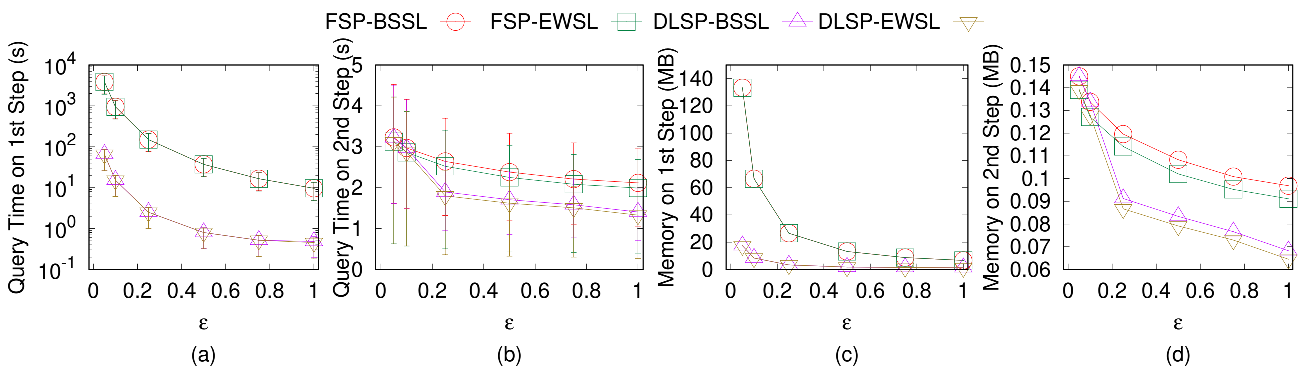

Figure 15: Effect of ϵ on *BH-small* dataset with separated query time and memory usage in two steps

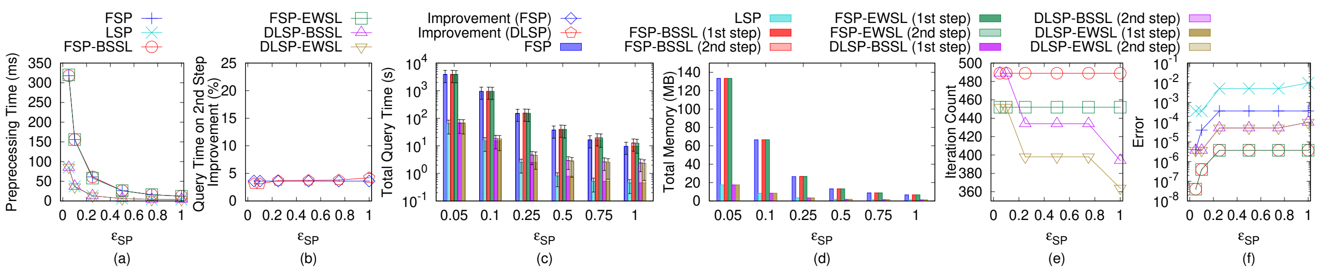

Figure 16: Effect of ϵ_{SP} on *BH-small* dataset

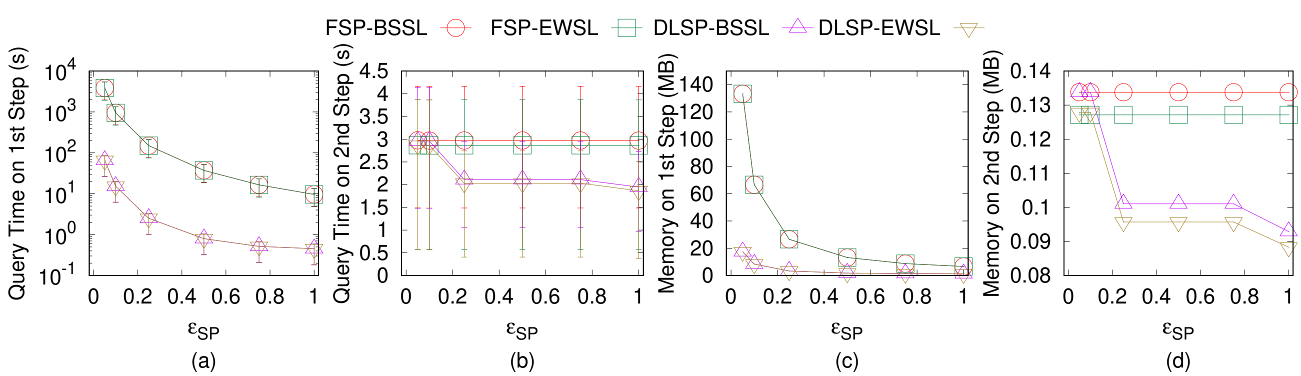

Figure 17: Effect of ϵ_{SP} on *BH-small* dataset with separated query time and memory usage in two steps

for scalability test. Figure 38 and Figure 40 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

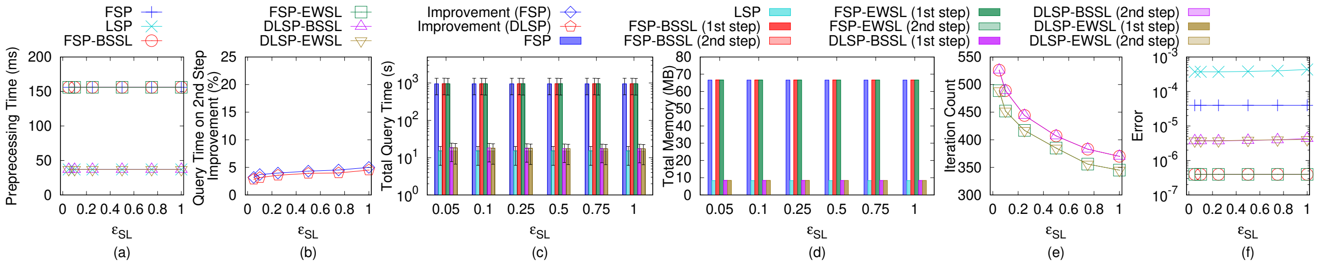

Figure 18: Effect of ϵ_{SL} on *BH-small* dataset

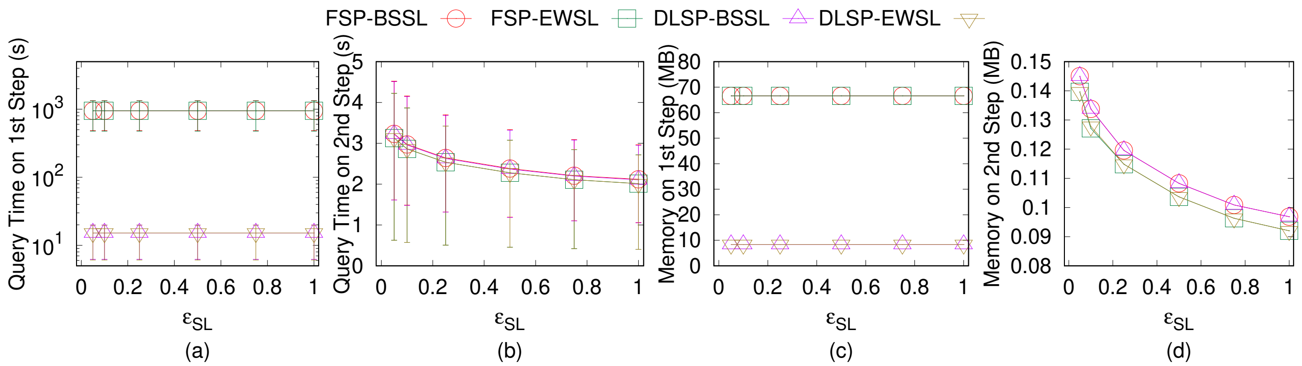

Figure 19: Effect of ϵ_{SL} on *BH-small* dataset with separated query time and memory usage in two steps

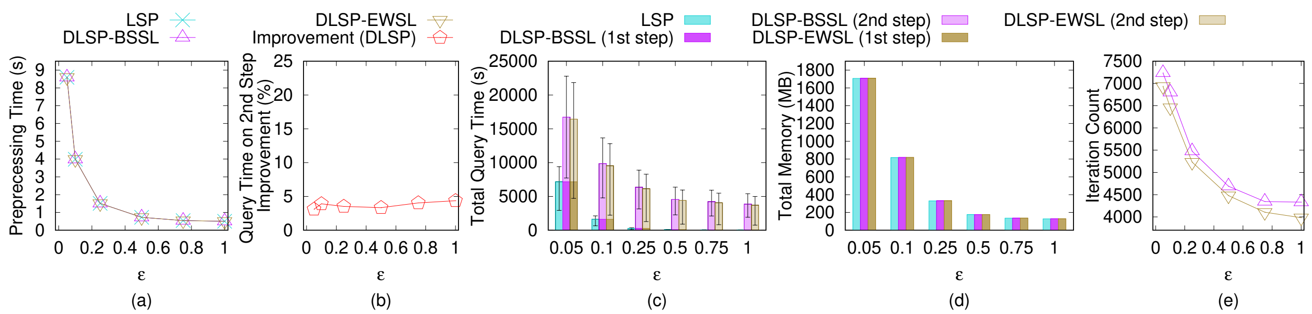

Figure 20: Effect of ϵ on *BH* dataset

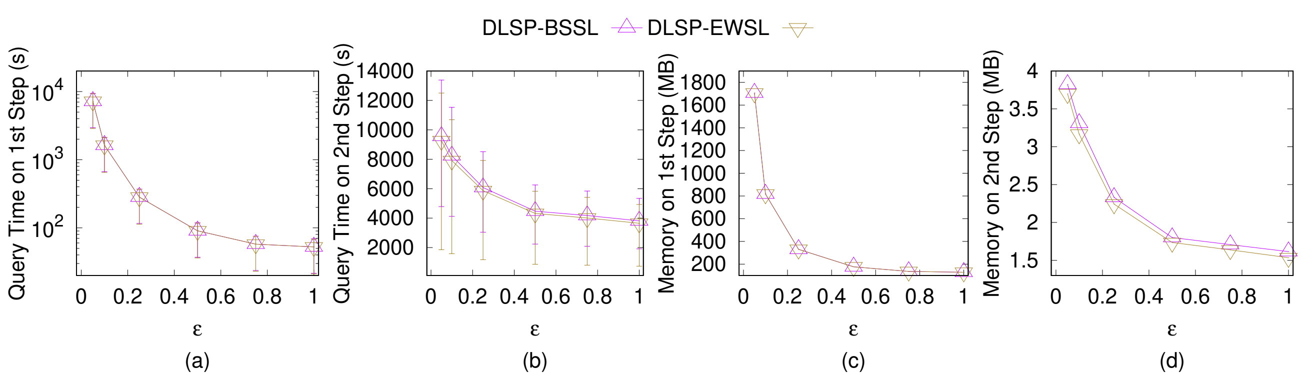

Figure 21: Effect of ϵ on *BH* dataset with separated query time and memory usage in two steps

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

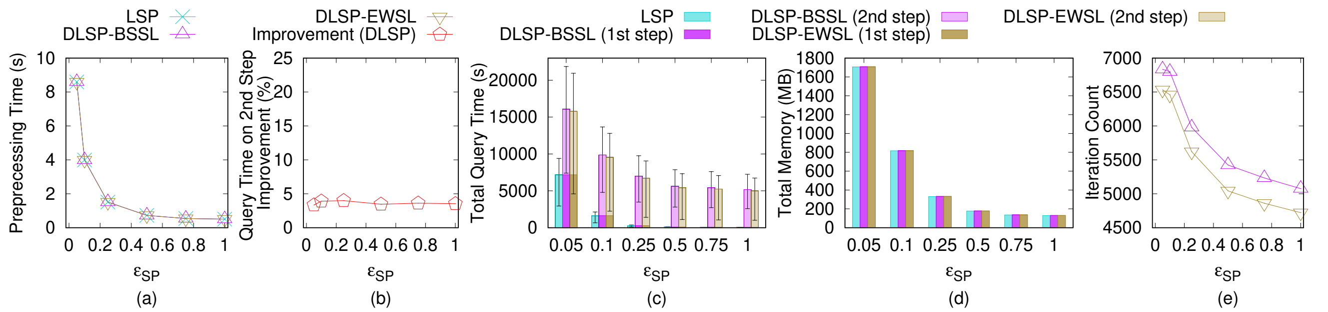

Figure 22: Effect of ϵ_{SP} on *BH* dataset

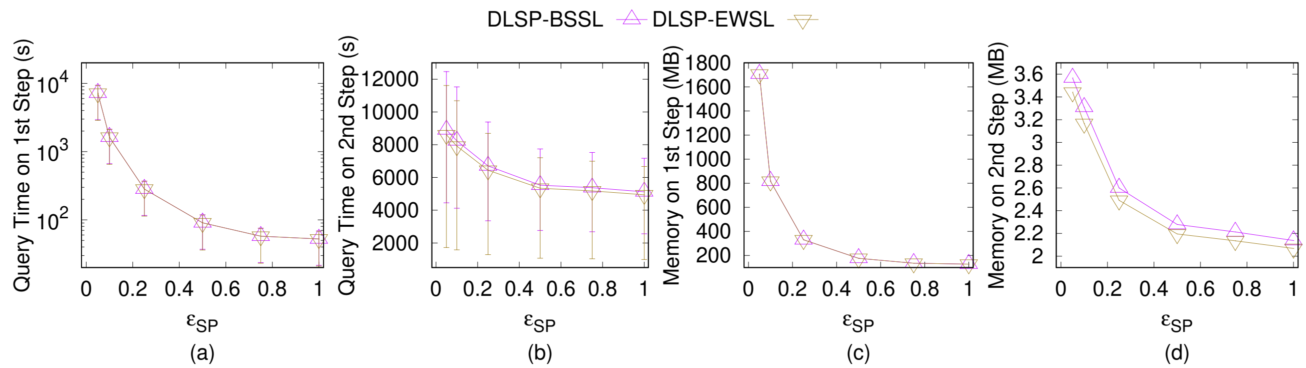

Figure 23: Effect of ϵ_{SP} on *BH* dataset with separated query time and memory usage in two steps

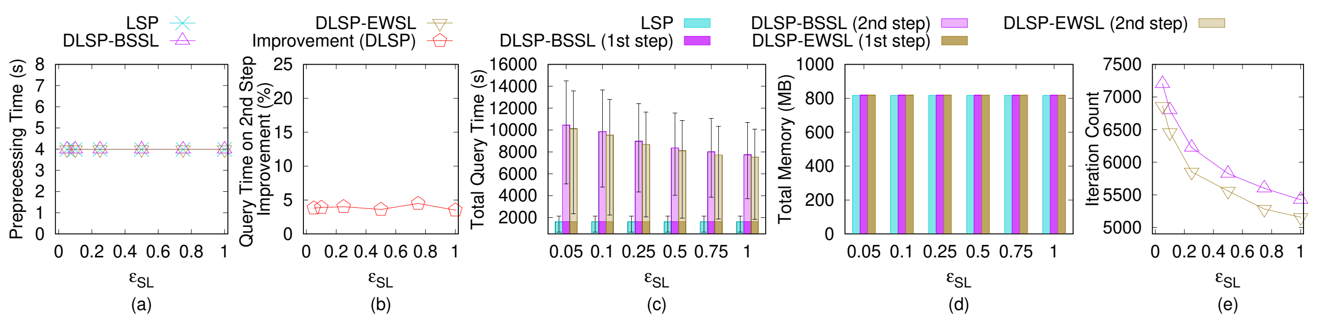

Figure 24: Effect of ϵ_{SL} on *BH* dataset

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. On the set of large-version datasets, algorithm *DLSP-EWSL* could still return a path that follows Snell's law in reasonable time.

D.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 [32, 37, 38]. Let $T_t = (V_t, E_t, F_t)$ be our target terrain that we want to generate with ex_t edges along x-coordinate, ey_t edges along y-coordinate and dataset size of DS_t, where $DS_t = 2 \cdot ex_t \cdot ey_t$. Let $T_o = (V_o, E_o, F_o)$ be the original terrain that we currently have with ex_o edges along x-coordinate, ey_o edges along y-coordinate and dataset size of DS_o, where $DS_o = 2 \cdot ex_o \cdot ey_o$. We then generate $(ex_t + 1) \cdot (ey_t + 1)$ 2D points (x, y) based on a Normal distribution $N(\mu_N, \sigma_N^2)$, where $\mu_N = (\overline{x} = \frac{\sum_{v_o \in V_o} x_{v_o}}{(ex_o+1)\cdot(ey_o+1)}, \overline{y} = \frac{\sum_{v_o \in V_o} y_{v_o}}{(ex_o+1)\cdot(ey_o+1)})$ and $\sigma_N^2 = (\frac{\sum_{v_o \in V_o}(x_{v_o} - \overline{x})^2}{(ex_o+1)\cdot(ey_o+1)}, \frac{\sum_{v_o \in V_o}(y_{v_o} - \overline{y})^2}{(ex_o+1)\cdot(ey_o+1)})$. In the end, we project each generated point (x, y) to the surface of T_o and take the projected point (also add edges between neighbours of two points to form edges and faces) as the newly generate T_t.

D.3 Case Study

D.3.1 User Study (Path Advisor). Figure 41 and Figure 42 show the result for Path Advisor user study when varying ϵ. Our user study

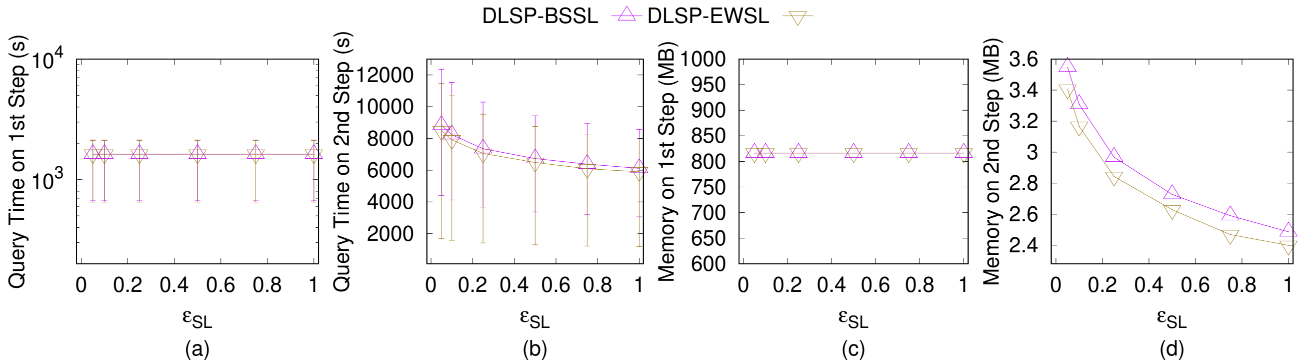

Figure 25: Effect of ϵ_{SL} on *BH* dataset with separated query time and memory usage in two steps

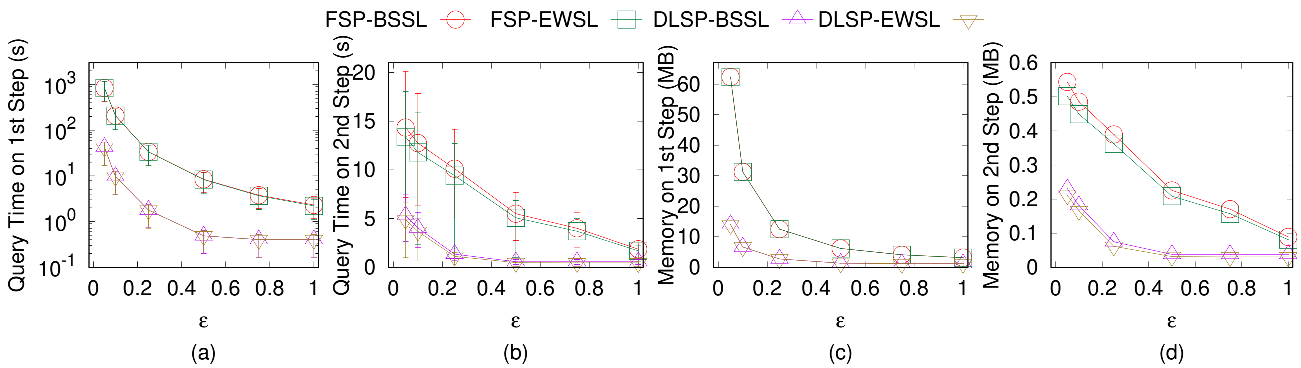

Figure 26: Effect of ϵ on *EP-small* dataset with separated query time and memory usage in two steps

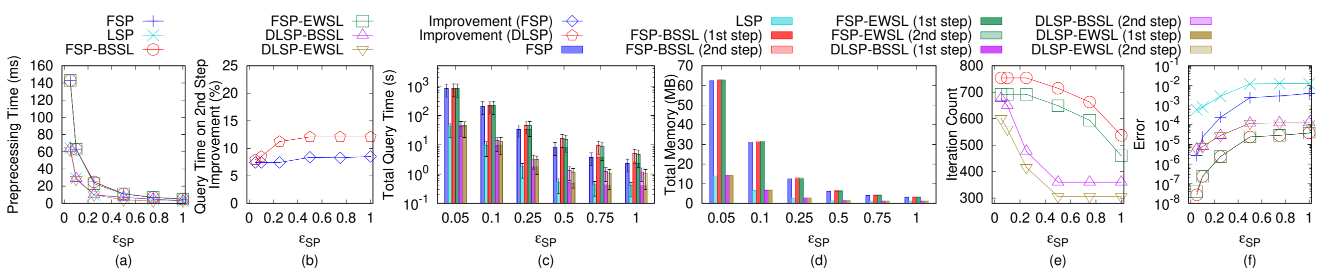

Figure 27: Effect of ϵ_{SP} on *EP-small* dataset

in Section 5.3.1 has already shown that most users think the blue path (i.e., the *SL*-weighted shortest path that follows Snell's law calculated using algorithm *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL*) is the most realistic one. Our user study in Section 5.3.1 has also already shown that when $\epsilon = 0.5$, the total query times for algorithm *FSP*, *LSP*, *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL* are 16.64s, 0.28s, 17.44s, 17.43s, 0.39s and 0.38s, respectively, which shows that *DLSP-EWSL* is still the best given that the path need to follow Snell's law. These results could be found in Figure 41 and Figure 42. In addition, in a map application, the query time is the most crucial factor since users would like to get the result in a shorter time. Thus, *DLSP-EWSL* is the most suitable algorithm for Path Advisor.

D.3.2 User Study (Cyberpunk 2077). We conducted another user study on Cyberpunk 2077 [2], a popular 3D computer game. The dataset is *CP* dataset [3] used in our experiment. We set the weight of a triangle in terrain to be the slope of that face. We randomly selected two points as source and destination, respectively, and repeated it for 100 times to calculate the path. Figure 43 and Figure 44 show the result for Cyberpunk 2077 user study when varying ϵ. In these two figures, when $\epsilon = 0.5$, the total query times for algorithm *FSP*, *LSP*, *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL* are 21.61s, 0.62s, 22.79s, 22.73s, 1.22s and 1.19s, respectively. It still shows that *DLSP-EWSL* is the best given that the path need to follow Snell's law. It is important to get real-time response in computer games. Thus, *DLSP-EWSL* is the most suitable algorithm for Cyberpunk 2077.

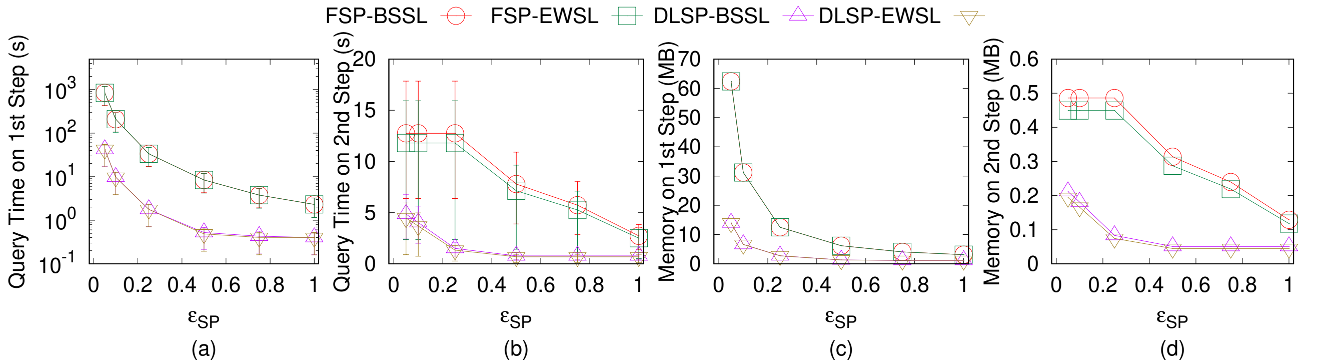

Figure 28: Effect of ϵ_{SP} on *EP-small* dataset with separated query time and memory usage in two steps

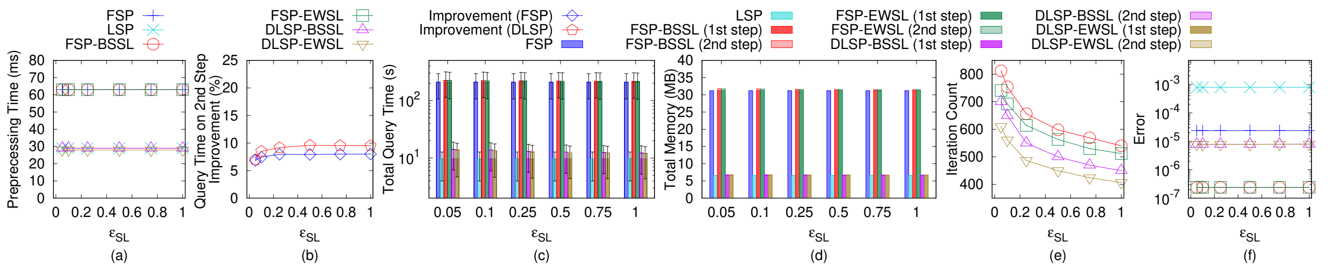

Figure 29: Effect of ϵ_{SL} on *EP-small* dataset

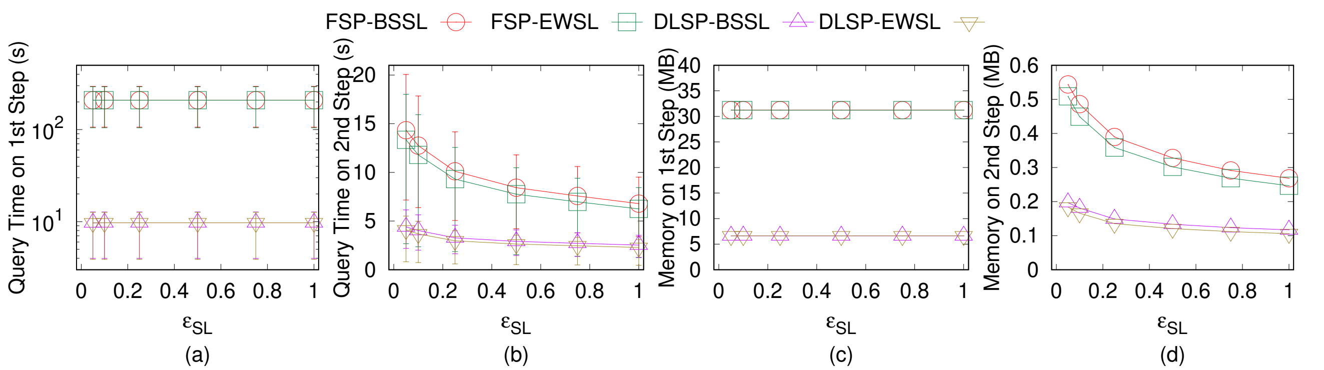

Figure 30: Effect of ϵ_{SL} on *EP-small* dataset with separated query time and memory usage in two steps

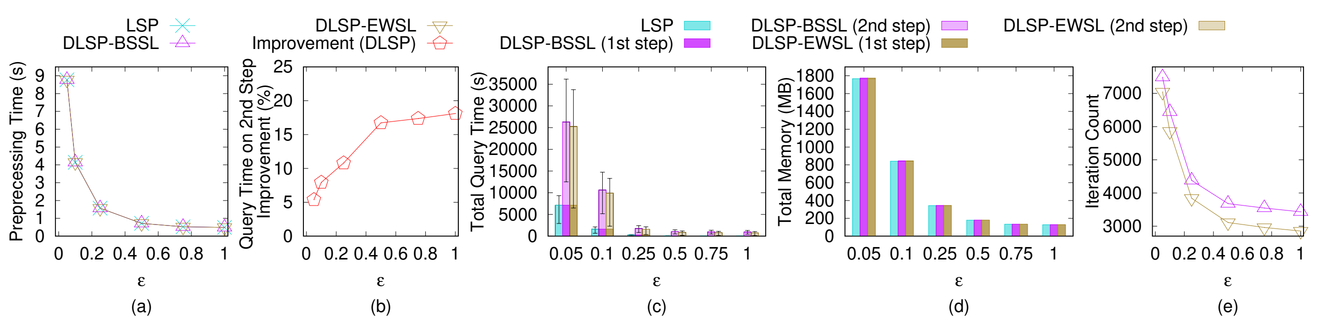

Figure 31: Effect of ϵ on *EP* dataset

D.3.3 Motivation Study. Figure 45 and Figure 46 show the result for seabed motivation study when varying ϵ. Our motivation study

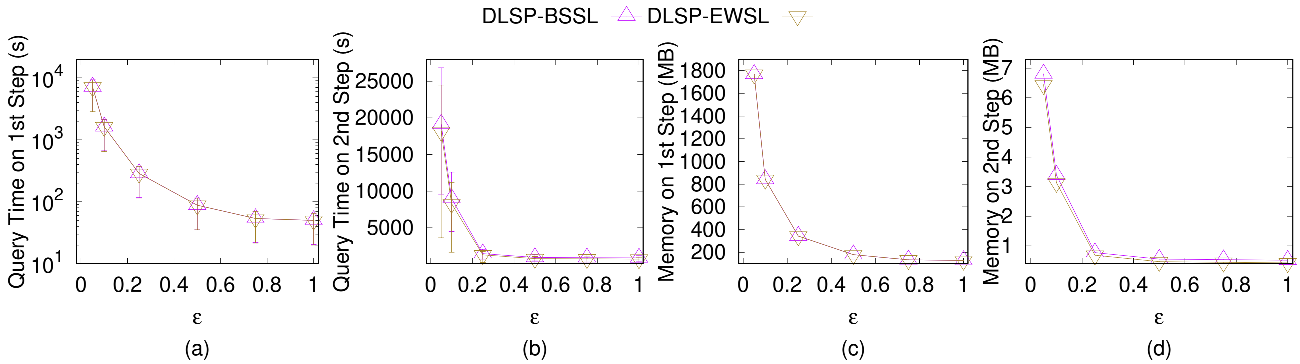

Figure 32: Effect of ϵ on *EP* dataset with separated query time and memory usage in two steps

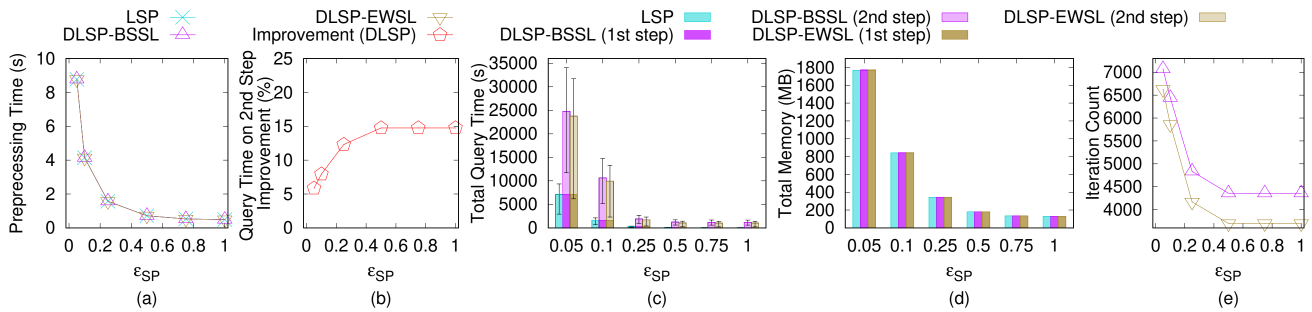

Figure 33: Effect of ϵ_{SP} on *EP* dataset

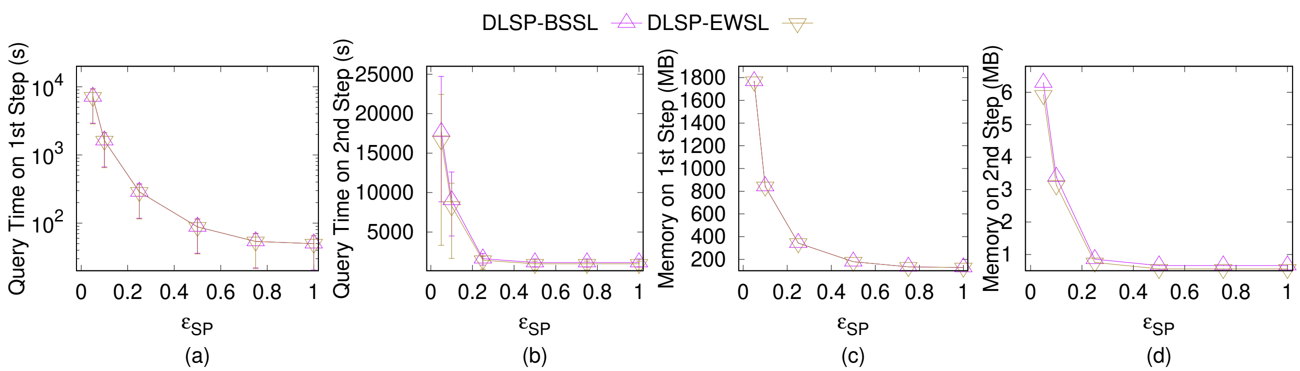

Figure 34: Effect of ϵ_{SP} on *EP* dataset with separated query time and memory usage in two steps

in Section 5.3.2 has already shown that the blue path (i.e., the *SL*-weighted shortest path that follows Snell's law calculated using algorithm *FSP-BSSL*, *FSP-EWSL*, *DLSP-BSSL* and *DLSP-EWSL*) is the most realistic one since it could avoid the regions with higher hydraulic pressure, and thus, could reduce the construction cost of undersea optical fiber cable. In Figure 45 and Figure 46, *DLSP-EWSL* has the smallest proprecessing 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.

E PROOFS

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

Proof. We prove it for the extreme case, i.e., k_{SP} is maximized. This case happens when the edge has maximum length L and it joins two vertices has minimum radius r. Since each edge contains two endpoints, we have two sets of Steiner points from both endpoints, and we have the factor 2. From algorithm *DLSP* initial path calculation step, 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 $k_{SP} \leq 2(1 + \log_\lambda \frac{L}{r})$. ◻

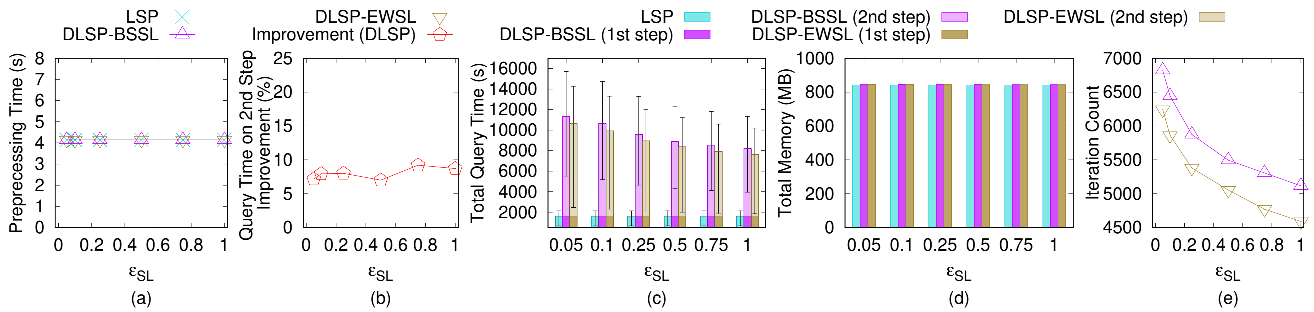

Figure 35: Effect of ϵ_{SL} on *EP* dataset

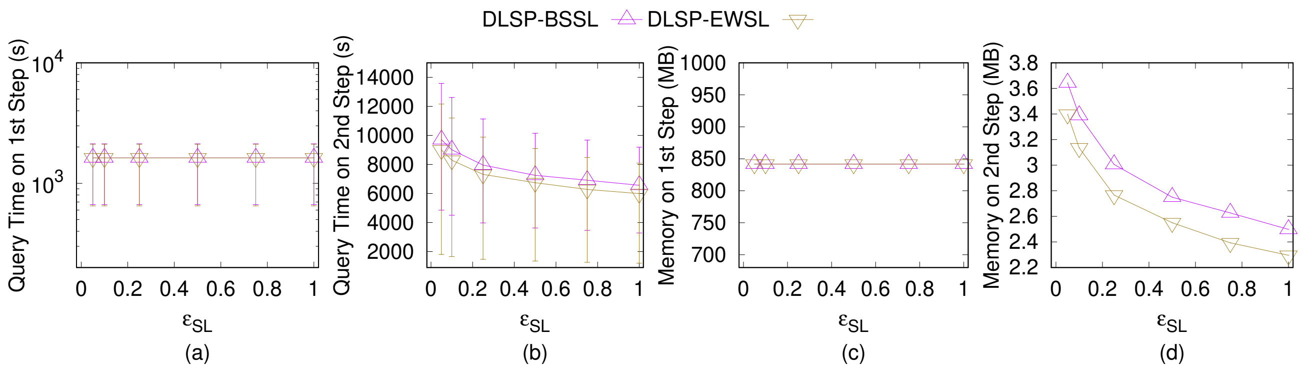

Figure 36: Effect of ϵ_{SL} on *EP* dataset with separated query time and memory usage in two steps

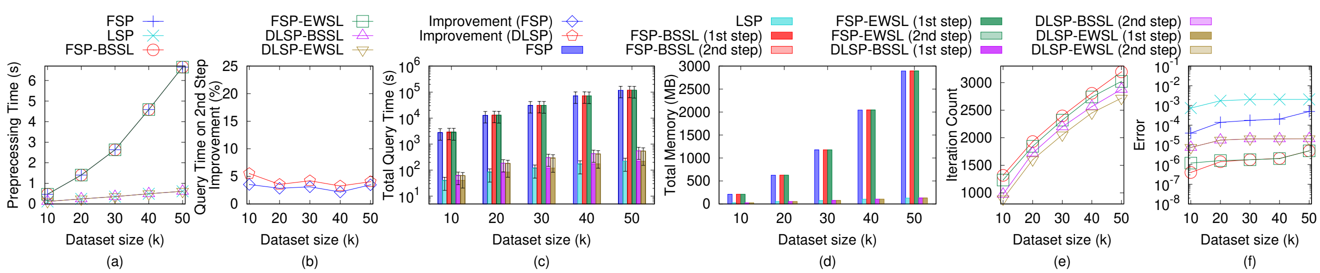

Figure 37: Effect of dataset size on a set of small-version datasets

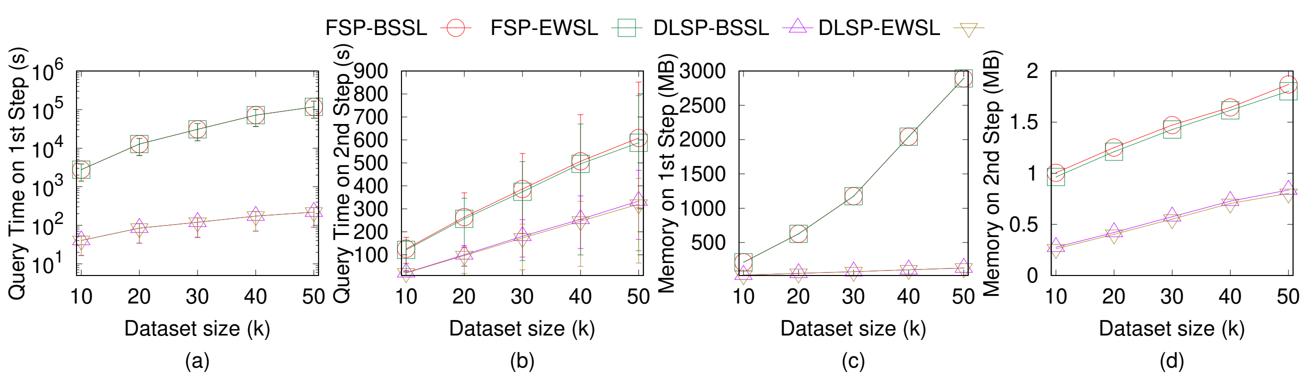

Figure 38: Effect of dataset size on a set of small-version datasets with separated query time and memory usage in two steps

Lemma E.2. *In algorithm DLSP initial path calculation step,* $\epsilon'_{SP} = \frac{1+\epsilon_{SP}+\frac{W}{w}-\sqrt{(1+\epsilon_{SP}+\frac{W}{w})^2-4\epsilon_{SP}}}{4}$ *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

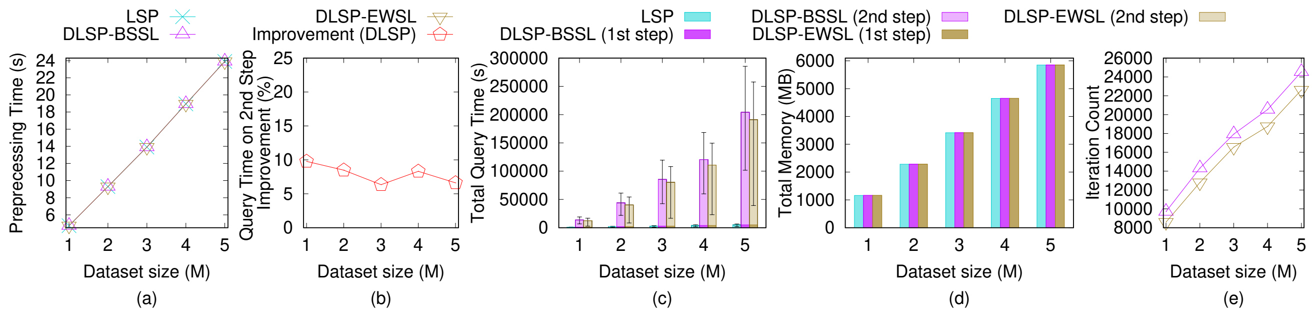

Figure 39: Effect of dataset size on a set of large-version datasets

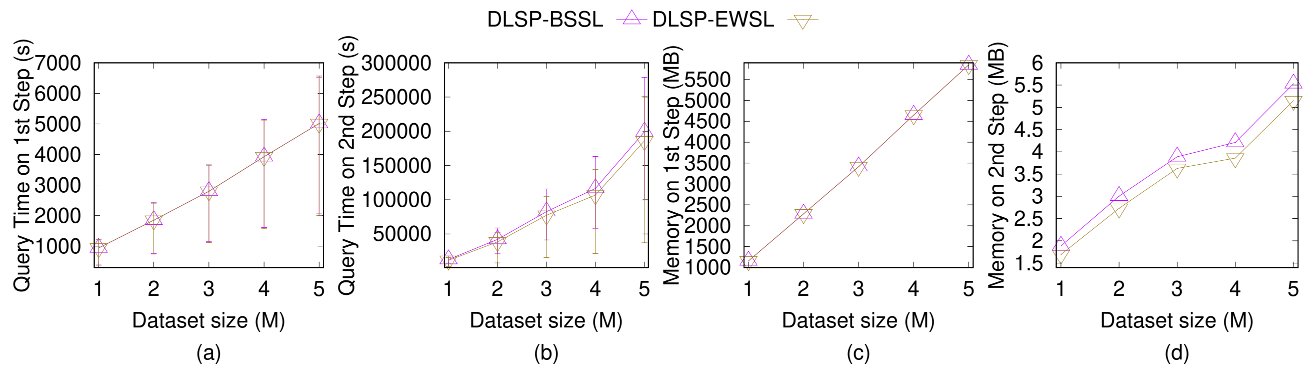

Figure 40: Effect of dataset size on a set of large-version datasets with separated query time and memory usage in two steps

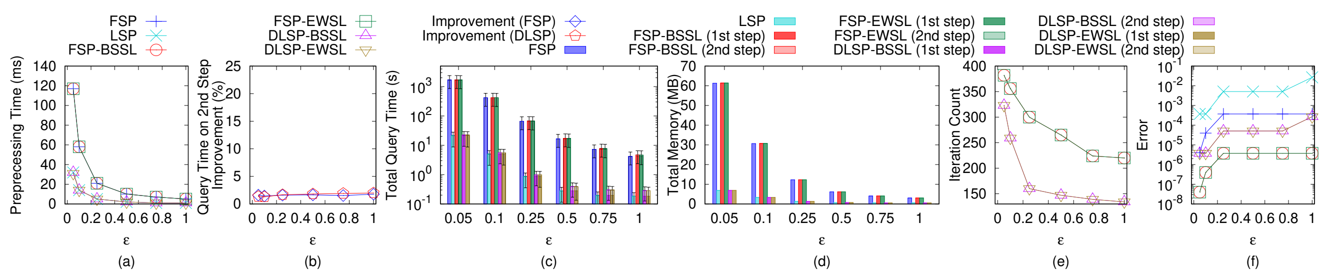

Figure 41: Effect of ϵ for Path Advisor user study

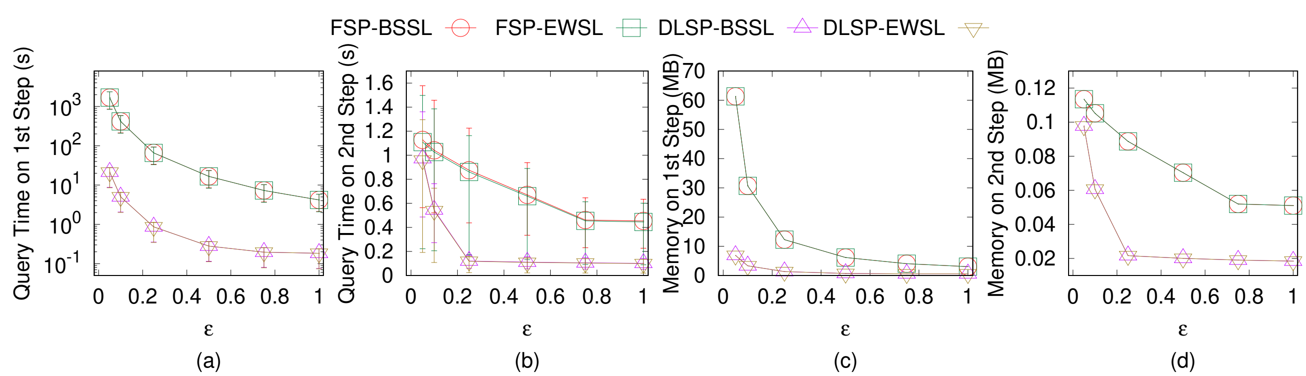

Figure 42: Effect of ϵ for Path Advisor user study with separated query time and memory usage in two steps

follows.

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

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

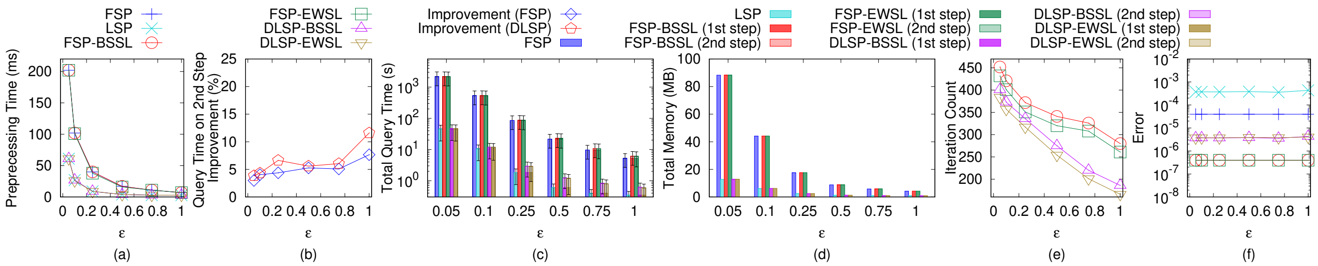

Figure 43: Effect of ϵ for Cyberpunk 2077 user study

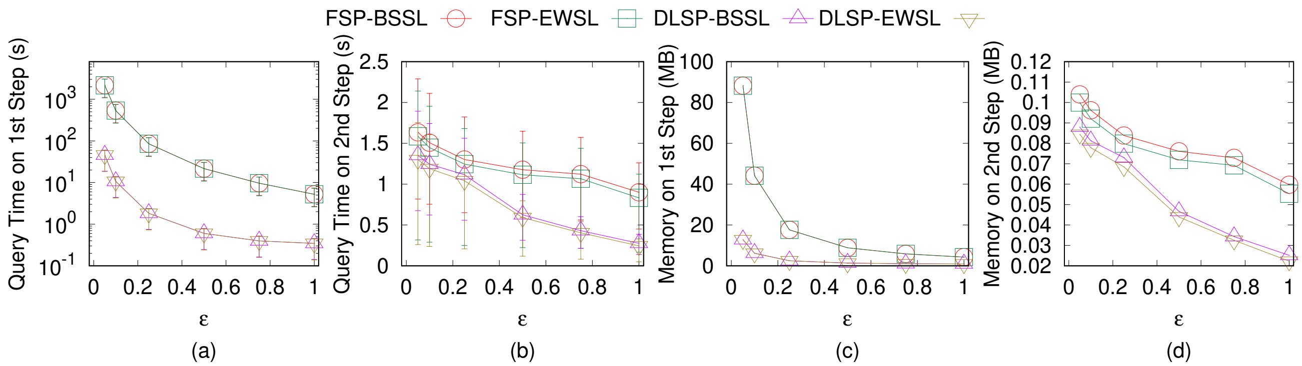

Figure 44: Effect of ϵ for Cyberpunk 2077 user study with separated query time and memory usage in two steps

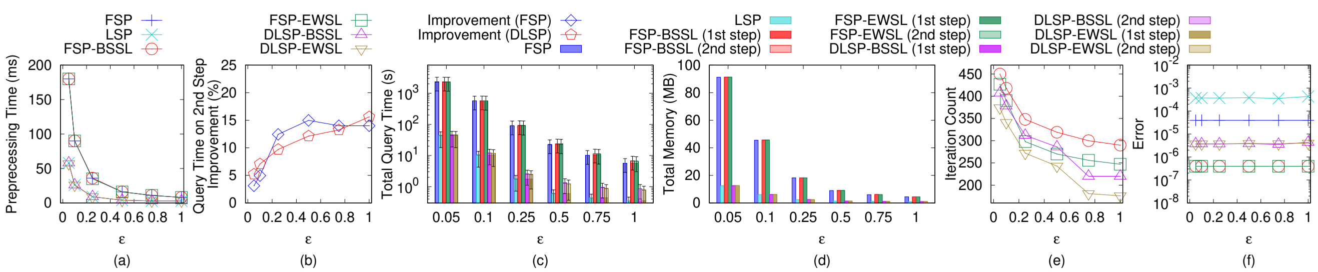

Figure 45: Effect of ϵ for seabed motivation study

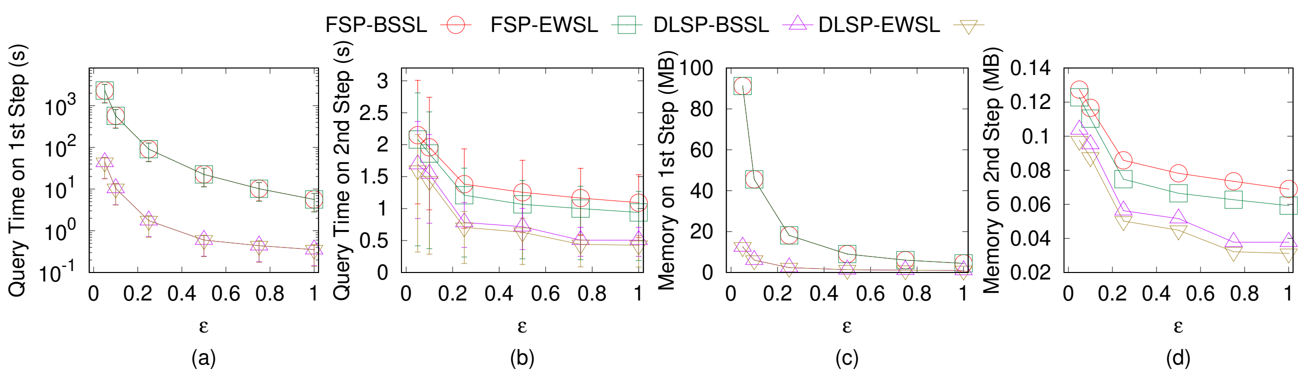

Figure 46: Effect of ϵ for seabed motivation study with separated query time and memory usage in two steps

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

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

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

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

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

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 E.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 colinear. 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 $\eta = (y_a z_b - z_a y_b)^2 + (z_a x_b - x_a z_b)^2 + (x_a y_b - y_a x_b)^2$. □

Theorem E.4. *The running time for algorithm DLSP initial path calculation step 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 step guarantees that $|\Pi'_{DLSP}(s,t)| \leq (1 + \epsilon_{SP})|\Pi^*(s,t)|$.*

Proof of Theorem E.4. Firstly, we prove the running time and memory usage. Following Lemma E.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 E.2 and Lemma E.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 prove the error bound. A proof sketch of this could be found in Theorem 1 of [9] and a detailed proof could be found in Theorem 3.1 of [27]. But, in [9, 27], they have $|\Pi'_{DLSP}(s,t)| \leq (1 + (2 + \frac{2W}{(1-2\epsilon'_{SP})\cdot w})\epsilon'_{SP})|\Pi^*(s,t)|$ 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 $|\Pi'_{DLSP}(s,t)| \leq (1 + \epsilon_{SP})|\Pi^*(s,t)|$ where $\epsilon_{SP} > 0$. □

Proof of Theorem 3.1. Firstly, we prove the average case running time. For the average case, we assume $\frac{1}{3}$ of $\Pi'_{DLSP}(s,t)$ passes on the edge (i.e., no need to use algorithm *DLSP* full edge sequence conversion step for refinement), $\frac{1}{3}$ of $\Pi'_{DLSP}(s,t)$ belongs to single endpoint case, and the remaining $\frac{1}{3}$ of $\Pi'_{DLSP}(s,t)$ belongs to successive endpoint case. Let the number of path segments in $\Pi'_{DLSP}(s,t)$ be l and we know $l = O(n^2)$. Let $T'_{DLSP} = O(n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}))$ be the running time of algorithm *DLSP* initial path calculation step, whose running time will not affected by l. Let $T_{avg}(l)$ be the average case running time for algorithm *DLSP* full edge sequence conversion step in terms of l and $T_{avg}(n)$ be the average case running time for algorithm *DLSP* full edge sequence conversion step in terms of n. Then, $T_{avg}(l) = T'_{DLSP} + T_{avg}(\frac{l}{3}) + O(\frac{\zeta l}{3}) = T'_{DLSP} + O(\zeta l)$, and $T_{avg}(n^2) = T'_{DLSP} + O(\zeta n^2)$. Therefore, $T_{avg}(n) = T'_{DLSP} + O(\zeta n) = O(n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}) + \zeta n) = O(\mu_1 n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}) + \zeta n)$, where μ_1 is $O(1)$ in the average case.

Secondly, we prove the worst case running time. The worst case would be that all the points in $\Pi'_{DLSP}(s,t)$ passes the original vertices in V (i.e., all the vertices are successive endpoints), and there will be $O(n)$ of these points. Then, algorithm *DLSP* initial path calculation step is applied, and only one point is refined (i.e., only one point in $\Pi'_{DLSP}(s,t)$ is refined on the edge). Let $T_{worst}(l)$ be the worst case running time for algorithm *DLSP* initial path calculation step in terms of l and $T_{worst}(n)$ be the average case running time for algorithm *DLSP* initial path calculation step in terms of n. Then, $T_{worst}(l) = T'_{DLSP} + T_{worst}(l-1) = O(l) \cdot T'_{DLSP}$, and $T_{worst}(n^2) = O(n^2) \cdot T'_{DLSP}$. Therefore, $T_{worst}(n) = O(n) \cdot T'_{DLSP} = O(n^2 \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}})) \leq O(n^2 \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}) + \zeta n) = O(\mu_1 n \log \frac{LN}{\epsilon_{SP}} \log(n \log \frac{LN}{\epsilon_{SP}}) + \zeta n)$, where μ_1 is $O(n)$ in the worst case.

Thirdly, we prove the memory usage. Algorithm *DLSP* initial path calculation step 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 prove the error bound. In algorithm *DLSP* full edge sequence conversion step, we only use the refinement path $\Pi_{DLSP}(s,t)$ if its weighted distance is shorter than $\Pi'_{DLSP}(s,t)$ (i.e., $|\Pi_{DLSP}(s,t)| \leq |\Pi'_{DLSP}(s,t)|$). So, using Theorem E.4, we know $|\Pi_{DLSP}(s,t)| \leq |\Pi'_{DLSP}(s,t)| \leq (1 + \epsilon_{SP})|\Pi^*(s,t)|$. 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 $|\Pi_{DLSP}(s,t)| > |\Pi'_{DLSP}(s,t)|$, then we will not use $\Pi_{DLSP}(s,t)$ for calculating the edge sequence S, we still use $\Pi'_{DLSP}(s,t)$. In this case, the error ratio is still the error ratio of algorithm *DLSP* initial path calculation step, i.e., $|\Pi'_{DLSP}(s,t)| \leq (1 + \epsilon_{SP})|\Pi^*(s,t)|$. □

Proof of Theorem 3.2. Firstly, we prove the best case running time. Let l be the number of edges in S. In the best case, the algorithm *EWSL* effective weight pruning step could directly find the optimal position of the intersection point on the first edge in S in $O(1)$ time. According to Lemma 7.1 in [33], $l = O(n^2)$, so the best case running time of algorithm *EWSL* is $O(n^2) = O(\mu_2 n^2)$, where μ_2 is $O(1)$ in the best case.

Secondly, we prove the worst case running time. In the algorithm *EWSL* binary search initial path and *SL*-weighted shortest path finding step, it first takes $O(l)$ time for computing the 3D surface Snell's ray Π_m since there are l edges in S and we need to use Snell's law l times to calculate the intersection point on each edge. 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 [33], $l = O(n^2)$, so the worst case running time of algorithm *EWSL* is $O(n^4 \log \frac{nWL}{h\epsilon_{SL} w}) = O(\mu_2 n^2)$, where μ_2 is $O(n^2 \log(\frac{nNW}{w\epsilon}))$ in the worst case.

Thirdly, we prove the memory usage, since the *SL*-weighted shortest path that follows Snell's law will pass l edges, so the memory usage is $O(l) = O(n^2)$.

Finally, we prove 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:

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

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 3.3. Firstly, we prove 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 3.1 and Theorem 3.2. Since we let $\epsilon = \epsilon_{SP} = \epsilon_{SL}$, we could get the running time and the memory usage in terms of ϵ.

Secondly, we prove the error bound. Following Theorem 3.1 and Theorem 3.2, we have proved that $|\Pi'_{DLSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)|$ 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 based on T, 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 based on T, and these two terms are actually the same thing.

Case two: $S = S^*$ and $|\Pi_{SL}(s,t|S)| > |\Pi_{DLSP}(s,t)|$: $|\Pi(s,t)| = \min(|\Pi_{DLSP}(s,t)|, |\Pi_{SL}(s,t|S)|) = |\Pi_{DLSP}(s,t)| \leq |\Pi'_{DLSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)| = (1+\epsilon)|\Pi^*(s,t)|$.

Case three: $S \neq S^*$ and $|\Pi_{SL}(s,t|S)| \leq |\Pi_{DLSP}(s,t)|$: $|\Pi(s,t)| = \min(|\Pi_{DLSP}(s,t)|, |\Pi_{SL}(s,t|S)|) \leq |\Pi_{DLSP}(s,t)| \leq |\Pi'_{DLSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)| = (1+\epsilon)|\Pi^*(s,t)|$.

Case four: $S \neq S^*$ and $|\Pi_{SL}(s,t|S)| > |\Pi_{DLSP}(s,t)|$: $|\Pi(s,t)| = \min(|\Pi_{DLSP}(s,t)|, |\Pi_{SL}(s,t|S)|) = |\Pi_{DLSP}(s,t)| \leq |\Pi'_{DLSP}(s,t)| \leq (1+\epsilon_{SP})|\Pi^*(s,t)| = (1+\epsilon)|\Pi^*(s,t)|$.

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