Finding Shortest Paths on Terrains by Killing Two Birds with One Stone

Manohar Kaul[†] Raymond Chi-Wing Wong[‡] Bin Yang[†] Christian S. Jensen[†]
[†]Aarhus University
[†]{mkaul,byang,csj}@cs.au.dk
[‡]The Hong Kong University of Science and Technology
[‡]{raywong}@cse.ust.hk

ABSTRACT

With the increasing availability of terrain data, e.g., from aerial laser scans, the management of such data is attracting increasing attention in both industry and academia. In particular, spatial queries, e.g., k-nearest neighbor and reverse nearest neighbor queries, in Euclidean and spatial network spaces are being extended to terrains. Such queries all rely on an important operation, that of finding shortest surface distances. However, shortest surface distance computation is very time consuming. We propose techniques that enable efficient computation of lower and upper bounds of the shortest surface distance, which enable faster query processing by eliminating expensive distance computations. Empirical studies show that our bounds are much tighter than the best-known bounds in many cases and that they enable speedups of up to 43 times for some well-known spatial queries.

1. INTRODUCTION

We are witnessing an increasing availability of terrain data: More regions are being covered, and the coverage is becoming increasingly accurate and up-to-date. This acquisition of terrain data is motivated by a plethora of applications that are not constrained by road networks.

The defense industry was amongst the earliest to recognize the importance of terrain models to simulate battlefield landscapes to allow *tactical path planning* [4] through valleys (or across ridges) of the terrain via shortest terrain paths. Other applications include robot path planning for unmanned vehicles on terrains and geo-realistic computer games.

In academia, several recent studies [5, 8, 10–12] focus on the challenges presented by terrain data. For example, they consider "terrain" versions of some well-known spatial queries such as shortest path queries, k-nearest neighbor queries, and reverse nearest neighbor queries.

Surface shortest path queries [8] are fundamental in their own right and occur as an aspect of many other spatial queries (e.g., *surface k-nearest neighbor (k-NN) queries* [5, 10, 11], *surface range queries*, and *surface reverse nearest neighbor queries* [12]). Given a source point s and a destination point t on a terrain, a *shortest surface path query* returns the shortest surface path from s to t on the surface. Figure 1 shows a surface path P from s to t, the shortest surface path Π_s from s to t and the direct Euclidean distance Π_E from s to t. Given a set of objects and a query point q on the surface, a *surface k-NN query* returns k objects on the surface such that no other objects are closer to q, where the "closeness" is computed by a surface shortest path query.

Computing surface shortest paths is much more challenging and more expensive than computing network shortest paths on a road network. Specifically, the best-known algorithm for finding the surface shortest path is the Chen-and-Han algorithm [3] that is recognized as the state-of-the-art algorithm in the literature [5, 8, 10, 11]. Its time complexity is $O(N^2)$, where N is the number of vertices used to represent the terrain.

In our experiments, when there are 20K vertices, this algorithm takes 7.2 hours to find a surface shortest path, which is extremely time-consuming. In contrast, Dijkstra's algorithm [6] takes only 0.04 seconds to find the corresponding shortest network path when the dataset used has the same number of vertices.

Motivated by this observation, several existing studies [5, 8, 10–12] propose *efficient* methods to find *lower* and *upper bounds* of the shortest surface distance. These can then be used to avoid some of the *expensive* surface distance computations inherent in spatial queries.

To illustrate, consider a surface 1-NN query. Suppose that q is the query point and there are two objects, o_1 and o_2. If the lower bound of the shortest surface distance between o_1 and q exceeds the upper bound of the shortest surface distance between o_2 and q then o_1 can be pruned. Thus, we do not consider o_1 and need not calculate the exact shortest surface distance between o_1 and q.

A popular method for finding the upper bound of the shortest surface path distance from a source s to a destination t is to find the *shortest network distance* from s to t based on a *Delaunay graph* of the terrain [1]. Figure 2 shows the Delaunay graph of the terrain in Figure 1. In Figure 2, Π'_n is a network path from s to t, and Π_n is the shortest network path from s to t.

We can map each network path in the Delaunay graph to a path on the surface of the terrain (by mapping the vertices and edges used in the network path). Intuitively, each mapped path is a surface path on the terrain with the constraint that the path must pass through only the vertices and the edges of the Delaunay graph. Figure 3 shows the surface path p that is obtained from the Delaunay graph network path Π'_n.

The reason why the shortest network distance is commonly used

Proceedings of the VLDB Endowment, Vol. 7, No. 1

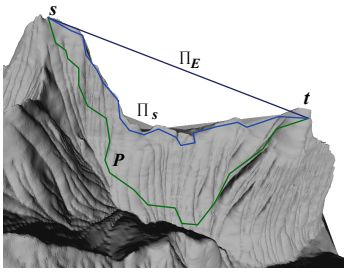


Figure 1: A Terrain

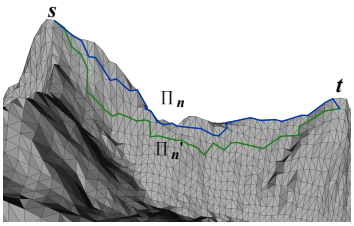


Figure 2: Delaunay Graph

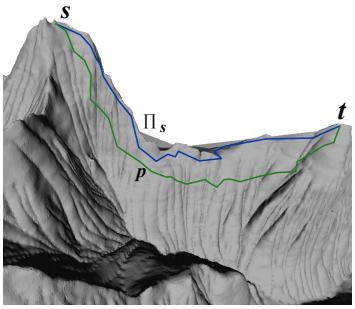


Figure 3: Surface Path From the Delaunay Graph

as the upper bound of the shortest surface distance [5, 8, 10, 12] is that this bound is tight and is computationally cheap compared to the shortest surface distance computation.

Next, it is popular to lower bound the shortest surface path distance by the *Euclidean distance*, which is cheap to compute. However, this lower bound can be very *loose*. To illustrate, recall Figure 1, where Π_E is the Euclidean distance. This distance does not capture any information about the surface of the terrain, and thus it is significantly shorter than the shortest surface distance. In our experiments, the shortest surface distance is up to 9 times the Euclidean distance.

Motivated by the above observations, we propose another method to find the lower bound of the shortest surface distance that is tight and computationally-cheap. The major feature of this method is to "kill two birds with one stone." Specifically, whenever we want to find the lower and the upper bounds of the shortest surface distance, we only need to find the shortest network distance in the Delaunay graph. This distance serves as an upper bound, as discussed previously; and when multiplied by a constant factor derived from the terrain, it also serves as a lower bound.

In our experiments, this lower bound is generally tighter than the previous lower bound. For example, our lower bound is upto 2.8 times larger than the previous lower bound for the largest dataset size. Importantly, this lower bound method does not introduce a significant overhead of finding both the upper and the lower bounds of the shortest surface distance. It thus does not adversely affect the overall performance of the algorithms that use lower and upper bounds.

Although computing shortest network distances on the Delaunay graph is fast, it can be attractive to compute lower and upper bounds even more quickly. This can be achieved by sacrificing the tightness of the bounds. Thus, we propose an algorithm to generate a smaller graph G' from the Delaunay graph G so that the bounds can be computed faster on G'.

A key challenge is to generate G' so that the bounds do not become overly loose. With this in mind, we introduce an input parameter ω that controls the looseness of the bounds. When ω is set to its minimum value 1, G' yields the same bounds as the original graph G. If ω is set to a larger value, the tightness of the bounds is sacrificed. However, we ensure that lower bound $\widetilde{L}$ calculated on G' is not significantly smaller than the lower bound L calculated on G; and we ensure that the upper bound $\widetilde{U}$ calculated on G' is not significantly larger than the upper bound U calculated on G. Specifically, we maintain the following two inequalities: $\frac{1}{\omega} \cdot L \leq \widetilde{L} \leq L$ and $U \leq \widetilde{U} \leq \omega \cdot U$. Note that the inequality for $\widetilde{L}$ is different from the inequality for $\widetilde{U}$ because a looser lower bound corresponds to a smaller lower bound and a looser upper bound corresponds to a larger upper bound.

Our contributions can be summarized as follows. First, to the best of our knowledge, we are the first to extensively study the improvement of the lower bound in shortest surface distance computations, an important component in many spatial queries. Second, we propose to use the network distance for both upper and lower bounds, which yields new tighter bounds without introducing significant computational overhead. Third, we study how the tightened bounds can be incorporated in standard as well as recent, complex algorithms in order to speed up spatial queries. Fourth, we propose an approach to generate smaller graphs that yield faster lower and upper bound computations while providing guarantees for the tightness of the bounds. Fifth, we present a comprehensive empirical study that offers insight into the accuracy, efficiency, and scalability properties of the framework.

The remainder of the paper is organized as follows. Section 2 formulates the problem. Section 3 discusses the lower and upper bounds. Section 4 describes related work. Section 5 outlines our proposed algorithm for generating smaller graphs. Section 6 covers the empirical study of the proposed algorithms and framework. Finally, Section 7 concludes the paper.

2. PRELIMINARIES

Consider a three-dimensional space. Each point p is represented by an x-coordinate, a y-coordinate, and a z-coordinate. Usually, the z-coordinate of a point is said to be the *elevation* of this point.

A *terrain* is the graph of a continuous function that assigns every point on a horizontal plane to an elevation. In the literature, a terrain is typically represented by a *triangulated irregular network* (TIN) model that consists of a set $\mathcal{T}$ of faces each of which is represented by a triangle. Each triangle has three corners called *vertices* and three edges, each connecting two of its corners. Each vertex is a point in the three-dimensional space. We assume that each interior angle of every triangle/face is non-zero. To ensure this, we replace any triangle violating this assumption by a single edge. Figure 1 shows an example of a terrain. Two distinct triangles are said to be *adjacent* if they share an edge e. In this model, there are two types of triangles in $\mathcal{T}$, namely *normal triangles* and *boundary triangles*. Each normal triangle is adjacent to three other triangles in $\mathcal{T}$, and each boundary triangle is adjacent to one or two other triangles in $\mathcal{T}$. A point p is said to be on the terrain if there exists a triangle in $\mathcal{T}$ such that p is on the plane containing this triangle and p is inside this triangle.

We denote V and E to be the set of all vertices and the set of all edges in the model. The *Delaunay graph* G of the terrain is defined to be a weighted graph where the set of vertices and the set of edges in this graph are V and E, respectively, and the weight of each edge is the Euclidean distance between the two end-points of

the edge. In the following, for brevity, we simply write "graph" for "Delaunay graph."

We denote the line segment connection between two points p and p' to be (p, p'). We define the length of line segment (p, p'), denoted by $|(p, p')|$, to be the Euclidean distance between p and p'.

Given two vertices s and t in V, a *surface path* from s to t, denoted by $\Pi(s,t)$, is a sequence $\langle p_1, p_2, ..., p_n\rangle$ where (1) $p_1 = s$, (2) $p_n = t$, and (3) each p_i is a point along an edge in E. The surface path is thus composed of $n-1$ line segments, (p_1, p_2), (p_2, p_3), ..., (p_{n-1}, p_n). Figure 1 shows an example of a surface path P from s to t. The *length* of a surface path $\Pi(s,t)$, denoted by $|\Pi(s,t)|$, is defined to be $\sum_{i=1}^{n-1} |(p_i, p_{i+1})|$. The *shortest surface path* from s to t is the surface path from s to t with the smallest length.

Further, a *network path* from s to t, denoted by $\Pi_G(s,t)$, is represented by a sequence $\langle v_1, v_2, ..., v_n\rangle$ where (1) $v_1 = s$, (2) $v_n = t$, (3) each v_i is a vertex in V, and (4) each (v_i, v_{i+1}) is an edge in E. Figure 2 shows an example of a network path from s to t. The *length* of a network path $\Pi_G(s,t)$, denoted by $|\Pi_G(s,t)|$, is defined to be $\sum_{i=1}^{n-1} |(v_i, v_{i+1})|$. Finally, the *shortest network path* from s to t is defined to be the network path from s to t with the smallest length.

From the literature, we have the following lemma.

LEMMA 1 ([5, 10, 12]). *For any two vertices v and v' in V, $|\Pi(v,v')| \le |\Pi_G(v,v')|$.*

3. UPPER AND LOWER BOUNDS

In this section, we give the theoretical property that the shortest network distance can be used as the lower bound and the upper bound of the shortest surface distance.

LEMMA 2 (DISTANCE BOUND). *Let $\Pi(s,t)$ and $\Pi_G(s,t)$ be the shortest surface path and the shortest network path between source s and destination t on terrain $\mathcal{P}$, respectively. Then,*

$$\lambda \cdot |\Pi_G(s,t)| \le |\Pi(s,t)| \le |\Pi_G(s,t)|,$$

where $\lambda = \min\{\frac{\sin\theta_m}{2}, \sin\theta_m \cos\theta_m\}$ and θ_m is the minimum interior angle of a triangle in the terrain.

Proof: We need to show two inequalities: $\lambda \cdot |\Pi_G(s,t)| \le |\Pi(s,t)|$ and $|\Pi(s,t)| \le |\Pi_G(s,t)|$. The second inequality is derived from Lemma 1. In the following, we focus on showing the correctness of the first inequality.

Suppose that $\Pi(s,t)$ is a sequence $< p_1, p_2, ..., p_n >$, where p_i is a point along an edge. Note that $p_1 = s$ and $p_n = t$. Each line connecting p_i and p_{i+1} is on face f_i.

In the following, we define that each point p_i has its *owner*, denoted by o_i, which is one of the corners of the face containing p_i. Specifically, we set $o_1 = s$ and $o_n = t$. Consider a point p_i along an edge e of a face f where $i \in [2, n-1]$. (If p_i is a vertex so that multiple edges contain p_i, then we arbitrarily pick one of the edges as e.) Let A and B be the two end-points of the edge e. Note that A and B are the two vertices on the terrain. If $|(A, p_i)| \neq |(B, p_i)|$, we set o_i to be the end-point with the smaller Euclidean distance to p_i. Otherwise, we set o_i to the end-point of e shared with the edge that p_{i-1} is along.

Consider a sequence $O : \langle o_1, o_2, ..., o_n\rangle$. Each o_i is a vertex on the terrain. In addition, for each $i \in [2, n]$, we know that either (i) o_i is equal to o_{i-1} or (ii) (o_i, o_{i-1}) is an edge of a face on the terrain. We deduce that sequence O forms a network path from s to t since $o_1 = s$ and $o_n = t$. Let $|O|$ be the distance of O. Since $\Pi_G(s,t)$ is the shortest network path from s to t, we have that $|\Pi_G(s,t)| \le |O|$. In the remaining part of the proof, we show that $\lambda \cdot |O| \le |\Pi(s,t)|$. With these two inequalities, we derive that $\lambda \cdot |\Pi_G(s,t)| \le |\Pi(s,t)|$, which completes the proof.

To show that $\lambda \cdot |O| \le |\Pi(s,t)|$, consider a pair (o_i, o_{i+1}) on a single face f_i. We want to show that $\lambda \cdot |(o_i, o_{i+1})| \le |(p_i, p_{i+1})|$ for each $i \in [1, n-1]$.

Consider two cases. *Case 1:* $o_i = o_{i+1}$. In this case, illustrated in Figure 4, we know that $|(o_i, o_{i+1})| = 0$. Thus, the inequality holds.

Case 2: $o_i \neq o_{i+1}$. Face f_i has two corners/vertices, o_i and o_{i+1}. Let F be the remaining corner of f_i. Let C be the mid-point of edge (o_i, F), let D be the mid-point of edge (o_{i+1}, F), and let E be the mid-point of edge (o_i, o_{i+1}). We denote the interior angles of f_i at vertices o_i, o_{i+1}, and F by α, β, and γ, respectively.

We further consider four sub-cases. *Case 2(a):* Both p_i and p_{i+1} are not along edge (o_i, o_{i+1}). This case is illustrated in Figure 5(a). First, by the mid-point theorem, we have

$$|(C, D)| = \tfrac{1}{2} \cdot |(o_i, o_{i+1})|. \quad (1)$$

Consider four sub-cases. *Case (i):* $\alpha < \frac{\pi}{2}$ and $\beta < \frac{\pi}{2}$. In addition, p_i is on (o_i, C) only, and p_{i+1} is on (o_{i+1}, D) only because o_i is the owner of p_i and o_{i+1} is the owner of p_{i+1}. Thus, we know that $|(p_i, p_{i+1})| \ge |(C, D)|$. From Equation 1, we obtain $\frac{1}{2} \cdot |(o_i, o_{i+1})| \le |(p_i, p_{i+1})|$. Also, since $\sin\theta_m \cos\theta_m = \frac{\sin 2\theta_m}{2}$ and $\sin 2\theta_m \le 1$, we have $\sin\theta_m \cos\theta_m \le \frac{1}{2}$. Since $\sin\theta_m \cos\theta_m \le \frac{1}{2}$ and $\frac{\sin\theta_m}{2} \le \frac{1}{2}$, we have $\lambda \le \frac{1}{2}$. Thus, the inequality holds.

Case (ii): $\alpha \ge \frac{\pi}{2}$ and $\beta < \frac{\pi}{2}$ (illustrated in Figure 5(b)).

Since $\alpha \ge \frac{\pi}{2}$ and $\alpha + \beta + \gamma = \pi$, we have $\beta + \gamma < \frac{\pi}{2}$. Since $\beta \ge \theta_m$ and $\gamma \ge \theta_m$, we have $\beta + \gamma \ge 2\theta_m$. We derive that

$$2\theta_m \le \beta + \gamma < \tfrac{\pi}{2}. \quad (2)$$

We draw a perpendicular line from D to edge (o_i, F). Let G be the end-point of this line which is along edge (o_i, F). Similarly, p_i is on (o_i, C) only and p_{i+1} is on (o_{i+1}, D) only. Thus, we have

$$|(p_i, p_{i+1})| \ge |(D, G)|. \quad (3)$$

Consider the triangle with corners C, D, and G. By the mid-point theorem, we know that the line connecting C and D is parallel to edge (o_i, o_{i+1}). We deduce that the interior angle of this triangle at corner C is equal to $\pi - \alpha$. Note that $|(D, G)| = |(C, D)| \cdot \sin(\pi - \alpha)$. Since $\alpha + \beta + \gamma = \pi$, we have $|(D, G)| = |(C, D)| \cdot \sin(\beta + \gamma)$. From Inequality 2, we have $|(D, G)| \ge |(C, D)| \cdot \sin(\theta_m + \theta_m) = |(C, D)| \cdot \sin 2\theta_m$. Since $\sin 2\theta_m = 2 \sin\theta_m \cos\theta_m$, we have

$$|(D, G)| \quad \ge \quad 2 \cdot |(C, D)| \cdot \sin\theta_m \cdot \cos\theta_m \quad (4)$$

Thus, from Equation 1 and Inequalities 3 and 4, we obtain $\sin\theta_m \cdot \cos\theta_m \cdot |(o_i, o_{i+1})| \le |(p_i, p_{i+1})|$. Since $\lambda \le \sin\theta_m \cdot \cos\theta_m$, we have $\lambda \cdot |(o_i, o_{i+1})| \le |(p_i, p_{i+1})|$..

Case (iii): $\alpha < \frac{\pi}{2}$ and $\beta \ge \frac{\pi}{2}$. This case is illustrated in Figure 5(c) and is similar to Case (ii).

Case (iv): $\alpha \ge \frac{\pi}{2}$ and $\beta \ge \frac{\pi}{2}$. This case is impossible because the third angle must be greater than 0 and the sum must be equal to π. We conclude that the inequality holds for *Case 2(a)*.

Case 2(b): p_i is along edge (o_i, o_{i+1}) but p_{i+1} is not.

We consider four sub-cases. *Case (i):* $\beta < \frac{\pi}{2}$ and $\gamma < \frac{\pi}{2}$. This case is illustrated in Figure 6(a).

We draw a line from E to edge (F, o_{i+1}) such that this line is perpendicular to edge (F, o_{i+1}). Let G be the end-point of this line which is along edge (F, o_{i+1}). Since p_i is on (o_i, E) only and p_{i+1} is on (o_{i+1}, D) only, we have $|(p_i, p_{i+1})| \ge |(E, G)|$. Consider the triangle with corners E, G, and o_{i+1}. We have

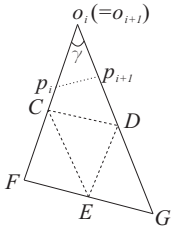
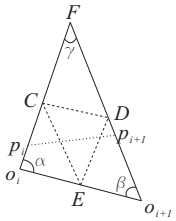
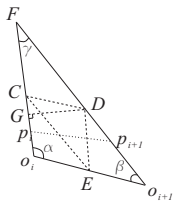


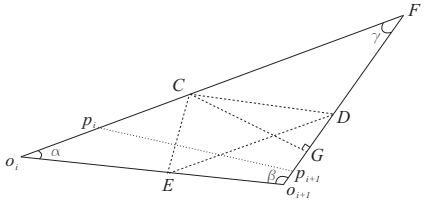
Figure 4: Case 1 in the Proof of Lemma 2



(a) Case 2(a)(i)

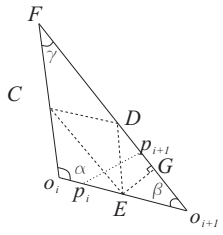


(b) Case 2(a)(ii)

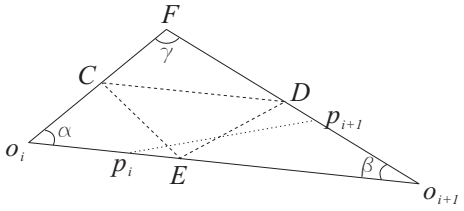


(c) Case 2(a)(iii)

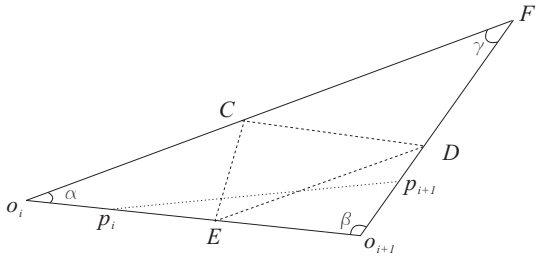
Figure 5: Case 2(a) in the Proof of Lemma 2



(a) Case 2(b)(i)



(b) Case 2(b)(ii)



(c) Case 2(b)(iii)

Figure 6: Case 2(b) in the Proof of Lemma 2

$|(E, o_{i+1})| \sin\beta = |(E, G)|$. Since $\frac{1}{2} \cdot |(o_i, o_{i+1})| = |(E, o_{i+1})|$, we have $\frac{1}{2} \cdot |(o_i, o_{i+1})| \sin\beta = |(E, G)|$. We deduce that $\frac{\sin\beta}{2} \cdot |(o_i, o_{i+1})| \leq |(p_i, p_{i+1})|$. Since $\theta_m \leq \beta < \frac{\pi}{2}$, we have $\frac{\sin\theta_m}{2} \cdot |(o_i, o_{i+1})| \leq |(p_i, p_{i+1})|$. Since $\lambda \leq \frac{\sin\theta_m}{2}$, the inequality holds.

Case (ii): $\beta < \frac{\pi}{2}$ and $\gamma \geq \frac{\pi}{2}$. This case is illustrated in Figure 6(b).

By the mid-point theorem, we have $|(D, E)| = \frac{1}{2} \cdot |(o_i, F)|$. By the sine rule, we have $\frac{\sin\beta}{|(o_i,F)|} = \frac{\sin\gamma}{|(o_i,o_{i+1})|}$. Thus, we derive $|(o_i, o_{i+1})| = \frac{\sin\gamma}{\sin\beta} \cdot |(o_i, F)|$. Since $\sin\gamma \leq 1$, we have $|(o_i, o_{i+1})| \leq \frac{1}{\sin\beta} \cdot |(o_i, F)|$. We deduce that $\frac{\sin\beta}{2} \cdot |(o_i, o_{i+1})| \leq |(D, E)|$. Since $\theta_m \leq \beta < \frac{\pi}{2}$, we have $\frac{\sin\theta_m}{2} \cdot |(o_i, o_{i+1})| \leq |(D, E)|$. In this case, since p_i is on (o_i, E) only and p_{i+1} is on (o_{i+1}, D) only, we know that $|(p_i, p_{i+1})| \geq |(D, E)|$. We derive that $\frac{\sin\theta_m}{2} \cdot |(o_i, o_{i+1})| \leq |(v_i, p_{i+1})|$. Since $\lambda \leq \frac{\sin\theta_m}{2}$, we have $\lambda \cdot |(o_i, o_{i+1})| \leq |(p_i, p_{i+1})|$.

Case (iii): $\beta \geq \frac{\pi}{2}$ and $\gamma < \frac{\pi}{2}$. This case is illustrated in Figure 6(c). Since p_i is on (o_i, E) only and p_{i+1} is on (o_{i+1}, D) only, we have $|(p_i, p_{i+1})| \geq |(E, o_{i+1})|$. Since $|(E, o_{i+1})| = \frac{1}{2} \cdot |(o_i, o_{i+1})|$, we have $\frac{1}{2} \cdot |(o_i, o_{i+1})| \leq |(p_i, p_{i+1})|$. Besides, since $\sin\theta_m \cos\theta_m = \frac{\sin 2\theta_m}{2}$ and $\sin 2\theta_m \leq 1$, we have $\sin\theta_m \cos\theta_m \leq \frac{1}{2}$. Since $\sin\theta_m \cos\theta_m \leq \frac{1}{2}$ and $\frac{\sin\theta_m}{2} \leq \frac{1}{2}$, we have $\lambda \leq \frac{1}{2}$. We derive that $\lambda \cdot |(o_i, o_{i+1})| \leq |(p_i, p_{i+1})|$.

Case (iv): $\beta \geq \frac{\pi}{2}$ and $\gamma \geq \frac{\pi}{2}$. As *Case (iv)* of *Case 2(a)*, this case is impossible.

Case 2(c): p_{i+1} is along edge (o_i, o_{i+1}) but p_i is not. This case is similar to Case 2(b).

Case 2(d): Both p_i and p_{i+1} are along edge (o_i, o_{i+1}). In this case, we know that $o_i = p_i$ and $o_{i+1} = p_{i+1}$. This can be explained by the observation that the shortest surface path on a three-dimensional terrain corresponds to the *straight* line from s to t on the two-dimensional plane on which all faces involved in the shortest surface path are *unfolded* [3]. We show this statement by contradiction. Suppose that both p_i and p_{i+1} are not the end-points of edge (o_i, o_{i+1}). Then, two points p_a and p_b exist where $1 \leq a < i$ and $i + 1 < b \leq n$ such that p_a and p_b are o_i and o_{i+1}, respectively. For each $j \in [a, b]$, p_j is along a line segment from p_a to p_b (or edge (o_i, o_{i+1})). This means that the path $< p_a, p_{a+1}, ..., p_b >$ corresponds to edge (o_i, o_{i+1}). This leads to a contradiction of the observation in [3], this path can simply be represented as $< p_a, p_b >$, where b is equal to $a + 1$. The above proof by contradiction also applies when *only one* of p_i and p_{i+1} is not the end-point of edge (o_i, o_{i+1}).)

Thus, $|(o_i, o_{i+1})| = |(p_i, p_{i+1})|$.

Since $\lambda \cdot |(o_i, o_{i+1})| \leq |(p_i, p_{i+1})|$ for each $i \in [1, n-1]$, we deduce that $\lambda \cdot \sum_{i=1}^{n-1} |(o_i, o_{i+1})| \leq \sum_{i=1}^{n-1} |(p_i, p_{i+1})|$. Thus, we conclude that $\lambda \cdot |O| \leq |\Pi(s, t)|$. □

According to the above lemma, $\lambda \cdot |\Pi_G(s, t)|$ is a lower bound and $|\Pi_G(s, t)|$ is an upper bound, where $\lambda = \min\{\frac{\sin\theta_m}{2}, \sin\theta_m \cos\theta_m\}$. Here, λ depends on θ_m that can be obtained from the terrain. According to our experimental results, the minimum interior angle of a triangle on the surface of the whole terrain is at least 45^o in all terrain datasets we used. Thus, λ is at least 0.35. In particular, in some cases, the lower bound computed above is 2.8 times larger than the existing lower bound, which is the Euclidean distance between two points. This suggests that this lower bound is better than the existing lower bound in these cases.

In practice, the interior angle of a triangle cannot be too small based on the concept of a *realistic terrain* studied extensively in the computational geometry community [2]. A *realistic terrain* is represented by a TIN model, which considers important parameters like the slope of each face and the minimum interior angle of a triangle in the TIN. Thus, we can find a TIN model such that the interior angle of a triangle can be maximized [2].

In some cases, the existing lower bound exceeds the above lower bound. In one scenario, the shortest surface path is exactly the same as the shortest network path. Thus, we use the maximum of the existing lower bound and the above lower bound as the final lower bound. Note that there is no significant additional overhead in computing the above lower bound since the shortest network distance is already available from the upper bound computation.

4. RELATED WORK

We review existing approaches using lower and upper bounds.

4.1 Multi-Resolution Range Ranking Method

Deng et al. [5] propose a multi-resolution range ranking (MR3) method for a surface k-NN query. This method uses lower and

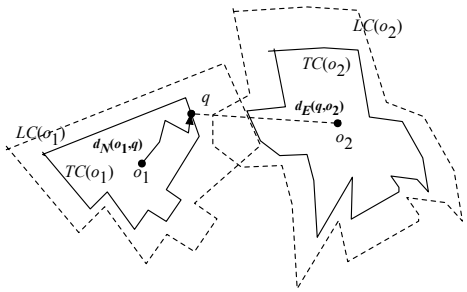


Figure 7: Tight Cells and Loose Cells of Objects o_1 and o_2

upper bounds for pruning some objects that are far away from the query point q.

Specifically, it involves the following four steps. Let H be the (conceptual) horizontal plane at the sea level of the terrain. Let O be a set of objects in the dataset for the surface k-NN query.

Step 1 (2D k-NN Query): Let q' be the query point projected on H. Let O' be the set of objects in O projected on H. The MR3 method finds a set S' of k objects in O' nearest to q' on H. Let S be the set of all objects whose projections are in S'.

Step 2 (Surface Distance Computation): It finds the k-th nearest object o in S from q according to their surface distances.

Step 3 (2D Range Query): It computes the upper bound U of the shortest surface distance between q and o. It performs a range query from q' with its radius equal to U and obtain a set T' of objects from the range query. Let T be the set of all objects whose projections are in T'.

Step 4 (Surface Distance Ranking): It finds the k objects in T whose surface distances are at most the surface distance between q and the k-th nearest object in T.

Since Step 4 finds surface distances, the MR3 method makes use of the upper and lower bounds for pruning. Even though it is equipped with the lower and upper bound computations for pruning, it is found in [10, 12] that the MR3 method does not return accurate k-NN results, especially as k gets larger.

4.2 Voronoi Diagram Based Method

Shahabi et al. [10] propose a Voronoi diagram-based approach for computing a surface k-NN query, which is the state-of-the-art method for this query. Unlike the MR3 method, it returns accurate results. This approach exploits so-called *tight cells* and *loose cells*. Let O be a set of objects on a surface.

Given an object $o \in O$, the *tight cell* of o, $TC(o)$, is the region on the surface such that each point p in this region has a network distance to o that is at most the Euclidean distance between p and each other object in O. In Figure 7 showing two objects o_1 and o_2, the region enclosed by the solid line containing o_1 corresponds to $TC(o_1)$ and the region enclosed by the solid line containing o_2 corresponds to $TC(o_2)$. Consider a point q along the boundary of $TC(o_1)$. Note that the network distance between q and o_1 ($d_N(o_1, q)$) is equal to the Euclidean distance between q and o_2 ($d_E(q, o_2)$). Given an object $o \in O$, the *loose cell* of o, $LC(o)$, is the region on the surface such that each point p in this region has a Euclidean distance to o that is at most the network distance between p and each other object in O. In Figure 7, the region enclosed by the dashed line containing o_1 corresponds to $LC(o_1)$ and the region enclosed by the dashed line containing o_2 corresponds to $LC(o_2)$.

[10] found the following three properties related to tight cells and loose cells. (1) It is possible that the loose cell of an object can overlap with the loose cell of another object. (2) The loose cell of an object does not overlap with the tight cell of another object. (3) The tight cell of an object is completely inside its loose cell.

If a query point q is inside the tight cell of an object o, we know that there is only one such tight cell containing q. Thus, o is the nearest neighbor of q according to the surface distance. Otherwise, q is inside a number of loose cells of objects, but is outside the tight cells of these objects. Let O' be these objects. We are not sure which object in O' is nearest to q, and we need to run the algorithm for finding the shortest surface distance $|O'|$ times to find the answer. According to the above observation, if the area of the tight cell is larger, then it will be more likely that q is in a tight cell and we do not need to issue the time-consuming algorithm for finding the shortest surface distance. Besides, if the area of the loose cell is smaller then it will be more likely that the size of O' is smaller, and thus we will run the algorithm for finding the shortest surface distance fewer times.

Shahabi et al. [10] propose an algorithm that first finds the tight cells of all objects and the loose cells of all objects. Then it finds the k nearest neighbors on the surface from a query point q according to these cells. Specifically, the algorithm first finds all tight and loose cells covering q in order to determine the 1-NN o. Then, it expands all the *neighbors* of the loose cell of o to incrementally find the next nearest neighbor from q. During the expansion, it finds the shortest surface distance between q and each object accessed.

Interestingly, the above algorithm can be improved significantly when our bounds are used. Specifically, as we described before, the lower bound we proposed is at least the Euclidean distance studied by [10]. We modify the tight cell definition and the loose cell definition by replacing the Euclidean distance calculation by our lower bound. Since our lower bound is at least the Euclidean distance, it is easy to verify that the area of the tight cell of each object is larger and the area of the loose cell of each object is smaller. As we explained, a larger tight cell and a smaller loose cell can avoid a lot of shortest surface path computation and thus the algorithm can be speeded up.

4.3 Other Existing Approaches

Other approaches exist that use lower and upper bounds. Examples include continuous surface k-NN queries [11] and reverse nearest neighbor queries [12].

Xing et al. [11] propose an algorithm for a continuous surface k-NN query that relies on the concept of the *expansion area* of a query point q. Specifically, given a query point q, the expansion area of q is a region containing all objects such that the Euclidean distance between each object in this region and q is at most a threshold being updated during the execution of the algorithm. The algorithm considers all objects in the expansion area of q only and calculates the shortest surface distance between q and each of these objects. Similarly, when our bounds are used, we can replace the Euclidean distance by our lower bound. In this case, fewer objects remain in the expansion area. Thus, the algorithm can be speeded up.

Yan et al. [12] propose algorithms to find bichromatic and monochromatic reverse nearest neighbors on terrains. The algorithms proposed by [12], the only best-known algorithm for these queries, rely on the tight and loose cells proposed by [10]. Similarly, with our new bound adoption, it is expected that these algorithms can run more quickly.

5. GENERATING A SMALLER GRAPH

As we described in Section 1, in order to efficiently compute the network distance (which is used in our upper/lower bound computation), we propose to generate a *smaller* graph G' from G. We introduce a parameter ω such that the bounds computed from G'

are not quite different from the bounds computed from G. We give a formal definition for ω as follows. Let $L_G(s,t)$ and $U_G(s,t)$ be the lower and upper bounds of a surface distance from a source s to a destination t computed based on a graph G, respectively.

PROPERTY 1 (NETWORK DISTANCE PROPERTY). *Given an original graph $G = (V, E)$ and another graph $G' = (V', E')$, we say that G' satisfies the* network distance property *if and only if for any two vertices s and t in V, the following two inequalities are satisfied.*

$$\frac{1}{\omega} \cdot L_G(s,t) \leq L_{G'}(s,t) \leq L_G(s,t) \quad (5)$$

$$U_G(s,t) \leq U_{G'}(s,t) \leq \omega \cdot U_G(s,t) \quad (6)$$

As described in Section 1, we would like to maintain Property 1. The major idea behind generating a smaller graph G' from G is to remove some vertices from G and re-adjust the edges in G such that G' satisfies the network distance property.

5.1 Generating G'

We propose an iterative approach that removes a vertex v iteratively from G to generate G'. Consider an iteration of this approach. Before we execute this iteration, we have a graph G' where some vertices have been removed. After the execution, we remove a vertex v from G' and form a smaller graph G''. Let V' and E' be the set of vertices and the set of edges in G', respectively. Let $V'' = V' - \{v\}$ and E'' be the vertices and edges in G'', respectively. We denote the operation of removing vertex v from G' by $o(G', v)$; thus $G'' = o(G', v)$.

Algorithm 1 shows the algorithm for generating G'. In this algorithm, we have to check whether G' satisfies the network distance property (Property 1).

Algorithm 1 Algorithm for Generating G'

```
1: G' ← G
2: while there exists a vertex v in G' such that a graph G'' obtained after the removal operation of v from G' satisfies the network distance property do
3:     G'' ← o(G', v)
4:     G' ← G''
5: return G'
```

In Property 1, we have to maintain Inequalities 5 and 6. Note that we detailed how to compute $L_G(s,t)$ and $U_G(s,t)$ (based on G) in Section 3. That is,

$$L_G(s,t) = \lambda \cdot |\Pi_G(s,t)|$$
$$U_G(s,t) = |\Pi_G(s,t)|.$$

However, how to compute $L_{G'}(s,t)$ and $U_{G'}(s,t)$ (based on G') has not been specified yet. Next, we describe a method to compute $L_{G'}(s,t)$ and $U_{G'}(s,t)$. Formally, given distinct vertices s and t in V (from the original graph G), we denote the *estimated distance* between s and t on a smaller graph G' by $\widetilde{d}_{G'}(s,t)$. In Section 5.1.1, we describe a method to compute this estimated distance such that it satisfies the following property.

PROPERTY 2 (ESTIMATED DISTANCE PROPERTY). *Given an original graph G and another graph G', we say that G' satisfies the* estimated distance property *if and only if for any two vertices s and t in V,*

$$|\Pi_G(s,t)| \leq \widetilde{d}_{G'}(s,t) \leq \omega \cdot |\Pi_G(s,t)|. \quad (7)$$

After we describe how to compute $\widetilde{d}_{G'}(s,t)$, we will argue in Section 5.2 that the above property holds.

We define $L_{G'}(s,t)$ and $U_{G'}(s,t)$ as follows.

$$L_{G'}(s,t) = \frac{\lambda}{\omega} \cdot \widetilde{d}_{G'}(s,t)$$
$$U_{G'}(s,t) = \widetilde{d}_{G'}(s,t)$$

Now, given the concept of $\widetilde{d}_{G'}(s,t)$, we know how to compute $L_{G'}(s,t)$ and $U_{G'}(s,t)$.

LEMMA 3 (PROPERTY RELATIONSHIP). *Given a graph G', if G' satisfies the estimated distance property, then G' satisfies the network distance property.*

Proof: First, we show that Inequality 6 holds. Since G' satisfies the estimated distance property, for any two vertices s and t in V, $|\Pi_G(s,t)| \leq \widetilde{d}_{G'}(s,t) \leq \omega \cdot |\Pi_G(s,t)|$. Since $U_G(s,t) = |\Pi_G(s,t)|$ and $U_{G'}(s,t) = \widetilde{d}_{G'}(s,t)$, we derive that $U_G(s,t) \leq U_{G'}(s,t) \leq \omega \cdot U_G(s,t)$. Thus, Inequality 6 holds.

Next, we show that Inequality 5 also holds. We first show that $L_{G'}(s,t) \leq L_G(s,t)$. Since $L_{G'}(s,t) = \frac{\lambda}{\omega} \cdot \widetilde{d}_{G'}(s,t)$ and $\widetilde{d}_{G'}(s,t) \leq \omega \cdot |\Pi_G(s,t)|$, we derive that $L_{G'}(s,t) \leq \lambda \cdot |\Pi_G(s,t)|$. Since $L_G(s,t) = \lambda \cdot |\Pi_G(s,t)|$, we conclude that $L_{G'}(s,t) \leq L_G(s,t)$.

In the following, we show that $\frac{1}{\omega} \cdot L_G(s,t) \leq L_{G'}(s,t)$. Note that $L_{G'}(s,t) = \frac{\lambda}{\omega} \cdot \widetilde{d}_{G'}(s,t) \geq \frac{\lambda}{\omega} \cdot |\Pi_G(s,t)|$. Since $L_G(s,t) = \lambda \cdot |\Pi_G(s,t)|$, we have $L_{G'}(s,t) \geq \frac{1}{\omega} \cdot L_G(s,t)$.

Thus, Inequality 5 also holds. In conclusion, G' satisfies the network distance property. □

With the above lemma, if G' satisfies the estimated distance property (Property 2), then G' satisfies the network distance property (Property 1). In the following, we focus on checking whether G' satisfies the estimated distance property.

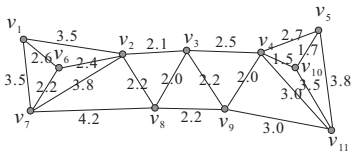
There are four remaining issues in Algorithm 1 described in Sections 5.1.1 – 5.1.4.

5.1.1 How to Compute $\widetilde{d}_{G'}(s,t)$

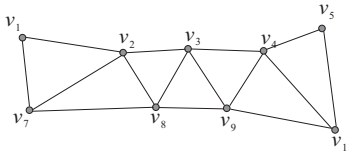
As the first step in describing how to compute $\widetilde{d}_{G'}(s,t)$, we define the concept of "guest" and "host" vertices by drawing an analogy to a familiar occurrence where *guests* leave, while *hosts* remain at a residence.

Consider graph G'. Let $\overline{V'} = V - V'$, where $\overline{V'}$ and V' denote the set of *removed* and *remaining* vertices, respectively. Each removed vertex $\overline{v}$ in $\overline{V'}$ is associated with a non-empty set of some remaining vertices in V'. Each (remaining) vertex in this set is called a *host* of $\overline{v}$ and the set of hosts is denoted by $\mathcal{H}(\overline{v})$. If v is a host of a removed vertex $\overline{v}$, then $\overline{v}$ is said to be a *guest* of v. Note that each guest is in the *removed* vertex set ($\overline{V'}$) and each host is in the remaining vertex set V'. Given a remaining vertex $v \in V'$, we define the *guest set* of v, denoted by $\mathcal{G}(v)$, to be the set of all guests of v. Note that a vertex $\overline{v} \in \overline{V'}$ is associated with at least one host and can be associated with multiple hosts, and a vertex $v \in V'$ can be associated with no guest or multiple guests.

Given a vertex $v \in V'$, we maintain not only the guest set of v, $\mathcal{G}(v)$, but also the information about the estimated distance between v and each $\overline{v}$ of its guests in $\mathcal{G}(v)$ (i.e., $\widetilde{d}_{G'}(\overline{v}, v)$)). Specifically, we define the *guest information set* of v, denoted by $\mathcal{GL}(v)$, to be the set of entries where each entry is in the form of $(\overline{v}, \widetilde{d}_{G'}(\overline{v}, v))$ for each $\overline{v} \in \mathcal{G}(v)$. Here, each entry contains the second component in the form of $\widetilde{d}_{G'}(\overline{v}, v)$ which can be computed when we create or update this entry. Details will be described later



(a) G



(b) G'

v	$\mathcal{G}(v)$	$\mathcal{GL}(v)$		$\mathcal{G}(v)$	$\mathcal{GL}(v)$
v_1	$\{v_6\}$	$\{(v_6, 2.6)\}$	v_7	$\{v_6\}$	$\{(v_6, 2.2)\}$
v_2	$\{v_6\}$	$\{(v_6, 2.4)\}$	v_8	ϕ	ϕ
v_3	ϕ	ϕ	v_9	ϕ	ϕ
v_4	$\{v_{10}\}$	$\{(v_{10}, 1.5)\}$	v_{11}	$\{v_{10}\}$	$\{(v_{10}, 3.5)\}$
v_5	$\{v_{10}\}$	$\{(v_{10}, 1.7)\}$			

(c) $\mathcal{G}(\cdot)$ and $\mathcal{GL}(\cdot)$

v	$\mathcal{H}(v)$
v_6	$\{v_1, v_2, v_7\}$
v_{10}	$\{v_4, v_5, v_{11}\}$

(d) $\mathcal{H}(\cdot)$

Figure 8: An Example Illustrating Guests and Hosts

in Section 5.1.2. At this moment, we assume that this component is given.

EXAMPLE 1 (HOST AND GUEST). Figure 8(a) shows the original graph G. Suppose that we remove vertex v_6 and vertex v_{10} from G, and a smaller graph G' is generated as shown in Figure 8(b). According to some methods which will be described later, we generate the guest set and the guest information set of each remaining vertex in G', as shown in Figure 8(c). Similarly, the host set of each vertex removed can be found in Figure 8(d). □

With the concepts of "guest" and "host," we are ready to describe how we compute $\widetilde{d}_{G'}(s,t)$.

Given any two vertices s and $t \in V$, we want to *estimate* the network distance between s and t on the smaller graph $G' = (V', E')$. Since some vertices are removed from V and cannot be found in V', s may occur in V' or not, and t may also occur in V' or not.

This yields three cases to be considered when we compute the shortest network distance between s and t. *Case 1:* Both s and t are not found in V', *Case 2:* Only one of s and t is found in V', and *Case 3:* Both s and t are found in V'.

Consider Case 1. Note that each vertex not in V' is associated with at least one host that can be found in V'. The major idea of estimating the distance between s and t involves three components, namely the *guest-to-host distance*, the *host-to-host distance*, and the *host-to-guest distance*.

We first estimate the distance between s which is not in V' and one of its hosts, says u. We call this distance the *guest-to-host distance* from s to u. Second, we estimate the distance between this host and one of the hosts of t, says u'. We call this distance the *host-to-host distance* from u to u'. Third, we estimate the distance between u' and t. We call this distance the *host-to-guest distance* from u' to t.

Consider the guest-to-host distance. Given a host $u \in \mathcal{H}(s)$, we have $s \in \mathcal{G}(u)$ and there exists an entry $(s, \widetilde{d}) \in \mathcal{GL}(u)$. We define the *estimated guest-to-host distance* from s to u, denoted by $dist_1(s,u)$, to be $\widetilde{d}$, where $\widetilde{d}$ is the second component of the entry $(s, \widetilde{d}) \in \mathcal{GL}(u)$.

Consider the host-to-host distance. Given a vertex u and a vertex u', we define the *estimated host-to-host distance* from u to u', denoted by $dist_2(u,u')$, to be $|\Pi_{G'}(u,u')|$.

Consider the host-to-guest distance. Given a host $u' \in \mathcal{H}(t)$, similarly, we define the *estimated host-to-guest distance* from u' to t, denoted by $dist_3(u',t)$, to be $\widetilde{d}$ where $\widetilde{d}$ is the second component of the entry $(t, \widetilde{d}) \in \mathcal{GL}(u')$.

In the above discussion, we just consider a particular host u of s and a particular host u' of t. In general, there are multiple hosts of s and multiple hosts of t. Thus, the set of all possible pairs of hosts of s and hosts of t can be denoted by $\mathcal{H}(s) \times \mathcal{H}(t)$. Among all possible pairs, we want to find the *best host pair* yielding the smallest estimated distance. Formally, we define the *best host pair* of (s,t), denoted by $h_o(s,t)$, to be $\arg\min_{(u,u') \in \mathcal{H}(s) \times \mathcal{H}(t)} dist_1(s,u) + dist_2(u,u') + dist_3(u',t)$. If (u,u') is the best host pair of (s,t), we say that u is the best host of s and u' is the best host of t.

We define $\widetilde{d}_{G'}(s,t) = dist_1(s,u) + dist_2(u,u') + dist_3(u',t)$, where $(u,u') = h_o(s,t)$.

Consider Cases 2 and 3, which are simpler than Case 1. If s is a vertex in V', we set the guest-to-host distance to be 0. In this case, we define the best host of s to be itself. If t is a vertex in V', we set the host-to-guest distance to be 0. Then, we define the best host of t to be itself.

Now, we know how to compute $\widetilde{d}_{G'}(s,t)$. In Section 5.2, we will show that Property 2 (based on $\widetilde{d}_{G'}(s,t)$) holds.

EXAMPLE 2 (ESTIMATED DISTANCE). Consider Example 1 (as shown in Figure 8). Suppose that we want to find the estimated distance between v_6 and v_{10} (i.e., $\widetilde{d}_{G'}(v_6, v_{10})$). Note that vertex v_6 has three hosts in $\mathcal{H}(v_6)$, namely v_1, v_2, and v_7, and vertex v_{10} has three hosts in $\mathcal{H}(v_{10})$, namely v_4, v_5, and v_{11}.

Consider the host v_2 of v_6. There exists an entry $(v_6, 2.4)$ in $\mathcal{G}(v_2)$. The estimated guest-to-host distance is 2.4. Consider the host v_4 of v_{10}. There exists an entry $(v_{10}, 1.5)$ in $\mathcal{G}(v_4)$. The estimated host-to-guest distance is 1.5. In this case, the estimated host-to-host distance is equal to $|\Pi_{G'}(v_2, v_4)| = 2.1 + 2.5 = 4.6$.

All pairs of hosts of v_6 and hosts of v_{10} correspond to $\mathcal{H}(v_6) \times \mathcal{H}(v_{10})$. It is easy to verify that (v_2, v_4) is the best host pair of (v_6, v_{10}). Finally, the estimated distance between v_6 and v_{10} is equal to $2.4 + 4.6 + 1.5 = 8.5$. □

5.1.2 How to Perform $o(G', v)$

The next issue is the details of the operation $o(G', v)$. Algorithm 2 shows the algorithm for the operation $o(G', v)$. In this algorithm, line 1 corresponds to finding all *neighbors* of the vertex v to be removed. Given a vertex v and a vertex v' in V', v is a *neighbor* of v' if and only if $(v, v') \in E'$ (or $(v', v) \in E'$). Given a vertex v in V' (from G'), we denote the set of all neighbors of v to be $N_{G'}(v)$. In this algorithm, line 7 corresponds to a standard triangulation method in the TIN model which is a process that partitions a given polygon into a number of triangles. We adopted the Delaunay triangulation method [1].

Moreover, in this algorithm, we introduce a function called $GLInsert$ which takes the guest information list of a vertex v', $\mathcal{GL}(v')$, and an entry in the form of $(v, \widetilde{d})$ as inputs, and outputs the updated guest information list of v'. The function returns the updated guest information list according to the following cases. *Case 1:* There does not exist any entry $(v, \widetilde{d'}) \in \mathcal{GL}(v')$. In this case, the updated guest information list to be returned is set to $\mathcal{GL}(v') \cup \{(v, \widetilde{d})\}$. *Case 2:* There exists an entry $(v, \widetilde{d'}) \in \mathcal{GL}(v')$. In this case, we further consider two sub-cases. *Case 2(a):* $\widetilde{d'} \leq \widetilde{d}$. In this sub-case, the updated guest information list to be returned is set to $\mathcal{GL}(v')$. *Case 2(b):* $\widetilde{d'} > \widetilde{d}$. In this sub-case, the updated guest information list to be returned is set to $\{\mathcal{GL}(v') - \{(v, \widetilde{d'})\}\} \cup \{(v, \widetilde{d})\}$.

5.1.3 Checking Whether G' Satisfies Estimated Distance Property

Algorithm 2 Algorithm for Removing a Vertex v from G' (i.e., $o(G', v)$)

1: $\mathcal{N} \leftarrow N_{G'}(v)$
2: **for** each $v' \in \mathcal{N}$ **do**
3: $\quad \widetilde{D}_{v'} \leftarrow \widetilde{d}_{G'}(v, v')$
4: remove v from V'
5: remove all edges containing v from E'
6: $P \leftarrow$ a polygon formed from a set of all the remaining edges which are adjacent to any two vertices in $\mathcal{N}$
7: triangulate P
8: $\mathcal{H}(v) \leftarrow \mathcal{N}$
9: **for** each $v' \in \mathcal{N}$ **do**
10: $\quad \mathcal{G}(v') \leftarrow \mathcal{G}(v') \cup \{v\} \cup \mathcal{G}(v)$
11: $\quad \mathcal{GL}(v') \leftarrow GLInsert(\mathcal{GL}(v'), (v, \widetilde{D}_{v'}))$
12: $\quad$ **for** each $(\overline{v}, \widetilde{d}) \in \mathcal{GL}(v)$ **do**
13: $\quad\quad \mathcal{GL}(v') \leftarrow GLInsert(\mathcal{GL}(v'), (\overline{v}, \widetilde{d} + \widetilde{D}_{v'}))$
14: $\quad\quad \mathcal{H}(\overline{v}) \leftarrow \mathcal{H}(\overline{v}) - \{v\} \cup \{v'\}$
15: **return** G'

The third issue is how to check whether G' satisfies the estimated distance property. Specifically, in Algorithm 1, whenever we remove a vertex v from V', we have to check whether the resulting graph after the vertex removal operation satisfies the estimated distance property (Property 2) (and thus the network distance property (Property 1)). This property requires that for *any* two vertices in V, Inequality 7 is satisfied. Checking this property naively is time-consuming. Fortunately, we just need to check two properties which are related to the *neighbors* of v only, instead of all vertices in V. In Section 5.2, we will show that if these two properties are satisfied, then Property 2 holds.

Before we introduce the two properties, let us give an intuition of these two properties. Consider an iteration of Algorithm 1. Just before this iteration, suppose that G' denotes the current graph with some vertices removed. Let v be the vertex to be removed in this iteration. It generates G'' which is equal to $o(G', v)$. Roughly speaking, we want to maintain two kinds of distance information stored in the graph after v is removed.

- The first kind of distance information is the *intra-distance information*. We want to make sure that the pairwise (estimated) distance between any two neighbors of v does not change too much after v is removed from the graph.
- The second kind of distance information is the *inter-distance information*. We want to make sure that the pairwise (estimated) distance between each neighbor of v and each guest of v does not change too much after v is removed from the graph.

The two properties are formally given as follows. The first property is called the *neighborhood error bound property* which is used to maintain the intra-distance information. The second property is called the *host-guest error bound property* which is used to maintain the inter-distance information.

PROPERTY 3 (NEIGHBORHOOD ERROR BOUND PROPERTY). *Let G' be a graph. Given a vertex v in V', v is said to satisfy the* neighborhood error bound property *in G' if and only if for any two vertices v_i and v_j in $N_{G'}(v)$,*

$$|\Pi_G(v_i, v_j)| \leq \widetilde{d}_{G''}(v_i, v_j) \leq \omega \cdot |\Pi_G(v_i, v_j)|,$$

where $G'' = o(G', v)$.

PROPERTY 4 (HOST-GUEST ERROR BOUND PROPERTY). *Let G' be a graph. Given a vertex v in V', v is said to satisfy the* host-guest error bound property *in G' if and only if for each vertex $\overline{v} \in \mathcal{G}(v)$ and each vertex $v' \in N_{G'}(v)$,*

$$|\Pi_G(v', \overline{v})| \leq \widetilde{d}_{G''}(v', \overline{v}) \leq \omega \cdot |\Pi_G(v', \overline{v})|,$$

where $G'' = o(G', v)$.

Given a graph G' and a vertex $v \in V$, v is said to satisfy the *removal property* in G' if and only if v satisfies both the neighborhood error property and the host-guest error property in G'.

With this removal property, in Algorithm 1, we change the checking condition in line 2 to that "there exists a vertex v in G' such that v satisfies the removal property in G'."

5.1.4 How to Find Vertex to be Removed

The last issue is how to find a vertex to be removed. As we described in the previous section, we need to find a vertex v in G' such that v satisfies the removal property. In our implementation, we find this vertex v by processing all vertices in a particular order based on the *regularity* of the polygon P formed from a set of all the remaining edges which are adjacent to any two vertices in $N_{G'}(v)$. We define a function $f(v)$ which takes a vertex v as an input and returns a non-negative real number as an output denoting how regular the shape of $N_{G'}(v)$ is. Specifically, we define $f(v)$ to be the average difference between an interior angle and the average interior angle within the polygon P. If $f(v)$ is smaller, then P is more regular. Here, we would like to choose a vertex v whose polygon is more regular for processing first. Triangulating the polygon can result in many triangles with more regular shapes. It is more likely that more triangles with more regular shapes increases the opportunity of simplifying the graph in the later process. This is because an irregular triangle containing one long side and one short side has *two extreme* scenarios for simplifying the graph, resulting in a lesser opportunity to remove vertices in this triangle.

5.2 Analysis

In this section, we show that the smaller graph G' satisfies the estimated distance property (Property 2).

LEMMA 4. *Let G' be the graph generated by Algorithm 1. Then, G' satisfies the estimated distance property.*

Proof Sketch: According to Property 2, we want to show that for any two vertices s and t in V, $|\Pi_G(s,t)| \leq \widetilde{d}_{G'}(s,t) \leq \omega \cdot |\Pi_G(s,t)|$.

In this proof, we focus on Case 1 mentioned in Section 5.1.1. Cases 2 and 3 can be shown similarly.

Let u be the best host of s and $\widetilde{d}_1$ be the corresponding guest-to-host distance Let u' be the best host of t and $\widetilde{d}_2$ be the corresponding host-to-guest distance. We have

$$\widetilde{d}_{G'}(s,t) = \widetilde{d}_1 + |\Pi_{G'}(u,u')| + \widetilde{d}_2. \tag{8}$$

By Property 3, we deduce that for any two remaining vertices v and v' in V', we have

$$|\Pi_G(v,v')| \leq \widetilde{d}_{G'}(v,v') \leq \omega \cdot |\Pi_G(v,v')|. \tag{9}$$

By Property 4, we deduce that for each vertex $v \in V'$ and each vertex $\overline{v} \in \mathcal{G}(v)$, we have

$$|\Pi_G(v,\overline{v})| \leq \widetilde{d}_{G'}(v,\overline{v}) \leq \omega \cdot |\Pi_G(v,\overline{v})|. \tag{10}$$

First, we show that $|\Pi_G(s,t)| \leq \widetilde{d}_{G'}(s,t)$. Since u and u' are in V', $\widetilde{d}_{G'}(u,u') = |\Pi_{G'}(u,u')|$. From Equation 9, we derive that $\widetilde{d}_{G'}(u,u') \geq |\Pi_G(u,u')|$ and thus $|\Pi_{G'}(u,u')| \geq |\Pi_G(u,u')|$.

Dataset Sizes D (points)	20K, **200K**, 400K, 800K, 1000K
User Parameter ω	1, **1.2**, 1.4, 1.6, 1.8, 2
k	**2**, 5, 10, 15, 20

Table 1: Parameter Settings

Since s and t are not in V', and u and u' are in V', from Equation 10, we derive that $\widetilde{d}_1 \geq |\Pi_G(s,u)|$ and $\widetilde{d}_2 \geq |\Pi_G(u',t)|$. From Equation 8, we derive that $\widetilde{d}_{G'}(s,t) \geq |\Pi_G(s,u)| + |\Pi_G(u,u')| + |\Pi_G(u',t)| \geq |\Pi_G(s,t)|$.

Second, we show that $\widetilde{d}_{G'}(s,t) \leq \omega \cdot |\Pi_G(s,t)|$. Consider path $\Pi_G(s,t)$. It is easy to verify that there exists a vertex $w \in \mathcal{H}(s)$ along this path and there exists a vertex $w' \in \mathcal{H}(t)$ along this path. We know that

$$|\Pi_G(s,t)| = |\Pi_G(s,w)| + |\Pi_G(w,w')| + |\Pi_G(w',t)|. \quad (11)$$

Consider the distance $\widetilde{D}$ from s to t in G' which is equal to the sum of the guest-to-host distance (i.e., $\widetilde{d}_{G'}(s,w)$), the shortest distance from w to w' and the host-to-guest distance (i.e., $\widetilde{d}_{G'}(w',t)$). Note that $\widetilde{D} = \widetilde{d}_{G'}(s,w) + |\Pi_{G'}(w,w')| + \widetilde{d}_{G'}(w',t)$. By Inequalities 9 and 10, we derive that $\widetilde{D} \leq \omega \cdot |\Pi_G(s,w)| + \omega \cdot |\Pi_G(w,w')| + \omega \cdot |\Pi_G(w',t)|$. Thus, from Equation 11, we obtain $\widetilde{D} \leq \omega \cdot |\Pi_G(s,t)|$. Since $\widetilde{D} \geq \widetilde{d}_{G'}(s,t)$, we have $\widetilde{d}_{G'}(s,t) \leq \omega \cdot |\Pi_G(s,t)|$. A detailed proof can be found in [13]. □

6. EXPERIMENTS

6.1 Experimental Setup

Data Sets and Parameter Settings: Experiments were conducted on the Eagle Peak (EP) dataset (http://data.geocomm.com/). This widely used dataset is from Wyoming, USA, covers an area of 10.7 x 14 km^2, and has 1.3 million data points [5, 8, 10–12]. We used sub-regions of varying sizes to obtain robust results.

The experiments were conducted by varying several parameters to study the effect of the trade-offs among accuracy, efficiency, and memory usage. Table 1 shows the parameters with their default values shown in bold. The default value of ω is set to 1.2, which means that there is a 20% error for generating a smaller graph. Experiments were conducted with default parameter values unless explicitly stated.

Implementation: The core algorithms were implemented in C and C++, and some auxiliary tasks were implemented in Perl. A terrain tool, developed by CMU, called Triangle (http:// www.cs.cmu.edu/ ∼quake/ triangle.html), was employed for generating the TIN model with a minimum interior angle quality. In all datasets, with this tool, θ_m generated is at least 45^o. The Chen-and-Han implementation [7] was used to compute shortest surface paths. All experiments were carried out on a Fedora 18 Linux machine with an Intel Xeon E5 CPU (20MB cache, hyper-threading, 8 cores) and 32 GB internal memory.

All experiments were conducted 100 times. Average values were reported in our final results. For each spatial query with a query location, following [10, 12], we generate a query location randomly and select 10% of the vertices in the TIN model randomly as objects.

Note that there are two contributions in this paper. The first contribution is the proposed *tighter bounds*, and the second contribution is the proposed *smaller graph*. In order to highlight the significance of our contributions, we study them both individually and combined. In Section 6.2, we study the effect of our *tighter bounds* based on the original graph. Section 6.3, shows how a compressed *smaller graph* affects existing bounds. In Section 6.4, we show how our bounds based on our smaller graph improve existing results. Finally, Section 6.5 depicts the scalability of our bound computation and algorithms using our bounds.

6.2 Our Bounds and the Original Graph

In Section 6.2.1, we compare our distance bounds with *existing* distance bounds based on an *original* graph. In Section 6.2.2, we study how the performance of some existing algorithms (described in Section 4) are improved when our bounds are used.

6.2.1 Distance Bound Comparison

Based on the original graph, on average our lower bound is 5,075 meters, while the existing lower bound is 2,671 meters only. The improvement ratio for the lower bound on EP is 1.9. Conducting the same experiment with different source and destination points, and also over different data set sizes (20K–1M vertices), we get an overall average improvement ratio of 2.8.

6.2.2 Impact on Existing Methods

We study how our bounds can be used for three popular spatial queries, namely (1) surface k-NN queries, (2) surface range queries, and (3) reverse surface NN queries.

(1) Surface k-NN Queries: The impact of our bounds for surface k-NN queries based on the original graph G are studied. We compared three algorithms, namely the straightforward approach (SF), the *MR3* approach (*MR3*) [5] and the Voronoi diagram-based approach (*VOR*) [10].

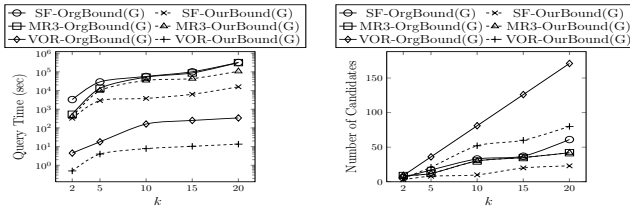
SF is an algorithm containing two steps. In the *filtering* step, it finds all objects whose *lower* bounds of their shortest surface distances to a given query point q are at most the k-th smallest *upper* bound of the shortest surface distance from an object to q. In the *refinement* step, it then computes the shortest surface distances of all objects found in the filtering step and returns k objects with the least shortest surface distances.

For the rest of the paper, we denote the various combinations of *algorithms*, *bound types*, and *graph types* as **A**-**O***Bound*(**G**). **A** is a placeholder for the implemented algorithms, with possible values $\{SF, MR3, VOR, MSRNN\}$ (where *MSRNN* is an algorithm [12] which will be used later in our experiments). **O** can be original/existing (*Org*) or new (*Our*) bounds, and **G** can be the original graph (*G*) or the smaller graph (*G'*). For example, *SF-OrgBound(G)* denotes the *SF* algorithm using existing original bounds on the original graph *G*.

In this section, we are studying the performance of **A**-*OurBound(G)* compared with **A**-*OrgBound(G)* for each existing algorithm **A**.

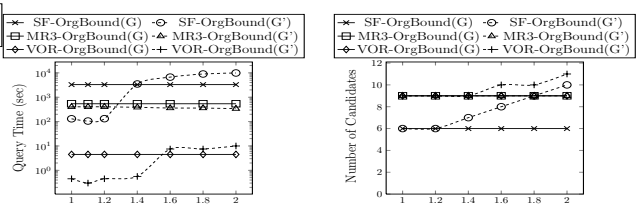
Figure 9(a) shows that every algorithm **A** using our bounds (i.e., **A**-*OurBound(G)*) is faster than its counterpart using the original existing bounds (i.e., **A**-*OrgBound(G)*) on graph G. Since k increases, fewer candidates (Figure 9(b)) need refinement due to our tighter lower bounds, resulting in an order of magnitude speedup in *VOR*/*SF*. Specifically, when k increases from 2 to 20, there is an increase from 9 to 25.2 times, respectively. Although *VOR* has a larger candidate set, its query time is the lowest. This is because it has the lowest cost of processing per candidate compared with other algorithms. Specifically, *VOR* precomputes the tight/loose cells and computes shortest surface distances *incrementally* by expanding cells. Since other algorithms lack such an incremental cell expansion, they have larger query times.

(2) Surface Range Queries: We conducted experiments for surface range queries with a fixed range of 500m. Since there are no existing algorithms for these queries, we conducted experiments with a straightforward (SF) algorithm only. Similar to surface kNN



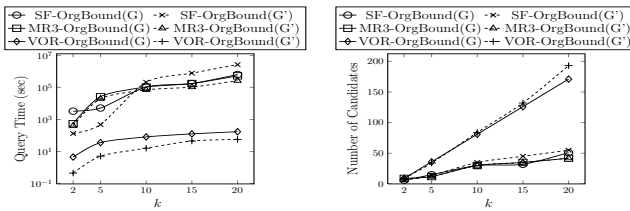
(a) (b)

Figure 9: Impact of Our Bounds on Original Graph G - Surface k-NN Queries: Effect of k



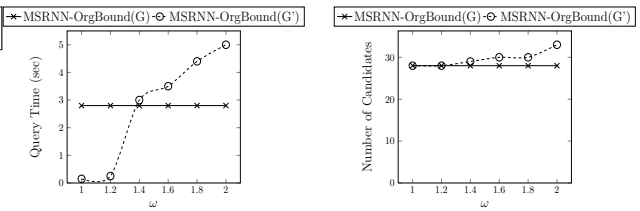
(a) (b)

Figure 10: Impact of Existing Bounds on Our Smaller Graph G' - Surface k-NN Queries: Effect of ω



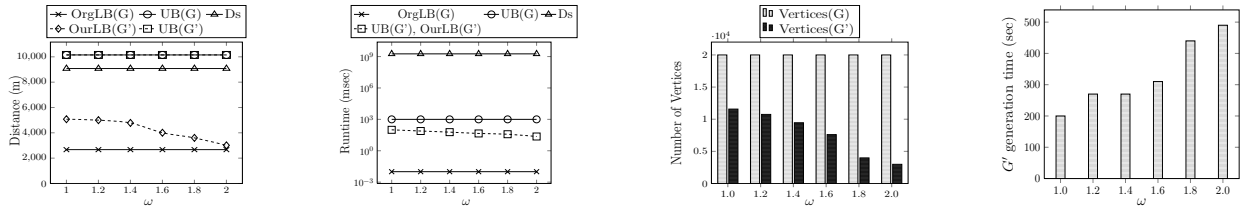
(a) (b)

Figure 11: Impact of Existing Bounds on Our Smaller Graph G' - Surface k-NN Queries: Effect of k



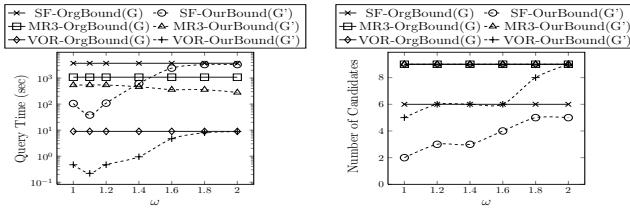
(a) (b)

Figure 12: Impact of Existing Bounds on Our Smaller Graph G' - Reverse Surface NN Queries: Effect of ω



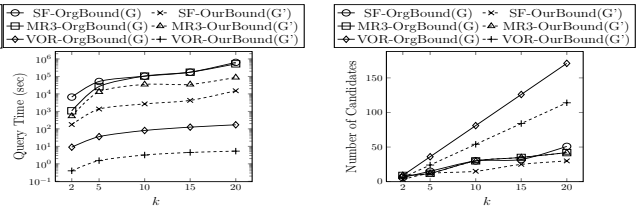
(a) (b) (c) (d)

Figure 13: Impact of Our Bounds on Our Smaller Graph - Distance Bounds: Effect of ω



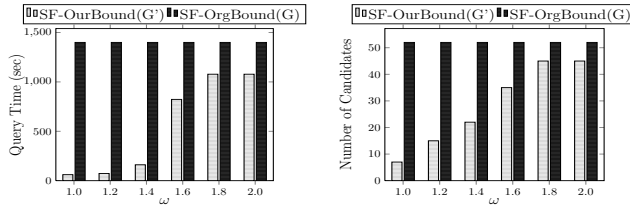
(a) (b)

Figure 14: Impact of Our Bounds on Our Smaller Graph - Surface k-NN Queries: Effect of ω



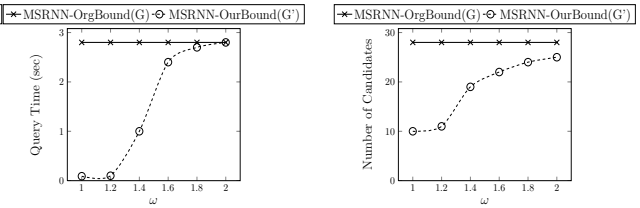
(a) (b)

Figure 15: Impact of Our Bounds on Our Smaller Graph - Surface k-NN Queries: Effect of k



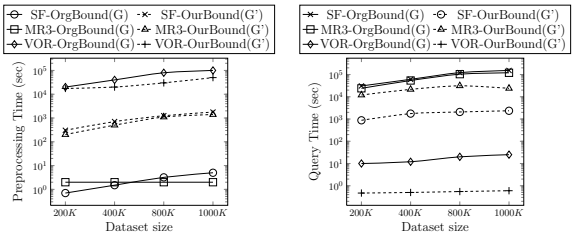
(a) (b)

Figure 16: Impact of Our Bounds on Our Smaller Graph - Surface Range Queries: Effect of ω



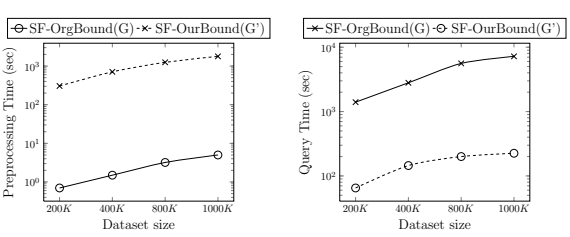
(a) (b)

Figure 17: Impact of Our Bounds on Our Smaller Graph - Reverse Surface NN Queries: Effect of ω



(a) (b)

Figure 18: Scalability: Surface k-NN Queries



(a) (b)

Figure 19: Scalability: Surface Range Queries

Query Type	Our Bound Only (Section 6.2)	Small Graph Only (Section 6.3)	Our Bound and Small Graph (Section 6.4)
Surface k-NN (VOR)	9	15	38
Surface Range	10	1	23.4
Surface Reverse k-NN	18.6	10	31.7

Table 2: Speedup Comparison (With Default Parameters)

queries, in the context of surface range queries, SF contains two steps. In the *filtering* step, it finds all objects whose *lower* bounds of their shortest surface distances to a given query point q are at most a given range value r. In the *refinement* step, it computes the shortest surface distances of all objects found in the filtering step and finally returns all objects whose shortest surface distances to q are at most r.

Similar to surface k-NN queries, we implemented it with two variations, namely *SF-OrgBound(G)* and *SF-OurBound(G)*. The query time and the number of candidates of *SF-OrgBound(G)* are $1,400$ seconds and 52, respectively. But, the query time and the number of candidates of *SF-OurBound(G)* are 107 seconds and 7, respectively, showing a speedup of nearly an order of magnitude.

(3) Reverse Surface NN Queries: We implemented the algorithm for monochromatic reverse surface NN queries in [12], namely *MSRNN*. In the following, we focus on reverse surface 1-NN queries. Similar to surface k-NN queries, we implemented it with two variations, namely *MSRNN-OrgBound(G)* and *MSRNN-OurBound(G)*. The query time and the number of candidates for *MSRNN-OrgBound(G)* is 2.8 second and 28, respectively. However, the query time and the number of candidates for *MSRNN-OurBound(G)* is 0.15 second and 10, respectively, which means that the algorithm using our bounds is 18.6 times faster and explores fewer candidates when compared to the one using the existing bounds.

Conclusion: We find that the algorithms using our bounds on the original graph perform more efficiently than the algorithms using existing bounds on the same original graph. Refer to the first contribution in the column with header "Our Bound Only" in Table 2 for speedups.

6.3 Existing Bounds and Our Smaller Graph

6.3.1 Distance Bound Comparison

Existing lower bound computation uses the Euclidean distance which is independent of the underlying graph and hence is unaffected by our graph compression. On the other hand, the upper bound which is the network distance on the smaller graph is affected. When $\omega = 1$, the existing upper bound calculated based on a smaller graph is $10,150$ meters (same as the network distance on the original graph) and takes 24.4 milliseconds to compute. However, it takes 2.8 seconds to find the existing upper bound calculated based on the original graph. Note that when $\omega = 1$, the bounds calculated based on the original graph are exactly the same as those calculated based on the smaller graph. Besides, there are some vertices which can be removed even when $\omega = 1$, resulting in a faster time. When ω increases to 2, the reduction in the number of vertices causes the upper bound to reach $10,290$ meters with a smaller runtime of 8.1 milliseconds. When bounds are computed based on G', substantial speedup is achieved.

6.3.2 Impact on Existing Methods

For each algorithm **A**, we study the effect of **A** using the original bounds on the smaller graph G' (**A**-*OrgBound(G')*) to **A** using original bounds on G (**A**-*OrgBound(G)*).

(1) Surface k-NN Queries: We varied ω and k to study the effects. Figure 10 shows the results when ω is varied. In Figure 10(a), when ω lies between 1 and 1.2, graph compression causes speedier bound computations. At $\omega = 1.1$, we achieve speedups of nearly 15 and 28 for *VOR* and *SF*, respectively.

However, for $\omega > 1.4$, the query times of *SF-OrgBound(G')* and *VOR-OrgBound(G')* get larger than *SF-OrgBound(G)* and *VOR-OrgBound(G)* respectively, because the bounds are looser, resulting in more exact surface distance computations. Figure 10(b) shows the number of candidates explored in each algorithm.

Figure 11 shows the results when k is varied. We observe a similar trend as before.

(2) Surface Range Queries: Improvement in surface range queries can only be found when the lower bound is improved, which in turn improves the pruning capacity in the filtering step. Since the original lower bound is the Euclidean distance, which is inert to changes in the underlying graph structure, we notice that varying ω does not change the lower bounds and hence does not affect the query time and the number of candidates to refine. In our experiments, the query time and the number of candidates to be refined are $1,400$ seconds and 52, respectively.

(3) Reverse Surface NN Queries: Since, in Figure 12, we observe a similar behavior as the *Surface k-NN Queries* in Section 6.3.2(1), the same explanations hold true in this case too. Note that in Figure 12(a), when $\omega = 1.4$ (i.e., 40% error), the query time of *MSRNN-OrgBound(G')* is slightly larger than that of *MSRNN-OrgBound(G)*. This is because when ω is a large value (in this case, $\omega = 1.4$), the error is already large (40%). Then, the bounds calculated based on G' are larger, resulting in more candidates which need more surface shortest path queries. We argue that 40% is already a large error and thus it is not recommended to set ω to a large value (e.g., 1.4).

Conclusion: We find that the algorithms using existing bounds on the smaller graph perform more efficiently than those using the same existing bounds on the original graph when ω is set to a value smaller than 1.4 (40% error). Refer to the second contribution in column with header "Small Graph Only" in Table 2 for speedups.

6.4 Our Bounds and Our Smaller Graph

6.4.1 Distance Bound Comparison

In Figure 13, we denote *OrgLB(G)* to be the original lower bound on G and *OurLB(G')* to be our lower bound on G'. We also denote *UB(G)* and *UB(G')* to be the upper bounds on G and G', respectively. We denote *Ds* to be the surface distance.

Figure 13(a) shows that our lower bound *OurLB(G')* is larger than the existing lower bound *OrgLB(G)*. Figure 13(b) shows that the computation time of Ds is the greatest and it took 504 hours to compute Ds. Computing the network distance on a smaller graph G' (i.e., *UB(G')* and *OurLB(G')*) is nearly an order of magnitude faster than computing the network distance on an original graph (i.e., *UB(G)*). Even when $\omega = 1$, the computation of the network distance on a smaller graph G' is faster. Figure 13(c) shows that the number of remaining vertices in G' decreases when ω increases. Furthermore, Figure 13(d) shows that the time of generating G' increases when ω increases because more vertices are removed.

6.4.2 Impact on Existing Methods

Similar to previous sections, for each algorithm **A**, we study the effect of our bounds on our smaller graph, denoted as **A**-*OurBound(G')*.

(1) Surface k-NN Queries: Figure 14 shows the performance of algorithms when ω changes. Figure 14(a) shows that the query times

of *SF-OurBound(G')* and *VOR-OurBound(G')* increase with ω because the upper/lower bounds calculated are looser and thus more candidates are generated for computing the exact shortest surface distances (as illustrated in Figure 14(b)). At $\omega = 1.1$, *SF* and *VOR* show speedups of 96 and 38 times, respectively.

Figure 15 shows the results of varying k. Figures 15(a) and (b) show that the query times and the number of candidates of all algorithms increase with k.

(2) Surface Range Queries: Figure 16(a) shows that the query time of *SF-OurBound(G')* is smaller than that of *SF-OrgBound(G)* since the computation time of *SF-OurBound(G')* is based on a smaller graph G' compared with *SF-OrgBound(G)* which is based on G. Specifically, our tightest bound ($\omega = 1$) produces a speedup of 23.4 times. When ω increases, the computation time of *SF-OurBound(G')* increases, since more candidates objects have to be explored (as shown in Figure 16(b)) due to looser bounds.

(3) Reverse Surface NN Queries: Figure 17(a) shows that the query time of *MSRNN-OurBound(G')* is smaller than that of *MSRNN-Orgbound(G)* when ω is smaller than 1.8. At $\omega = 1$, a speedup of 31.7 times is achieved. Figure 17(b) shows fewer candidates explored by the algorithm using our new bounds.

Conclusion: We find that the algorithms using our new tighter bounds on the smaller graph perform more efficiently than those using existing bounds on the original graph when ω is set to a value smaller than 1.4 (40% error). Table 2 compares the speedups of our individual contributions, i.e., "Our Bound Only" and "Small Graph Only," to the combined contributions "Our Bound and Small Graph."

6.5 Scalability

Here, we study the scalability of the existing algorithms described in Section 6.4.2 by varying the dataset size which is defined to be the total number of vertices used in the model.

Consider the scalability of surface k-NN queries with k set to 5. The *SF*, *MR3* and *VOR* approaches using our bounds (i.e., *SF-OurBound(G')*, *MR3-OurBound(G')* and *VOR-OurBound(G')*) have shorter query times compared with these approaches using the original bounds, as shown in Figure 18(b). In particular, the speedups of *SF*, *MR3* and *VOR* using our bounds are up to 68, 6.2 and 43 times, respectively, which is quite significant. Figure 18(a) shows the corresponding preprocessing times of these algorithms.

Consider the scalability for surface range queries (Figure 19). Similar results can be found in the figure. In particular, the speedup of SF using our bounds is at least 32.5 times.

6.6 Summary

In our experimental studies, our lower bound is up to 2.8 times larger (or better) than the Euclidean distance, the popular lower bound adopted in the literature. At $\omega = 1$, the computation of our upper bound computed on G' is 10 times faster than that of the original upper bound computed on G (with the same upper bound value). Importantly, all existing approaches relying on lower and upper bounds experience considerable speedups with our new bounds. In particular, the speedup experienced by *VOR*, i.e., the state-of-the-art algorithm, is up to 43 times for surface k-NN queries on the largest dataset (1M vertices, $k = 5$), which is quite significant.

In general, the best speedups are achieved when using our bounds on the smaller graph. A smaller graph can give a positive effect on faster bound computations but it can also introduce a negative effect on looser bounds (resulting in more candidates explored in some spatial queries). In our experiments, we find that ω should be set to a value smaller than 1.4 (which means a 40% error, a large error). When ω is set to a value smaller than 1.4, the positive effect outweighs the negative effect. When ω is set to a value larger than 1.4, in some cases, the negative effect may dominate the positive effect.

7. CONCLUSION

In this paper, we study a fundamental operation, i.e., shortest surface path computation, which is used widely in spatial queries. We find that we can compute the shortest network distance once and then use this distance for both the upper bound and lower bound of the shortest surface distance, which incurs only little overhead. In addition, when we need to compute the bounds quicker, we propose a method to generate a smaller graph from the Delaunay graph of the terrain such that the bound computation can be faster. Our experiments show that our lower bound is much tighter than the best-known lower bound. They also show that the existing state-of-art surface k-NN algorithm, i.e., *VOR*, can be speeded up nearly 43 times in the best case on the largest dataset.

There are a lot of promising research directions. First, it is of interest to derive the lower bound and upper bounds of the shortest surface path when the slope constraint is considered [8]. Second, it is of interest to study real time spatial queries such as continuous k nearest neighbors using our bounds.

8. ACKNOWLEDGMENTS

The work conducted by the co-authors from Aarhus University was supported by the Reduction project, funded by the European Comission as FP7-ICT-2011-7 STREP project number 288254. The work done by Raymond Chi-Wing Wong was supported by grant DAG11EG05G. The authors would also like to thank Dominik Scheder and Boris Aronov for valuable discussions and the reviewers for valuable comments.

9. REFERENCES

[1] M. De Berg. *Computational Geometry: Algorithms and Applications*. Springer, 2000.
[2] M. de Berg, M. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *Proc. SCG*, pages 294–303, 1997.
[3] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proc. SCG*, pages 360–369, 1990.
[4] D. L. Page, A. F. Koschan, M. A. Abidi and J. .L .Overholt. Ridge-valley Path Planning for 3D Terrains. *ICRA*, pages 119–124, 2006.
[5] K. Deng, X. Zhou, H. T. Shen, Q. Liu, K. Xu, and X. Lin. A multi-resolution surface distance model for *k*-nn query processing. *VLDB J.*, 17(5):1101–1119, 2008.
[6] M. Fredman and R. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Proc. FOCS*, pages 338–346, 1984.
[7] B. Kaneva and J. O'Rourke. An implementation of Chen and Han's shortest paths algorithm. In *Proc. CCCG*, 2000.
[8] L. Liu and R. C.-W. Wong. Finding shortest path on land surface. In *Proc. SIGMOD*, pages 433–444, 2011.
[9] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, NY, USA, 2nd ed, 1998.
[10] C. Shahabi, L. A. Tang, and S. Xing. Indexing land surface for efficient kNN query. *PVLDB*, 1(1):1020–1031, 2008.
[11] S. Xing, C. Shahabi, and B. Pan. Continuous monitoring of nearest neighbors on land surface. In *PVLDB*, 2(1):1114–1125, 2009.
[12] D. Yan, Z. Zhao, and W. Ng. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *Proc. CIKM*, pages 942–951, 2012.
[13] M. Kaul, R. C.-W. Wong, B. Yang, and C. S. Jensen. Finding Shortest Paths on Terrains by Killing Two Birds with One Stone (Technical Report). http://www.cse.ust.hk/~raywong/paper/terrain-technical.pdf