Proximity Queries on Terrain Surface

VICTOR JUNQIU WEI, Hong Kong Polytechnic University
RAYMOND CHI-WING WONG, Hong Kong University of Science and Technology
CHENG LONG, Nanyang Technological University
DAVID M. MOUNT, University of Maryland
HANAN SAMET, University of Maryland

Due to the advance of the geo-spatial positioning and the computer graphics technology, digital terrain data has become increasingly popular nowadays. Query processing on terrain data has attracted considerable attention from both the academic and the industry communities.

Proximity queries such as the shortest path/distance query, k nearest/farthest neighbor query and top-k closest/farthest pairs query are fundamental and important queries in the context of the terrain surfaces and they have a lot of applications in Geographical Information System, 3D object feature vector construction and 3D object data mining. In this paper, we first study the most fundamental type of query, namely shortest distance and path query, which is to find the shortest distance and path between two points of interest on the surface of the terrain. As observed by existing studies, computing the exact shortest distance/path is very expensive. Some existing studies proposed ϵ-approximate distance and path oracles, where ϵ is a non-negative real-valued error parameter. However, the best-known algorithm has a large oracle construction time, a large oracle size and a large query time. Motivated by this, we propose a novel ϵ-approximate distance and path oracle called the *Space Efficient distance and path oracle* (*SE*) which has a small oracle construction time, a small oracle size and a small distance and path query time thanks to its compactness of storing concise information about pairwise distances between any two points-of-interest. Then, we propose several algorithms for the k nearest/farthest neighbor and top-k closest/farthest pairs queries with the assistance of our distance and path oracle *SE*.

Our experimental results show that the oracle construction time, the oracle size and the distance and path query time of *SE* are up to two, three and five orders of magnitude faster than the best-known algorithm, respectively. Besides, our algorithms for other proximity queries including k nearest/farthest neighbor queries and top-k closest/farthest pairs queries significantly outperform the state-of-the-art algorithms by up to 2 orders of magnitude.

CCS Concepts: • **Information systems → Proximity search**; *Location based services*;

Additional Key Words and Phrases: shortest distance queries, shortest path queries, proximity queries, spatial database, terrain

Author's addresses: Victor Junqiu Wei, Department of Computing, Hong Kong Polytechnic University, email: wjqjsnj@gmail.com; Raymond Chi-Wing Wong, Department of Computer Science and Engineering, Hong Kong University of Science and Technology, email: raywong@cse.ust.hk; Cheng Long, School of Computer Science and Engineering, Nanyang Technological University, email: c.long@ntu.edu.sg; David M. Mount and Hanan Samet, Department of Computer Science, University of Maryland, email: {mount,hjs}@umd.edu .

DOI: 10.1145/nnnnnnn.nnnnnnn

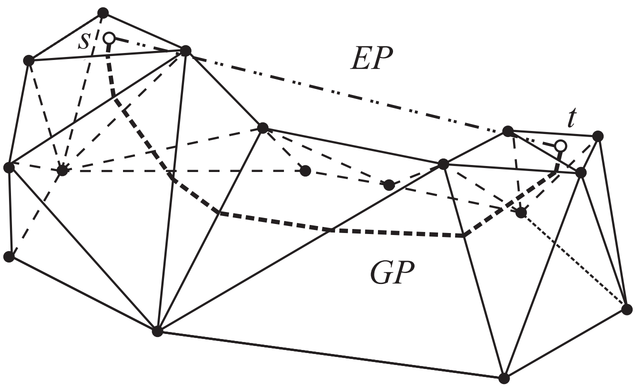

Fig. 1. An Example of Digital Terrain Surface and Geodesic Shortest Path

ACM Reference format:
Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, David M. Mount, and Hanan Samet. 2022. Proximity Queries on Terrain Surface. *ACM Trans. Datab. Syst.* 1, 1, Article 1 (September 2022), 65 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the advance of geo-spatial positioning and computer graphics technologies, digital terrain data has become increasingly popular nowadays, and it has been used in many applications such as Microsoft's Bing Maps and Google Earth in the industry community. The terrain data has also attracted considerable attention from the academic community [11, 13, 24, 25, 30, 39, 46, 47].

Terrain data is a planar graph, consisting of vertices, edges, and faces, where all the faces are triangles. Each face (or triangle) has three line segments called *edges* connected with each other at three *vertices*. An example of a piece of terrain data is shown in Figure 1, where we have 24 faces, 40 edges and 17 vertices. In this example, there are two points-of-interests (POIs), namely s and t, on the terrain surface. As could be observed from this figure, a POI may lie on a vertex of the terrain or not.

The *geodesic path* between two given locations (or points) on the surface of the terrain is the *shortest* path/route from one point to the other on the surface. The *geodesic distance* between two given locations (or points) on the surface of the terrain is the length of the geodesic path between them. For example, in Figure 1, s and t are two points-of-interest on the terrain surface and the shortest path from point s to point t is shown and is denoted by GP, which corresponds to a sequence of line segments on the faces of the terrain. Thus, GP is the geodesic path between s and t and the length of GP is the geodesic distance between s and t. Note that the geodesic distance is usually quite different from the Euclidean distance, and according to [11], in typical terrains, the geodesic distance can exceed the Euclidean distance by up to 300%. In Figure 1, the Euclidean distance between s and t is the length of the line segment EP. Proximity queries on the terrain surface are all based on the geodesic distances instead of the traditional Euclidean distance.

1.1 Application

In many applications, a set of points-of-interest (POIs) on the surface of the terrain is given, and the proximity queries (e.g., shortest distance/path query, *k* nearest/farthest neighbors query and top-*k* closest/farthest pairs query) have a lot of applications. Some examples are detailed as follows.

(1) Geographic Information System (GIS). In GIS, it is important to compute the geodesic shortest distance/path between two POIs. For example, hikers need the geodesic distance (resp. path) to measure the travel time (resp. obtain a hiking route) between a source and a destination which are landmarks (i.e., POIs) in practice [38]. Besides, the vehicles (e.g., Google Map camera car and military vehicles) estimate the geodesic distance (resp., path) to measure the travel cost (resp., obtain the travel route) [28, 43]. In life sciences, scientists conduct shortest distance and shortest path queries on residential locations (i.e., POIs) of the animals in the wildness to study their migration patterns [14, 31].

(2) Computer Graphics and Vision. In computer graphics and vision [27, 40], measuring similarities between two different 3D objects is very important. In order to measure similarities between objects, a number of reference points (which are POIs) [27, 40] on the surface of each object are selected. These reference points play an important role in similarity measurement since they are invariant to transformations such as rotation and translation. For each object, geodesic distances between all pairs of reference points are computed and are stored as a feature vector for similarity measurement. In this application, multiple geodesic distance computations are involved.

(3) Scientific Data 3D Modeling. There is a need to model scientific data in 3D models in areas like biology, chemistry, anthropology and archeology [1, 20]. In neuroimaging, similar to computer graphics and vision, a 3D model of an organ is associated with a set of reference points [1, 20] (which are POIs) corresponding to functional units on the organ and the scientists use the geodesic distances between reference points to analyze tumor development with magnetic resonance imaging (MRI) images. In neuroscience, scientists conducted spatial queries on a 3D brain model to study the neuron density and the number of branches in a region of the brain [41]. Similarly, multiple geodesic distance computations are involved in this application.

(4) Online 3D Virtual Game. In some online 3D virtual games like INGRESS and Pokémon GO, a city (e.g., San Francisco in INGRESS) has a terrain surface which consists of a number of *portals* or landmarks containing several monsters (which are POIs). For each portal or landmark, it is important to calculate the geodesic distance from this portal to each of the other portals so that the influence of this portal is estimated. Here, multiple computations for geodesic distances are involved. For each user, the *k*NN queries are required to obtain the portals or landmarks in their vicinity to visit and the geodesic shortest path is needed for the navigation.

(5) Spatial Data Mining. There are many data mining techniques used in spatial databases. For example, in the clustering technique, the inner-cluster distance (i.e., the monochromatic farthest pairwise distance) and the inter-cluster distance (i.e., bichromatic closest pairwise distance) are needed. In the co-location pattern mining, shortest distance and path queries, bichromatic nearest neighbor queries and monochromatic/bichromatic closest/farthest pairwise distance queries are also used frequently. In a city setting, examples of POIs include buildings and parks. In a wilderness setting, examples of POIs include radio-telemetry receivers set up for collecting animal movement. In the context of spatial data mining, the number of geodesic distance computations is very large.

1.2 Motivation

Consider a terrain T with N vertices. Let P be a set of n POIs on the surface of the terrain.

Due to a variety of applications in different domains as described in Section 1.1, computing geodesic distances and paths [2, 4, 7, 15, 24, 25, 30, 32, 45] is very important and is very fundamental to other proximity queries such as nearest neighbor queries [12, 13, 24, 25, 39, 46] and reverse nearest neighbor queries [25, 47].

Motivated by this, we aim to study three kinds of distance and shortest path queries, namely *vertex-to-vertex* (*V2V*) *distance and shortest path queries*, *POI-to-POI* (*P2P*) *distance and shortest path queries* and *arbitrary point-to-arbitrary point* (*A2A*) *distance and shortest path queries*. For the better clarity, we simply refer to the above three types of queries as P2P query, V2V query and A2A query, respectively. Consider the first two types of queries. Each V2V query returns the geodesic distance and path between a starting point s and a destination point t, where both s and t are vertices (from V). Each P2P query returns the geodesic distance and path between a starting point s and a destination point t, where both s and t are POIs (from P). Since P2P queries, considering both the concept of vertices and the concept of POIs, is more general than V2V queries, considering only the concept of vertices without the concept of POIs, P2P queries could be regarded as a generalization of V2V distance (resp. path) queries. Specifically, under the problem setting for P2P queries, if for each vertex in the problem setting for V2V queries, we create a POI which has the same coordinate values as this vertex, then the P2P queries will become the V2V queries. Thus, for clarity, in this paper, we focus on P2P queries. Consider the third type of queries. Each A2A query returns the geodesic distance and path between a starting point s and a destination point t, where both s and t are two arbitrary points on the surface of the terrain. Since A2A queries allow all possible points on the surface of the terrain, A2A queries generalize both P2P and V2V queries. For the ease of illustration, in the main body of this paper, we first study P2P queries. Later, in Appendix F, we study A2A queries.

Our natural goal of answering each P2P distance and shortest path query is to return the corresponding distance and path in a short time. However, none of the existing studies [2, 4, 7, 15, 24, 25, 30, 32, 45] could achieve this goal satisfactorily.

Firstly, all existing algorithms [7, 32, 45] computing exact geodesic distances and paths on-the-fly are still slow even for moderate-sized terrain data. The time complexities of the algorithms for computing exact geodesic distances proposed by [7, 32, 45], are $O(N^2 \log N)$, $O(N^2)$, $O(N \log^2 N)$ and $O(N^2 \log N)$, respectively, which is still very large when N is large. In the literature [24, 25, 39, 46], the algorithm proposed in [7] is recognized as a state-of-the-art algorithm in terms of efficiency. Many existing studies [24, 25, 39, 46] adopt this for finding the geodesic distances and paths. According to [24], the algorithm proposed in [7] took more than 300 seconds on a terrain with 200K vertices, which is far from being satisfactory.

Secondly, although some existing algorithms [24, 25, 30] were proposed to compute *approximate* geodesic distances and paths on-the-fly for reducing the computation time, all of these algorithms are still not efficient enough for proximity queries and applications involving many distance and path queries. The algorithm in [30] computes the approximate geodesic distance/path satisfying a *slope condition*, the algorithm in [25] computes the lower and upper bounds of the geodesic distances between two points which provides no guarantees on the qualities of the bounds found, and the algorithm in [24], which is an improved version of that in [25], runs in $O((N+N')\log(N+N'))$ time where N' is the number of additional vertices introduced to achieve a guarantee on the qualities of the lower and upper bounds found.

For other types of proximity queries, [12, 13, 39] studied the monochromatic version of kNN queries. But, they all take $O(N\log^2 N)$ time to answer a kNN query, which is very costly and prevents their usage in real-time applications. Besides, to the best of our knowledge, there is no existing work on the bichromatic version of k-nearest neighbor queries in the context of terrain datasets.

There are other proximity queries. One example is monochromatic and bichromatic k-farthest neighbor queries and the other example is monochromatic and bichromatic closest/farthest pair queries. Again, to the best of our knowledge, there is no existing work on these queries.

1.3 Our Distance and Path Oracle and Proximity Query Processing Algorithms

To efficiently process the geodesic distance and path queries, especially for those cases where queries for many different pairs of points are issued, some existing studies [2, 4, 15] aim at designing geodesic distance (e.g., [**?**]) (and/or the corresponding shortest path) oracles. To the best of our knowledge, all existing studies focused on building oracles for returning approximate geodesic distances or approximate geodesic paths only but no existing studies focused on building oracles for returning exact geodesic distances (which could be explained by the high computation cost of computing the exact geodesic distances and exact geodesic paths). All of these studies [2, 4, 15] are based on *auxiliary point-based oracles*. Specifically, they first introduce a large number of auxiliary points (edges), namely *Steiner points* (*edges*), on the surface of the terrain where each Steiner edge connects two Steiner points. Then, they construct a graph G_ϵ whose vertices (edges) are either original vertices (edges) or the Steiner points (edges). The exact distance between any two vertices/points on G_ϵ is an ϵ-approximate geodesic distance between these two vertices/points. The ϵ-approximate geodesic distance oracles proposed in [2, 4, 15] indexes the exact distances on G_ϵ. Among these studies, the oracle in [15] is the best, where the space complexity of the oracle (called the *oracle size*) is $O(\frac{N}{\sin(\theta)\cdot\epsilon^{1.5}}\log^2(\frac{N}{\epsilon})\log^2\frac{1}{\epsilon})$ where θ is the minimum inner angle of any face of the terrain surface. It can answer ϵ-approximate P2P distance and path queries in $O(\frac{1}{\sin(\theta)\cdot\epsilon}\log\frac{1}{\epsilon}+\log\log N)$ time.

Unfortunately, these auxiliary point-based oracles have two drawbacks. The first drawback is that each of these oracles has a large oracle building time and a large oracle size. This is because a large number of Steiner points (edges) are introduced during the oracle construction process and the number of Steiner points could be several orders of magnitude larger than the number of vertices on the surface of the terrain. Thus, each of these oracles has a poor empirical performance in terms of both the oracle building time and the oracle size. The second drawback is that each of these oracles is constructed based on the *structure* of the terrain without considering the information about POIs. In other words, it is constructed based on the set of vertices regardless of the set of POIs. For example, consider the case where there are only two POIs, a naive oracle storing the geodesic distance for one pair (of POIs) occupies a $O(1)$ space only but the oracle in [15] could introduce millions of Steiner points, resulting in a large oracle size and a large oracle building time.

Motivated by the drawbacks of the existing methods, we propose a distance oracle called the *S̲pace-E̲fficient Distance Oracle* (SE) such that for any point s and any point t in P, the oracle returns an *ϵ-approximation* of the geodesic distance between s and t efficiently, where

ϵ is a non-negative real user parameter, called the *error parameter*. Our *SE* has three good features: (1) small construction time, (2) small size and (3) small query time (compared with the best-known oracle [15]). This is because *SE* is *space-efficient* in the sense that its size is linear to n (i.e., no of POIs). Due to this space-efficient property, it is much easier for us to design an efficient algorithm for constructing the *SE* and an efficient algorithm for answering distance and shortest path queries.

Based on this oracle which provides the geodesic distance information of POIs, we also develope efficient query processing algorithms for other proximity queries such as *k nearest/farthest neighbors* and *top-k closest/farthest pairs queries*.

1.4 Contribution & Organization

We summarize our major contributions as follows. Firstly, we propose a novel distance oracle called *SE*, which can be computed efficiently, has small size and can answer ϵ-approximate geodesic distance and shortest path queries efficiently. Our *SE* answers not only P2P distance and path queries but also V2V distance and path queries. Thirdly, for V2V distance and path queries, our experimental results show that the building time, oracle size and query time of *SE* are, respectively, 5-100 times, 10-100 times and more than 1000 times smaller than those of the best-known distance and path oracle [15] on benchmark real datasets. For P2P distance and path queries, the building time, oracle size and query time of *SE* are 10-100 times, 10-1000 times and 100-10000 times smaller than those of the best-known distance and path oracle [15] on benchmark real datasets, respectively. Besides, for the other types of proximity queries studied (i.e., the k nearest neighbor and farthest neighbor queries, top-k closest pair and farthest pair queries), our algorithm also significantly outperforms the state-of-the-art.

This paper is an extension of the previous conference paper [44]. The conference version [44] is concerned with the distance query processing only. It proposed an indexing structure, namely distance oracle, for distance query on terrain surface. The oracle provides approximate distances with accuracy guarantee and significantly outperforms the state-of-the-art in terms oracle building time, oracle size and query time in the perspective of both theory and empirical performance. Compared with [44] which could be applied to distance query processing only, this paper extends the *distance oracle* studied to a *Distance and Path Oracle*. Specifically, we incorporate a new component into the oracle which renders it being able to support both distance and path queries. We also derive a non-trivial error bound for our path query algorithm. Besides, based on this oracle, the paper further studies many other proximity queries such as *k nearest/farthest neighbors* and *top-k closest/farthest pairs queries* which are also fundamental queries in spatial data processing and widely applied in many scenarios. For each query studied, we develop a new query processing algorithm with a theoretical guarantee for the accuracy and the time complexity. The additional experiments on the newly studied queries verify the efficiency and effectiveness of our newly developed techniques which significantly outperforms the competitors by orders of magnitudes.

We would like to make three notes on the new contribution of this work compared with its preliminary version [44]. (1) Despite that the distance query and the path query are closely related, the *SE* method in [44], which is a distance oracle only, can not handle the path query. This is because the *SE* oracle studed in [44] does not store any information regarding the shortest *path* from a source to a destination. Recall that the *SE* method pre-computes a set of distances and the query processing involves the look-up operations of the pre-computed distances and finally finds the one which could approximate the distance from a source to a destination. As such, although several on-the-fly single-source all-destination algorithms are performed in the preprocessing phase and they may involve some path information, *SE* only stores the distance information in the query phase (which are simply real-valued numbers) and discards redundant information (e.g., path information) if any. (2) Extending the original *SE*, which is a distance oracle only, to the oracle with the support of both distance and path query processing is challenging. Firstly, while the original *SE* could find the corresponding distance pre-computed for a source s and a destination t in the query, the two end-points of the distance fetched from the original SE are not necessarily s and t, and are normally other points. As such, it is still an open question how to utilize the auxiliary information provided by the pre-computed distance information of *SE* to support the path query given the mis-alignment problem between the points of the distance pre-computed and the query points as mentioned before. Secondly, in the design of the path query processing algorithm (which is supposed to fix the gap between the pre-computed distance information and the query points), it is non-trivial to guarantee the approximation error for an extension of the SE method. As could be noticed from the proof of Theorem 3.13, it requires considerable research effort in deriving the theoretical results of the approximation ratio. (3) Although our newly developed techniques in this paper adopt [44] as an inherent component (which could be regarded as an extension of [44]), the newly developed techniques are all up-to-date and could not be found in [44]. The proposed technique in this paper for each newly studied query (e.g., shortest geodesic path query, k nearest/farthest neighbor query and top-k closest/farthest pair query) is the state-of-the-art of the corresponding query.

The remainder of the paper is organized as follows. Section 2 provides the problem definition. Section 3 presents our distance and path oracle, namely *SE*. Section 4 gives the other proximity queries studied and the proposed query processing algorithms. Section 5 reviews the related work and introduces some baseline methods of the distance and path oracles and our query processing algorithms for other proximity queries. Section 6 presents the experimental results and Section 7 concludes the paper.

2 PROBLEM DEFINITION

Consider a terrain T. Let V be the set of all vertices on the surface of the terrain T, and E be the set of all edges on the surface of the terrain T. Let N be the size of T (i.e., $N = |V|$). Each vertex $v \in V$ has three coordinate values, denoted by x_v, y_v and z_v.

Let P be a set of POIs on the surface of the terrain T and n be the size of P (i.e., $n = |P|$). In the following discussion, we focus on the case when $n \leq N$. This is because in real-life applications, $n \leq N$. For example, in the BearHead dataset, one benchmark dataset used in the literature, $n = 4k$ and $N = 1.4M$. In the EaglePeak dataset, the other benchmark dataset, $n = 4k$ and $N = 1.5M$. The discussion about how we handle the case when $n > N$ can be found in Appendix H. Each POI $p \in P$ also has three coordinate values, denoted by x_p, y_p and z_p. In this paper, we assume that P contains no duplicate points since any two co-located POIs can be regarded as one POI in practice, and we can merge any two co-located POIs into one POI by a simple preprocessing step. Besides, in this paper, similar to many existing studies of the index-based algorithms, we leave the POI update operation (i.e., insertions and deletions) as a future work and focus on the static setting of the POIs.

Given two points, s and t, on the surface of T, the *geodesic shortest path* between s and t, denoted by $\Pi_g(s, t)$, is defined to be the shortest path between the two points on the surface of T. Note that the geodesic shortest path corresponds to a sequence of line segments on the surface of the terrain. Consider the example in Figure 1 where the geodesic shortest path between two points s and t is denoted by GP. Given two points, s and t, on the surface of T, the *geodesic distance* between s and t, denoted by $d_g(s,t)$, is defined to be the length of the geodesic shortest path between the two points, i.e., $\Pi_g(s,t)$, where the length of a path is defined to be the sum of the lengths of all line segments of the path. The geodesic distance $d_g(\cdot,\cdot)$ is a metric, and therefore it satisfies the triangle inequality.

Note that a full materialization of geodesic distances for all possible pairs of points in P is not feasible since the complexity of the oracle size and the complexity of the oracle building time are $O(n^2)$ and $O(nN\log^2 N)$, respectively, which are prohibitively large. Besides, a full materialization of all pairwise geodesic paths consumes even more space and requires more oracle building time.

As we described in Section 1, we would like to have an indexing structure with a short query time (for shortest distance queries, shortest path queries, monochromatic and bichromatic k nearest neighbor and farthest neighbor queries, and also monochromatic and bichromatic top-k closest pair and farthest pair queries), a low space cost and a short preprocessing step. Besides, it has accuracy guarantee for each query considered.

3 DISTANCE AND PATH ORACLE

We first present the overview of our distance and path oracle called SE in Section 3.1. Then, we present the first component of SE, called the *compressed partition tree*, in Section 3.2, the second component of SE, called the *node pair set*, in Section 3.3, the distance query processing algorithm based on SE in Section 3.4, the path query processing based on SE in Section 3.5, the construction algorithm of SE in Section 3.6, and some theoretical results of SE in Section 3.7.

3.1 Overview

Before giving an overview, we first give the concept of a *disk*. Given a point $p \in P$ and a non-negative real number r, a *disk* centered at p with radius equal to r on the terrain surface, denoted by $D(p,r)$, is defined to be a set of all possible points on the terrain surface whose geodesic shortest distance to p is at most r. That is, $D(p,r) = \{p' | d_g(p',p) \leq r \text{ and } p' \text{ is an arbitrary point on the terrain surface}\}$.

With this concept, we are ready to describe our distance oracle SE which includes two major components, namely the *compressed partition tree* and the *node pair set*.

The first component is the compressed partition tree in which each node corresponds to a disk containing a set of POIs. In the leaf level of the tree, there are n nodes each of which corresponds to a disk containing only one POI. Each node in this level has the smallest radius (since each node contains only one POI). In the level just above the leaf level of the tree, there are fewer nodes each of which corresponds to a disk containing one or more POIs. Each node in this level has a larger radius (since each node contains one or more POIs). Similarly, each node in a higher level has a larger radius. At the root level of the tree, the (root) node has the largest radius since it contains all n POIs. Note that for different levels, the tree has different number of nodes (with different radius).

The second component is the node pair set which is a set of the pairs of nodes from the compressed partition tree. In this node pair set, each node pair in the form of $\langle O, O' \rangle$ is associated with two sub-components: 1. the *distance* between the centers of the corresponding disks of O and O', 2. a list of several *intermediate points* lying on the shortest path between the centers of the corresponding disks of O and O', where O and O' are two nodes in the compressed partition tree. Besides, the node pair set satisfies one interesting property called the *unique node pair match property* which is the key to the query efficiency of our SE. The unique node pair match property states that for any two points, namely p and q, in P, there exists *exactly one* node pair $\langle O, O' \rangle$ in the node pair set such that O contains p and O' contains q.

Consider a distance query with a source point $s \in P$ and a destination point $t \in P$. Let h be the height of the tree. In all of our experimental results on benchmark real terrain datasets, h is smaller than 30. We could answer this distance query in $O(h)$ time using SE. The major idea is to find a node pair $\langle O, O' \rangle$ in the node pair set *efficiently* such that O contains s and O' contains t, and return the distance associated with this node pair. Interestingly, even though the distance returned is associated to this node pair, it will be shown later that the distance returned is an ϵ-approximation of the geodesic distance between s and t. Besides, each node pair maintains a set of *intermediate points* to be described later in details, each of which lies on the ϵ-approximate shortest path between the centers of the corresponding disks of O and O'.

Consider a path query from s to t. Our path query processing concatenates the following three sub-paths and returns the final concatenation: 1. the shortest path from s to the center of the corresponding disk of O; 2. the ϵ-approximate shortest path between the centers of the corresponding disks of O and O'; 3. the shortest path from the center of O' to t. For finding the second subpath efficiently, we adopt an efficient method by taking the advantage of the intermediate points which will be described in detail later.

The major challenge here is how to design SE which achieves the *space-efficient* property (mentioned in Section 1). We will describe the details of how we address this challenge.

3.2 Oracle Component 1: Compressed Partition Tree

In this section, we first present a hierarchical structure called a *partition tree* to index all POIs in P, which is used for constructing the first component (i.e., the compressed partition tree) of our distance oracle SE.

A *partition tree* is defined to be a tree with the following components.

- Each node O in the tree has two attributes, namely its *center*, denoted by c_O, and its *radius*, denoted by r_O, where c_O is a point in P and r_O is a non-negative real number.
- For each leaf node O, $D(c_O, r_O)$ contains only one point in P (which is c_O) (and thus contains no objects in P other than c_O). Note that there are n leaf nodes.
- For each internal node O, the center of each child of node O is in $D(c_O, r_O)$ and the radius of each child of node O is equal to $0.5 \cdot r_O$.
- Each node O in the tree is associated with its *representative set*, denoted by $RS(O)$, which is defined to be a set containing the centers of all the leaf nodes in the subtree rooted at O.

Given two nodes, namely O and O', the (*geodesic*) *distance* between O and O', denoted by $d_g(O, O')$, is defined to be $d_g(c_O, c_{O'})$.

Let h be the height of the partition tree. The partition tree has $h + 1$ layers, namely Layer 0, Layer 1, ..., Layer h. Layer 0 is the layer containing the root node only. For each $i \in [1, h]$, Layer i is the layer containing all child nodes of each node in Layer $(i - 1)$. Finally, Layer h is the layer containing all leaf nodes. If a node is in Layer i where $i \in [0, h]$, we also say that the depth of this node is i. Note that all nodes in the same layer have the same radii. The *radius* of Layer i, denoted by r_i, is defined to be the radius of one of the nodes in Layer i. For any $i, j \in [0, h]$, we say that Layer i is *higher* than Layer j (or Layer j is *lower* than Layer i) if and only if $i < j$.

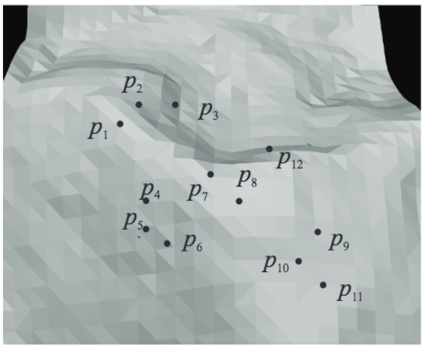

Fig. 2. An Example

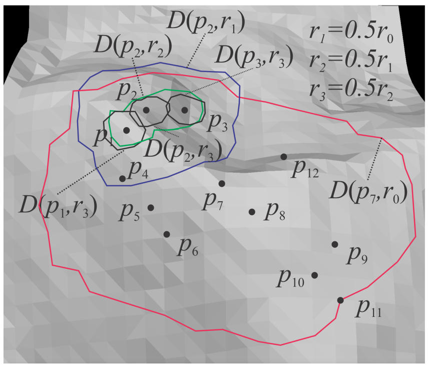

Fig. 3. Some Disks Used in Our Example

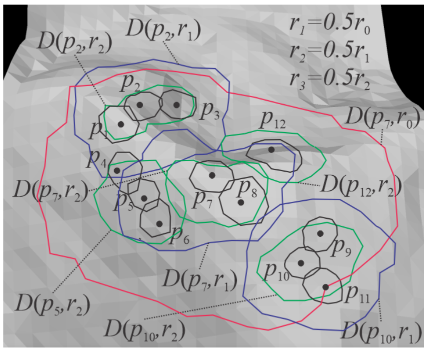

Fig. 4. All Disks Used in Our Example

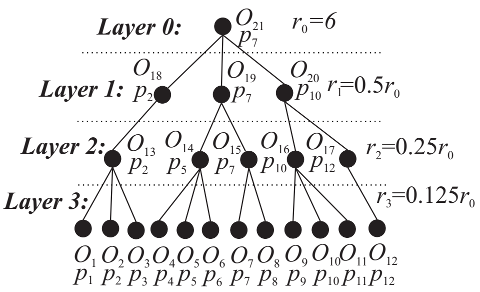

Fig. 5. An Example of Partition Tree

Next, we give the three properties of this partition tree to be satisfied. We will describe how to construct a partition tree satisfying these three properties later.

- **Separation Property:** For each $i \in [0, h]$, the radius of each node in Layer i is $\frac{r_0}{2^i}$ and the geodesic distance between any two nodes in this layer is at least $\frac{r_0}{2^i}$.
- **Covering Property:** For each layer where X denotes a set of all nodes in this layer, the region represented by $\bigcup_{O \in X} D(c_O, r_O)$ covers all points in P.
- **Distance Property:** For each node O in the tree, if O' is one of the descendant nodes of O, then $d_g(c_O, c_{O'})$ is at most $2 \cdot r_O$, i.e., $c_{O'}$ is in the disk $D(c_O, 2 \cdot r_O)$.

Given a node O in the partition tree, the *enlarged disk* of node O is defined to be $D(c_O, 2 \cdot r_O)$. From the Distance Property, we deduce that for each node O in the partition tree, all points in $RS(O)$ (which are points in P) are in the enlarged disk of node O.

Example 3.1 (Partition Tree). Consider the points on a terrain surface as shown in Figure 2. There are 12 points $p_1, p_2, p_3, ..., p_{12}$ in P.

Figure 3 shows three small disks, namely $D(p_1, r_3)$, $D(p_2, r_3)$ and $D(p_3, r_3)$, one medium-small disk, namely $D(p_2, r_2)$, one medium-large disk, namely $D(p_2, r_1)$, and one large disk, namely $D(p_7, r_0)$, where r_0, r_1, r_2 and r_3 are four non-negative real numbers. Note that r_0 is the radius of the large disk, r_1 is the radius of the medium-large disk, r_2 is the radius of the medium-small disk and r_3 is the radius of one of the small disks. We also show all disks to be used in this example in Figure 4.

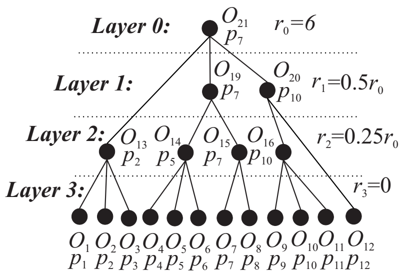

Fig. 6. An Example of Compressed Partition Tree

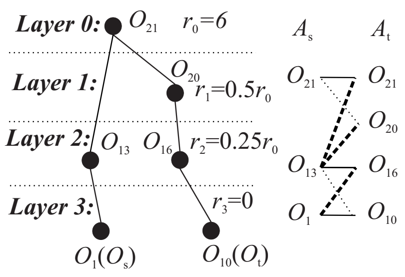

Fig. 7. An Example of Distance Query Processing

There are 21 disks in the figure, each of which centers at a point. For example, the disk $D(p_7, r_0)$ is a disk with its center equal to p_7 and its radius equal to $r_0 = d_g(p_7, p_{11})$.

Figure 5 shows a partition tree of height equal to 3 which is built based on the 12 points shown in Figure 2. In this figure, each black dot corresponds to a node in the tree. By definition, any two nodes in the same layer have the same radii. In Layer 0, there is only one node O_{21} (i.e., the root node) with its radius r_0 equal to $d_g(p_7, p_{11})$. In Layer 1, there are three nodes, namely O_{18}, O_{19} and O_{20}, each with its radius r_1 equal to $0.5r_0$. In Layer 2, there are 5 nodes, namely $O_{13}, O_{14}, O_{15}, O_{16}$ and O_{17} each with its radius r_2 equal to $0.25r_0$. In Layer 3, there are 12 nodes (i.e., leaf nodes), namely $O_1, O_2, ..., O_{12}$, each with its radius r_3 equal to $0.125r_0$. In the figure, we list the center of each node below the label of the node. For example, there is a label p_2 below the label O_{13}, which means that the center of O_{13} is p_2.

Consider the leaf node O_1 with its center equal to p_1 and its radius equal to r_3. It is easy to see that disk $D(p_1, r_3)$ contains only one point in P (i.e., p_1) as shown in Figure 3. The representative set of this node is a set containing only the center of this node (i.e., p_1). This holds as well for each of the other leaf nodes (e.g., node O_2 and node O_3).

Consider the internal node O_{13} with its center equal to p_2 and its radius equal to r_2. The center of each child of node O_{13} (i.e., node O_1, node O_2 and node O_3) is in disk $D(p_2, r_2)$ as shown in Figure 3. Besides, the radius of each child of node O_{13} is equal to $0.5 \cdot r_2$ (since the radius of each child is equal to $0.125r_0$ and $r_2 = 0.25r_0$). The representative set of this node is a set containing the centers of all the leaf nodes in the subtree root at O_{13} (i.e., the center of node O_1 (which is p_1), the center of node O_2 (which is p_2) and the center of node O_3 (which is p_3)). This holds as well for each of the other internal nodes.

It is easy to verify that the partition tree shown in this figure satisfies the three properties described above.

Next, we present our top-down method for building the partition tree.

- **Step 1 (Root Node Construction):** We create the root node as follows.
  - **Step (a) (Initialization):** We assign a variable i, denoting the layer number, with 0.
  - **Step (b) (Point Selection):** We randomly select a point p in P.
  - **Step (c) (Radius Computation):** We perform a single-source all-destination (SSAD) exact shortest path algorithm [7, 32, 45] which takes p as an input of the source point and executes until the search region of the algorithm covers all points in P. When we terminate the algorithm, we obtain the maximum distance d between p and a point in P.

  - **Step (d) (Node Construction):** We create a root node O where its center is set to p and its radius is set to d. Note that in this layer, $r_0 = d$.
- **Step 2 (Non-Root Node Construction):** We perform the following operations.
  - **Step (a) (Initialization):** We increment variable i by 1. We initialize a variable P', denoting a set of remaining points in P to be "covered" by a node in Layer i, to be P and in each iteration of Step (b), we remove some POIs from P' until P' is empty (i.e., each time we create a node, we remove some POIs "covered" by the node from P' until P' is empty).
  - **Step (b) (Iterative Step):** We perform the following iterative steps.
    - **Step(i) (Point Selection):** Let C be a set containing the centers of all nodes in Layer $i-1$. Let P_C be the set of remaining points in P' each of which is one of the centers of all nodes in Layer $i-1$ (i.e., $P_C = P' \cap C$). We will give a higher priority to points in P_C in the point selection. We randomly select a point p from P_C if $P_C \neq \emptyset$, and select a point p from P' based on a point selection strategy (to be described later) otherwise.
    - **Step (ii) (Point Covering):** We find a set S of all points in P' that are in $D(p, \frac{r_0}{2^i})$ by performing a single-source all-destination (SSAD) exact shortest path algorithm which takes p as an input of the source point and executes the algorithm until the distance between the boundary of the search region and p is greater than $\frac{r_0}{2^i}$. We remove all points in S from P'.
    - **Step (iii) (Node Creation):** We create a node O where its center is set to p and its radius is set to $\frac{r_0}{2^i}$. Then, we find the node O_{parent} in Layer $(i-1)$ whose distance to O is the minimum. We set the parent of O to O_{parent}.
    - **Step (iv) (Additional Node Creation):** We repeat the above steps (i.e., Steps (i)-(iii)) until P' is empty.
  - **Step (c) (Next Layer Processing):** We repeat the above steps (i.e., Step (a) and Step (b)) until the number of nodes in Layer i is equal to n.

Lemma 3.2. *The partition tree generated by the above procedure satisfies the Separation, Covering and Distance Properties.*

Proof. For the sake of space, all the proofs in the paper can be found in Appendix B. □

Some implementation details of this algorithm are given as follows.

Implementation Detail 1 (Point Selection Strategy in Step 2(b)(i)): We propose two heuristic-based point selection strategies as follows. The first one is called the *random selection strategy*. It *randomly* selects a point p from P'. The second one is called the *greedy selection strategy* which is to select a point from P' in the "densest" region (or formally cell) on the surface of the terrain. The major idea of this strategy is to select a point from P' in the densest region (because if this point is selected as the "center" of the disk, then this disk can cover many points (which could come from the densest region)). Specifically, this strategy requires some additional operations included in other steps, and we describe them as follows. (A) Between Step 2(a) and Step 2(b), we construct a grid on the x-y plane with the cell width equal to $O(\frac{r_0}{2^i})$. Then, we insert all points from P' in corresponding cells, and all point IDs in each cell are indexed in a B+-tree. We also build a max-heap containing all non-empty cells whose keys are the sizes of their B+-trees. (B) In Step 2(b)(i), in the case that $P_C = \emptyset$, we select a point p in P' by finding the cell with the greatest number of points in P' and randomly selecting a point p from P' in the cell. (C) In Step 2(b)(ii), for each point p' in S, we remove p' from the B+-tree of its corresponding cell and decrease the key of the cell in the max-heap by 1.

Implementation Detail 2 (SSAD algorithm): Note that in Step 1(c) and Step 2(b)(ii), we need to perform the SSAD algorithm [7, 32] which is a best-first search algorithm. There are two versions of this algorithm here, but the major principle is the same for each but with different stopping criteria. The major principle is described as follows. The algorithm performs a search that starts from s and expands its search with the vertex in V which has not been processed and has its minimum geodesic distance d_{min} to s. For each vertex expansion, all points in P on each face expanded together with the vertex are computed with their geodesic distances. Note that we know that for each vertex expansion, all vertices in V with their geodesic distances smaller than d_{min} have been processed. The first version of this algorithm (in Step 1(c)) has an input of a source point s only. For each vertex expansion, the first version of the algorithm checks whether all points in P have been visited. If yes, this algorithm terminates. The second version of this algorithm (in Step 2(b)(ii)) takes as its inputs a source point s and a distance threshold d' (denoting the boundary of the search region starting from s). For each vertex expansion, the second version of the algorithm checks whether d_{min} is larger than d'. If yes, this algorithm terminates. It is worth mentioning that the two versions of SSAD are common practices of the single-source all-destination geodesic shortest distance computation [7, 32]. The time complexity of each of these two versions is $O(\mathcal{N} \log \mathcal{N} + k)$, where $\mathcal{N}$ is the number of vertices in V processed and k is the number of points in P processed.

Implementation Detail 3 (Parent Finding): Recall that in Step 2(b)(iii), we need to find a node in Layer $(i-1)$ to become the parent node O_{parent} of O. Let Y be the set of the centers of all nodes in Layer $(i-1)$. We find this parent node by performing the Single-Source All-Destination (SSAD) algorithm which takes the center of O as an input of the source point and executes until one point in Y is reached. Since O_{parent} has the smallest distance to O among all nodes in Layer $(i-1)$ and Covering Property states that in Layer $(i-1)$ where X denotes a set of all nodes in this layer, the region represented by $\cup_{O' \in X} D(c_{O'}, r_{O'})$ covers all points in P (including the center of node O (i.e., c_O)), we deduce that there exists a node in Layer $(i-1)$, which is O_{parent}, such that c_O is in $D(c_{O_{parent}}, r_{O_{parent}})$. Thus, the distance between c_O and the boundary of the search region of SSAD is at most $r_{O_{parent}} (= 2r_O)$.

Implementation Detail 4 (Set Operation): Variable P' (in Step 2) denoting a set of points could be implemented with a B+-tree data structure. The ID's of all points in P' are maintained in the B+-tree for processing (e.g., point ID insertions and point ID deletions). Thus, constructing P' takes $O(n \log n)$ time and an update operation on P' takes $O(\log n)$ time.

Finally, we analyze the depth h of the partition tree. The following lemma presents the depth of the partition tree.

LEMMA 3.3. $h \leq \log(\frac{\max_{p,q \in P} d_g(p,q)}{\min_{p,q \in P} d_g(p,q)}) + 1$

By our assumption of Section 2 that there are no duplicate POIs, it follows that $\min_{p,q \in P} d_g(p,q)$ is strictly positive. We want to emphasize that the upper bound of h (i.e., $\log(\frac{\max_{p,q \in P} d_g(p,q)}{\min_{p,q \in P} d_g(p,q)})+1$) is a small value in practice. Firstly, in all of our experimental results, h is at most 30. Secondly, even in the extreme case where the minimum distance is one nanometer $(= 10^{-9}\text{m})$ and the maximum distance is the length of the Earth's equator $(\approx 4 \times 10^{7}\text{m})$, Lemma 3.3 yields an upper bound of only 56.

Consider the first component called the *compressed partition tree* which is a variation of the partition tree.

We construct the compressed partition tree $T_{compress}$ based on the original partition tree T_{org} as follows. Firstly, we generate $T_{compress}$ by duplicating T_{org}. Secondly, whenever there is a node O in $T_{compress}$ containing only one child node O_{child}, if there is a parent node O_{parent} of O, then we remove the parent-and-child relationship between O and O_{child} and then the parent of O_{child} is set to O_{parent}. Then, we delete O. We repeat this step iteratively until there is no node in $T_{compress}$ containing only one child. Thirdly, for each leaf node in $T_{compress}$, we set its radius to 0.

Note that each leaf node (containing no child node) is still kept after the above operation since each node removal operation involves a node containing only one child node. Note that for each point p in P, there exists exactly one leaf node whose center is p. Given a point p in P, its *corresponding leaf node*, denoted by O_p, is defined to be the leaf node in the compressed partition tree whose center is p. Besides, given a node O in the compressed partition tree, the layer number of the layer containing O in the compressed partition tree is defined to be the layer number of the layer containing O in the (original) partition tree.

Example 3.4 (Compressed Partition Tree). Consider the partition tree (Figure 5) in Example 3.1. According to the above procedure, since node O_{17} has only one child node (i.e., node O_{12}), we remove the parent-and-child relationship between O_{17} and O_{12} and then we set the parent of O_{12} to node O_{20} (which is the parent of O_{17} in the original partition tree). Then, we remove node O_{17}. After this operation, we do a similar operation for node O_{18} containing only one child node O_{13}. After that, no node in the resulting tree contains only one child. Finally, for each leaf node in the resulting tree, we set its radius (i.e., r_3) to 0. The resulting tree is the compressed partition tree as shown in Figure 6. Note that the layer number of the layer containing node O_{20} is 1 and the layer number of the layer containing node O_{12} is 3 (although the node O_{17} in Layer 2 of the (original) partition tree (which connects O_{12} and O_{20}) is removed).

As shown in the following lemma, the space complexity of the compressed partition tree is $O(n)$ (which is linear to n).

Lemma 3.5. *The compressed partition tree $T_{compress}$ has $O(n)$ nodes.*

Proof. Let m, k denote the number of nodes and edges in $T_{compress}$, respectively. By the construction of $T_{compress}$, $T_{compress}$ has n leaf nodes and every inner node in $T_{compress}$ has at least 2 children. Thus, $T_{compress}$ has $m - n$ inner nodes and at least $2 \cdot (m - n)$ edges. Since $T_{compress}$ is a tree, we obtain that $k = m - 1$. Thus, we obtain that $k = m - 1 \geq 2(m - n)$. Finally, we obtain that $m \leq 2n - 1$. □

3.3 Oracle Component 2: Node Pair Set

Consider the second component of SE called the *node pair set*. Each node pair set $\langle O, O' \rangle$ is associated with the distance $d_g(c_O, c_{O'})$ and a list $\mathcal{L}_M(O, O')$ of M points lying on the shortest path from c_O to c'_O (in the ascending order of their distances to c_O), where M is a user parameter. Before we define this, we give some definitions based on the compressed partition tree which will be used in the node pair set.

Given two nodes O and O' in the compressed partition tree, O and O' are *well-separated* [6] if and only if $d_g(c_O, c_{O'}) \geq (\frac{2}{\epsilon} + 2) \cdot \max\{r, r'\}$ where r is the radius of the enlarged disk of O and r' is the radius of the enlarged disk of O'. Given two nodes O and O' which are well-separated in the compressed partition tree, we say that $\langle O, O' \rangle$ is a *well-separated (node) pair*.

Given a node pair $\langle O, O' \rangle$ and two nodes $\underline{O}$ and $\underline{O}'$ in a tree where (1) $\underline{O}$ is either O or a descendant node of O and (2) $\underline{O}'$ is either O' or a descendant node of O', we say that $\langle O, O' \rangle$ *contains* $\langle \underline{O}, \underline{O}' \rangle$. Note that in our context, a node pair $\langle O, O' \rangle$ has an order. Specifically, even if $\langle O, O' \rangle$ *contains* $\langle \underline{O}, \underline{O}' \rangle$, it is possible that $\langle O', O \rangle$ does not contain $\langle \underline{O}, \underline{O}' \rangle$.

In practice, given two points p and $q \in P$ with their corresponding leaf nodes O_p and O_q in the compressed partition tree, we say that $\langle O, O' \rangle$ *contains* $\langle p, q \rangle$ if $\langle O, O' \rangle$ contains $\langle O_p, O_q \rangle$.

Next, we give a method of generating the node pair set given a compressed partition tree. We maintain a variable S storing a set of node pairs, initialized as $\{\langle O_{root}, O_{root} \rangle\}$ where O_{root} is the root node of the compressed partition tree. At each iteration, we extract a pair $\langle O_i, O_j \rangle$ from S which is not well-separated. Then, we select the node in the pair $\langle O_i, O_j \rangle$ whose radius is larger. Without loss of generality, we assume that O_i is selected and let $C_1, C_2, ..., C_m$ denote its children. Next, we insert $\langle C_1, O_j \rangle$, $\langle C_2, O_j \rangle$, ..., and $\langle C_m, O_j \rangle$ into S. For each $x \in [1, m]$, $\langle C_x, O_j \rangle$ is said to be a pair *generated by* $\langle O_i, O_j \rangle$ and O_i is said to be *split* from $\langle O_i, O_j \rangle$. Note that if O_i and O_j have the same radius, then we select the node with a smaller node ID in the pair $\langle O_i, O_j \rangle$ for processing. We repeat the above procedure until each pair in S is well-separated. Finally, for each node pair $\langle O, O' \rangle$ in S, we compute the shortest geodesic path from c_O to $c_{O'}$ and uniformly sample M points on the path and insert them into the list associated with the node pair. Note that all points in the list are ordered in the ascending order of their distances to c_O.

Let S be the set of node pairs returned by the above procedure. S is called the *node pair set* of *SE*.

In the above procedure, note that whenever we check whether a node pair $\langle O_i, O_j \rangle$ is well-separated, we have to compute the distance between O_i and O_j. Besides, for each node pair $\langle O, O' \rangle$ in S, it is needed to find the shortest geodesic path $\mathcal{P}$ from c_O to $c_{O'}$. Then, we simply uniformly sample M vertices from the path $\mathcal{P}$ and insert them all into $\mathcal{L}_M(O, O')$. Note that if $\mathcal{P}$ does not have M vertices, we insert all vertices on the path into $\mathcal{L}_M(O, O')$. Later in Section 3.6 as a part of the oracle construction, we will explain how we compute this distance and the path efficiently.

The following theorem shows a key property of the node pair set generated, namely the *unique node pair match property*.

Theorem 3.6. *Let S be the node pair set of* SE. *Each node pair in S is a well-separated pair and for any two points p and q in P, there exists exactly one node pair $\langle O, O' \rangle$ in S containing $\langle p, q \rangle$ and the distance associated with this node pair is an ϵ-approximate distance of $d_g(p, q)$.*

Next, we present the following theorem showing that there are $O(\frac{nh}{\epsilon^{2\beta}})$ node pairs considered in the procedure of generating the node pair set (which is linear to n) where β is a real number and is in the range from 1.5 and 2 in practice.

Theorem 3.7. *There are only $O(\frac{nh}{\epsilon^{2\beta}})$ node pairs considered in the procedure of generating the node pair set and thus there are $O(\frac{nh}{\epsilon^{2\beta}})$ in the node pair set of* SE.

Finally, we adopt a standard hashing technique, namely the *perfect hashing scheme* [8], to index all node pairs in the node pair set of *SE*. The hashing technique takes a linear space and requires a linear preprocessing time in expectation in terms of the number of the node pairs in the node pair set of *SE*. Given two nodes O and O' in the compressed partition tree, we could check whether there exists a node pair $\langle O, O' \rangle$ in the node pair set of *SE* in constant time and if so, it could also return the associated geodesic distance $d_g(O, O')$ in constant time.

3.4 Distance Query Processing

Next, we present how we use our distance oracle SE for a distance query with a source point $s \in P$ and a destination point $t \in P$.

We first present one naive method, whose time complexity is $O(h^2)$, for this distance query. Next, we present an efficient algorithm whose time complexity is $O(h)$.

Naive Method: Before we introduce the naive method, we give some notations first. Let O_{root} be the root node of the compressed partition tree. By our notation convention, we know that O_s denotes the corresponding leaf node of point s in the compressed partition tree and O_t denotes the corresponding leaf node of point t in the compressed partition tree. Let A_s be the array of size $h+1$ where $A_s[i]$ is equal to the node in Layer i along the path from O_s to O_{root} in the compressed partition tree if there exists a node in Layer i and is equal to $\emptyset$ otherwise for each $i \in [0, h]$. We have another notation A_t which has a definition similar to A_s and involves the path starting from O_t instead of O_s. We denote the Cartesian product between the set of all nodes in A_s and the set of all nodes in A_t by $A_s \times A_t$. It is easy to have the following observation from Theorem 3.6: there exists exactly one pair $\langle O, O' \rangle$ in $A_s \times A_t$ such that $\langle O, O' \rangle$ contains $\langle s, t \rangle$ and $\langle O, O' \rangle$ is in the node pair set of our SE.

Based on this observation, we have the following naive method for a distance query. Firstly, we find a leaf node O_s and a leaf node O_t. Then, we construct array A_s (A_t) by traversing from O_s (O_t) to O_{root}. Secondly, for each node $O \in A_s$ and each node $O' \in A_t$, we check whether node pair $\langle O, O' \rangle$ is in the node pair set of our SE. If so, we return the distance associated with $\langle O, O' \rangle$. Otherwise, we continue to check the next node pair.

Note that by this observation, the above naive method must return one distance value (associated with one node pair) at the end.

The correctness of the naive method (i.e., the ϵ-approximation) comes naturally from Theorem 3.6.

It is easy to verify that the time complexity of the naive method is $O(h^2)$ since the first step takes $O(h)$ time and the second step takes $O(h^2)$ time (because the second step involves $O(h^2)$ node pairs and each node pair requires to be checked with its existence in the node pair set of our SE in $O(1)$ time using the perfect hashing scheme).

Efficient Method: Next, we will present our efficient algorithm for the distance query which takes $O(h)$ time. Before we present the algorithm, we give some concepts first.

Let $Layer(O)$ be the layer number of the layer containing node O.

We categorize node pairs $\langle O, O' \rangle$ into one of three types. A node pair $\langle O, O' \rangle$ is said to be a *same-layer node pair* if O has the same layer as O' in the compressed partition tree (i.e., $Layer(O) = Layer(O')$). A node pair $\langle O, O' \rangle$ is said to be a *first-higher-layer node pair* if O has a higher layer than O' in the compressed partition tree (i.e., $Layer(O) < Layer(O')$). A node pair $\langle O, O' \rangle$ is said to be a *first-lower-layer node pair* if O has a lower layer than O' in the compressed partition tree (i.e., $Layer(O) > Layer(O')$).

Consider the compressed partition tree as shown in Figure 6. The node pair $\langle O_{14}, O_{15} \rangle$ is a same-layer node pair. The node pair $\langle O_{14}, O_7 \rangle$ is a first-higher-layer node pair and the node pair $\langle O_6, O_{15} \rangle$ is a first-lower-layer node pair.

By definition, in a same-layer node pair $\langle O, O' \rangle$, both node O and node O' are in the same layer. We know that in a first-higher-layer node pair $\langle O, O' \rangle$, since node O has a higher layer than node O', we know that there exists a layer higher than the layer containing node O', and thus we deduce that O' has a parent node in the compressed partition tree. With the following lemma, interestingly, we know that the layer containing the parent of node O' is equal to or higher than the layer containing node O. We could have a similar conclusion for a first-lower node pair.

Lemma 3.8. *Consider a node pair $\langle O, O'\rangle$ in the node pair set of our* SE. *If $\langle O, O'\rangle$ is a first-higher-layer node pair, then the layer containing the parent of node O' is equal to or higher than the layer containing node O. If $\langle O, O'\rangle$ is a first-lower-layer node pair, then the layer containing the parent of node O is equal to or higher than the layer containing node O'.*

Consider the compressed partition tree as shown in Figure 6. The error parameter ϵ is set to 2. Note that for illustration purpose, this error parameter is set to 2 but in practice, it should be set to a smaller value (e.g., 0.1) as what we did in our experimental studies. The node pairs $\langle O_{14}, O_7\rangle$ and $\langle O_{16}, O_{12}\rangle$ are both first-higher-layer node pairs in the node pair set of our SE. The parent of O_7 (O_{12}) is O_{15} (O_{20}). The layer containing O_{15} is the same as that containing O_{14} and the layer containing O_{20} is higher than that containing O_{16}. Similar illustrations could be made to the two first-lower-layer node pairs in the node pair set of our SE, namely $\langle O_6, O_{15}\rangle$ and $\langle O_{13}, O_{20}\rangle$, in a symmetric way.

Let $parent(O)$ be the parent of node O in the compressed partition tree.

With Lemma 3.8, we have the following observation.

Observation 1. *Consider a node pair $\langle O, O'\rangle$ in the node pair set of our* SE. *If $\langle O, O'\rangle$ is a first-higher-layer node pair, then $Layer(parent(O')) \leq Layer(O) < Layer(O')$. If $\langle O, O'\rangle$ is a first-lower-layer node pair, then $Layer(parent(O)) \leq Layer(O') < Layer(O)$.*

Consider the compressed partition tree as shown in Figure 6. The node pairs $\langle O_{14}, O_7\rangle$ and $\langle O_{16}, O_{12}\rangle$ are both first-higher-layer node pairs in the node pair set of our SE. $parent(O_7)$ ($parent(O_{12})$) is O_{15} (O_{20}). It is clear that $Layer(parent(O_7)) \leq Layer(O_{14}) < Layer(O_7)$ and $Layer(parent(O_{12})) \leq Layer(O_{16}) < Layer(O_{12})$. Similar illustrations could be made to the two first-lower-layer node pairs in the node pair set of our SE, namely $\langle O_6, O_{15}\rangle$ and $\langle O_{13}, O_{20}\rangle$, in a symmetric way.

Based on Observation 1, we give the major idea why we could have an efficient algorithm. Note that the naive method requires that $O(h^2)$ node pairs should be enumerated. However, our efficient method just needs to enumerate $O(h)$ node pairs. Specifically, our efficient method involves three steps. Roughly speaking, the first step handles same-layer node pairs in $A_s \times A_t$, the second step handles first-higher-layer node pairs in $A_s \times A_t$, and the third step handles first-lower-layer node pairs in $A_s \times A_t$.

Specifically, the first step checks whether there exists a node O in A_s and a node O' in A_t such that $\langle O, O'\rangle$ is a same-layer node pair and $\langle O, O'\rangle$ is in the node pair set of SE. If there exists such a node pair $\langle O, O'\rangle$, we return the distance associated with $\langle O, O'\rangle$. This can be done in $O(h)$ time by linearly scanning both arrays A_s and A_t from index 0 through h and checking whether $\langle A_s[i], A_t[i]\rangle$ is in the node pair set of SE where $i \in [0, h]$ (note that $\langle A_s[i], A_t[i]\rangle$ is a same-layer node pair). The second step is to check whether there exists a node N in A_s and a node N' in A_t such that $\langle O, O'\rangle$ is a first-higher-layer node pair and $\langle O, O'\rangle$ is in the node pair set of SE. If there exists such a node pair $\langle O, O'\rangle$, we return the distance associated with $\langle O, O'\rangle$. This can be done in $O(h)$ time by the following sub-steps. For each $i \in [1, h]$, if $A_t[i] \neq \emptyset$, then we obtain the layer number j of the layer containing the parent of $A_t[i]$ (in $O(1)$ time) and, for each $k \in [j, i)$, check whether $\langle A_s[k], A_t[i]\rangle$ is in the node pair set of SE (in $O(j - i)$ time) (note that it is sufficient to scan to check $\langle A_s[j], A_t[i]\rangle, \langle A_s[j+1], A_t[i]\rangle, ..., \langle A_s[i-1], A_t[i]\rangle$ for one particular node $A_t[i]$ in A_t based on Observation 1). It is easy to verify that the second step takes $O(h)$ time since we can scan $O(h)$ elements in A_s and $O(h)$ elements in A_t. The third step is similar to the second step, but

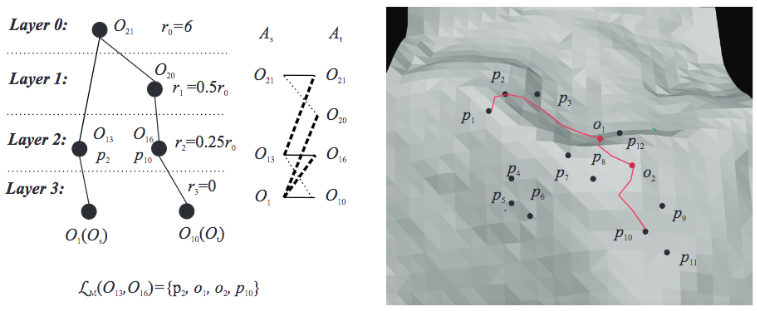

Fig. 8. An Example of Path Query Processing

this step focuses on the first-lower-layer node pairs instead of the first-higher-layer node pairs. Details are skipped here since similar descriptions are applied. Thus, the overall time complexity of the efficient method is $O(h)$.

Example 3.9 (Distance Query Processing). The error parameter ϵ is set to 2. Consider the example as shown at the left hand side in Figure 7, where O_s is O_1 and O_t is O_{10}. It shows all edges and all nodes along the path from the leaf node O_1 with its center p_1 to the root node and the path from the leaf node O_{10} with its center p_{10} to the root node. The pair $\langle O_{13}, O_{16}\rangle$ containing $\langle O_1, O_{10}\rangle$ is the pair in the node pair set of SE. *In this example, $A_s = [O_{21}, \emptyset, O_{13}, O_1]$ and $A_t = [O_{21}, O_{20}, O_{16}, O_{10}]$. Consider the figure at the right hand side in Figure 7. All node pairs processed in the query processing algorithm are shown in the form of node pairs connected by lines (which are solid lines, thin dashed lines and thick dashed lines). Specifically, each node pair connected by a solid line is a same-layer node pair processed. Each node pair connected by a thin dashed line is a first-higher-layer node pair processed. Each node pair connected by a thick dashed line is a first-lower-layer node pair processed. Our query algorithm checks all the three types of node pairs. When one of the node pairs processed is in the node pair set of* SE, *we return the distance associated with this node pair.*

It is worth mentioning that the total number of lines in this figure corresponds to the greatest number of node pairs processed, which is equal to $O(h)$ instead of $O(h^2)$ (denoting the total number of lines in a complete bipartite graph between A_s and A_t). Thus, the query step is very efficient.

It is easy to verify that the distance returned by the efficient method is ϵ-approximate based on Theorem 3.6.

3.5 Path Query Processing

To find the shortest geodesic path from a given source point s to a given destination point t, we first find the node pair $\langle O, O'\rangle$ in the second component of SE, the node pair set, containing O_s and O_t, respectively, with the same method in the distance query processing algorithm. Recall that the list $\mathcal{L}_M(O, O')$ is part of the second component, node pair set, in our oracle which is defined in Section 3.3. Then, for each adjacent pair o_i, o_{i+1} in the list $\mathcal{L}_M(O, O') = (o_1, o_2, ..., o_M)$, we perform a SSAD algorithm to find the shortest geodesic path $\Pi_g(o_i, o_{i+1})$ from o_i to o_{i+1}, where $i \in [1, M-1]$. Then, we perform the SSAD algorithm to find the following four geodesic shortest paths which are $\Pi_g(s, c_O)$, $\Pi_g(c_O, o_1)$, $\Pi_g(o_M, c_{O'})$ and $\Pi_g(c_{O'}, t)$. Finally, we concatenate all the paths found in the corresponding order and return the final result which is $\Pi_g(s, c_O)\cdot\Pi_g(c_O, o_1)\cdot\Pi_g(o_1, o_2)\cdot ......\cdot\Pi_g(o_{M-1}, o_M)\cdot\Pi_g(o_M, c_{O'})\cdot\Pi_g(c_{O'}, t)$, where $\cdot$ denotes the path concatenation operation.

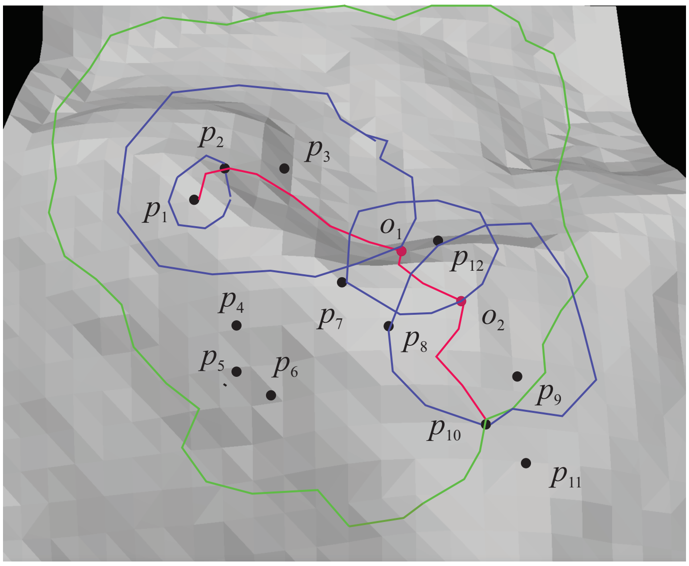

Fig. 9. Visited Vertices of Path Query Processing

Example 3.10 (Path Query Processing). The error parameter ϵ is set to 2. Consider the example as shown at the left hand side in Figure 8, where the source is p_1 and the destination is p_{10} and thus, O_s is O_1 and O_t is O_{10}. It shows all edges and all nodes along the path from the leaf node O_1 with its center p_1 to the root node and the path from the leaf node O_{10} with its center p_{10} to the root node. Our algorithm first performs a distance query processing and found that the pair $\langle O_{13}, O_{16}\rangle$ containing $\langle O_1, O_{10}\rangle$ is the pair in the node pair set of SE. *The list $\mathcal{L}_M(O_{13}, O_{16}) = (p_2, o_1, o_2, p_{10})$ which is constructed in the preprocessing phase contains the centers (i.e., p_2 and p_{10}) of O_{13} and O_{16} and two points (i.e., o_1 and o_2) uniformly sampled from the shortest path from p_2 to p_{10}. Then, we simply perform SSAD algorithm to find the following shortest paths, i.e., $\Pi_g(p_1, p_2)$, $\Pi_g(p_2, o_1)$, $\Pi_g(o_1, o_2)$, $\Pi_g(o_2, p_{10})$ and $\Pi_g(p_{10}, p_{10})$. Since $\Pi_g(p_{10}, p_{10})$ is $\emptyset$, we simply concatenate $\Pi_g(p_1, p_2)$, $\Pi_g(p_2, o_1)$, $\Pi_g(o_1, o_2)$ and $\Pi_g(o_2, p_{10})$ and return the concatenated path. Consider the figure at the right hand side in Figure 8. It shows $\Pi_g(p_1, p_2)$, $\Pi_g(p_2, o_1)$, $\Pi_g(o_1, o_2)$ and $\Pi_g(o_2, p_{10})$ and the red line is the their concatenated path which is the final result.*

It is worth mentioning that the running time of the SSAD algorithm is quadratic to the number of vertices visited and thus, the running time of all the SSAD algorithms in our algorithm (who returns the sub-paths on $\Pi_g(s, t)$) is significantly smaller than that of the SSAD algorithm with s as the source and t as the destination directly. Consider the example as shown in Figure 9. The large disk in green contains all the vertices visited by the SSAD algorithm with s as the source and t as the destination. The four small disks in blue contains the visited vertices by the four SSAD algorithms invoked by our algorithm. As could be observed from the figure, the number of visited vertices is apparently smaller than that of the SSAD algorithm with s as the source and t as the destination. Together with the fact that the running time of the SSAD algorithm is quadratic to the number of vertices visited, the running time of our algorithm is significantly smaller.

3.6 Oracle Construction

In this section, we first present a naive method of constructing SE and then present an efficient method of constructing SE.

Naive Method: We first present a naive method of constructing SE. First, we build a partition tree T_{org}. Then, we build a compressed partition tree $T_{compress}$ based on T_{org} and delete T_{org}. Next, we follow the procedure described in Section 3.3 to generate all node pairs for the node pair set. Note that for each node pair considered (constructed in S, resp.), we have to compute the distance (the shortest path, resp.) between the centers of the two nodes in the node pair. In the naive method, for each node pair considered, we perform the SSAD algorithm, which takes the center of one node in the node pair as an input of the starting point and performs the search until it reaches the center of the other node in the node pair.

We proceed to analyze the running time of the naive method. It takes $O(nhN\log^2 N)$ to build T_{org} since there are $O(nh)$ nodes in T_{org} and each node has to perform the SSAD algorithm which takes the center of this node as an input of the starting point and performs the search until it reaches a certain radius in $O(N\log^2 N)$ time. It takes $O(nh)$ time to construct $T_{compress}$, since $T_{compress}$ could be constructed with a postorder traversal of T_{org} and there are $O(nh)$ nodes in T_{org}. For each node pair $\langle O, O'\rangle$ generated, we need to perform the SSAD algorithm which takes the center of one node in the node pair as an input of the starting point and performs the search until it reaches the center of the other node in the node pair to compute $d_g(c_O, c_{O'})$ and also $\mathcal{L}_M(O, O')$. Thus, the total running time of generating the node pair set is $O(\frac{nh}{\epsilon^{2\beta}}N\log^2 N)$. In conclusion, the total running time of the naive method of constructing SE is $O(\frac{nhN\log^2 N}{\epsilon^{2\beta}})$.

Efficient Method: Since the naive method takes $O(\frac{nhN\log^2 N}{\epsilon^{2\beta}})$ time to construct the SE distance oracle, which is very costly, we propose an efficient algorithm of constructing SE next. The major reason why the naive method is slow is that in the naive method, for each node pair considered in the procedure described in Section 3.3, the naive method has to perform an expensive SSAD algorithm, and thus the number of times that the SSAD algorithm is called is equal to the number of node pairs considered. However, we will present an efficient algorithm which could reduce the number of times that the SSAD algorithm is called from the total number of node pairs considered to the total number of nodes in the (original) partition tree by using a new concept called an *enhanced node pair* (which is a node pair involving two nodes in the same layer of the (original) partition tree and satisfying a condition) (to be introduced later). Specifically, the efficient method has two major differences from the naive method. The first difference is that the efficient method includes an additional (pre-computation) step of computing the distance between the two nodes involved in each possible enhanced node pair. Although there are $O(hn^2)$ possible enhanced node pairs and we have to compute the distances of these pairs, the total number of times that the SSAD algorithm is called in this additional step is just equal to the total number of nodes in the (original) partition tree (which is $O(hn)$). The second difference is that the efficient method finds the distance of each node pair $\langle O, O'\rangle$ considered in the procedure described in Section 3.3 by searching one of the "pre-computed" distances of the enhanced node pairs containing the node pair $\langle O, O'\rangle$ and assigning this distance (of the enhanced node pair found) to the distance of the node pair $\langle O, O'\rangle$ (instead of performing the expensive SSAD algorithm). Note that the time complexities of both the search operation and the assignment operation are $O(h)$ (to be shown later), which is much lower than the time complexity of the SSAD algorithm (i.e., $O(N \log^2 N)$). Later, we will show that for each node pair $\langle O, O' \rangle$ considered in the procedure described in Section 3.3, there exists one enhanced node pair containing the node pair $\langle O, O' \rangle$, which is a key to the efficiency of the efficient method.

Before we present the efficient method, we define the concept of the *enhanced node pair*. Given two nodes O and O' in the (original) partition tree, $\langle O, O' \rangle$ is said to be an *enhanced node pair* if O and O' are in the same layer of the (original) partition tree and $d_g(O, O') < l \cdot r_O$ where $l = \frac{8}{\epsilon} + 10$. Note that l is about 4 times the well-separated factor (i.e., $\frac{2}{\epsilon} + 2$). The ratio of 4 ($= 2 \times 2$) is split two parts. The first part (i.e., a ratio of 2) comes from the radius of the *enlarged* disk of a node O (defined in the definition of the *well-separated pair*) which is two times the radius of node O. The second part (i.e., another ratio of 2) comes from our design.

With the definition of the *enhanced node pair*, we give the following lemma which is used in our efficient method.

Lemma 3.11. *Consider a node pair $\langle O, O' \rangle$ considered in the procedure described in Section 3.3. There exists an enhanced node pair $\langle \overline{O}, \overline{O}' \rangle$ such that (1) $\langle \overline{O}, \overline{O}' \rangle$ contains $\langle O, O' \rangle$, (2) $c_{\overline{O}} = c_O$ and (3) $c_{\overline{O}'} = c_{O'}$.*

The major idea why we can design an efficient method compared with the naive method is that the efficient method is designed based on Lemma 3.11 using the concept of the *enhanced node pair*.

We present the efficient algorithm of constructing SE as follows.

- **Step 1 (Tree Construction):** We build the partition tree T_{org} and a compressed partition tree $T_{compress}$ based on T_{org}. $T_{compress}$ just constructed becomes the first component of SE.
- **Step 2 (Enhanced Edge Creation):** We insert all possible *enhanced edges* into T_{org}. Specifically, for any two nodes O and O' in the *same* layer of the (original) partition tree T_{org}, if $\langle O, O' \rangle$ is an enhanced node pair, then we add an edge connecting them. We call an edge added in this step an *enhanced edge*. We associate a distance to each enhanced edge added. Specifically, for each enhanced edge connecting O and O', we associate the distance between these two nodes (i.e., $d_g(c_O, c_{O'})$) and the list $\mathcal{L}_M(O, O')$ with this edge. To construct all the enhanced edges *together*, for each node O in the partition tree, we perform the SSAD algorithm which takes c_O as an input of the source point and performs the search until the disk $D(c_O, l \cdot r_O)$ is totally expanded.
- **Step 3 (Perfect Hash Construction):** We insert all enhanced edges into the perfect hash [8] (with an oracle building time and a space cost which are linear to the total number of edges in expectation).
- **Step 4 (Node Pair Set Generation):** We generate the node pair set, the second component of SE, using T_{org} added with enhanced edges. Specifically, we follow the procedure described in Section 3.3 to generate all node pairs for the node pair set. However, we present a detailed implementation of how to compute $d_g(c_O, c_{O'})$ for each node pair $\langle O, O' \rangle$ generated in the procedure. For each node pair $\langle O, O' \rangle$ generated, we find an enhanced edge connecting a node $\overline{O}$ and a node $\overline{O}'$ in T_{org} such that (1) $\langle \overline{O}, \overline{O}' \rangle$ is an enhanced node pair, (2) $\langle \overline{O}, \overline{O}' \rangle$ contains $\langle O, O' \rangle$, (3) $c_{\overline{O}} = c_O$ and (4) $c_{\overline{O}'} = c_{O'}$. (Note that by Lemma 3.11, there exists such an enhanced edge.) This step of finding an enhanced edge can be done in $O(h)$ time by
  - (1) first obtaining c_O from O and $c_{O'}$ from O' (in $O(1)$ time),

- (2) then accessing the corresponding leaf node $\underline{O}$ of c_O and the corresponding leaf node $\underline{O}'$ of $c_{O'}$ (in $O(1)$ time),
- (3) traversing both the path $\mathcal{P}$ from $\underline{O}$ to the root node and the path $\mathcal{P}'$ from $\underline{O}'$ to the root node together starting from Layer h to Layer 0 to check whether the node $\overline{O}$ being traversed along $\mathcal{P}$ and the node $\overline{O}'$ being traversed along $\mathcal{P}'$ (in the same layer) have their node pair $\langle \overline{O}, \overline{O}' \rangle$ found in the perfect hash (in $O(h)$ time), and
- (4) returning the enhanced node edge connecting $\overline{O}$ and $\overline{O}'$ (if these two nodes have their node pair $\langle \overline{O}, \overline{O}' \rangle$ found in the perfect hash) (in $O(1)$ time).

Then, the distance and the list $\mathcal{L}_M(\overline{O}, \overline{O}')$ associated with this enhanced edge $\langle \overline{O}, \overline{O}' \rangle$ correspond to the distance (i.e., $d_g(c_O, c_{O'})$) and the list $\mathcal{L}_M(O, O')$ that we want.

3.7 Theoretical Analysis

Before analyzing *SE*, we introduce a well-known concept called the *largest capacity dimension* originally defined on a metric space [16, 26]. For the sake of space, the definition and the discussion of the *largest capacity dimension* could be found in the appendix. In the appendix, we show that in an extreme case where the terrain surface is a 2D plane, the *largest capacity dimension* β is at most 1.3. In a general case, β is a little bit larger than 1.3 (since the terrain surface could be regarded as a 2D surface with some fluctuations in terms of height).

Our experimental results show that the *largest capacity dimension* β of the terrain surface that we considered is between 1.3 and 1.5.

Then, we present the oracle building time, oracle size, query time and distance error bound of our *SE* in the following theorem.

Theorem 3.12. *The oracle building time, oracle size, distance query time and distance error bound of* SE *are* $O(\frac{N \log^2 N}{\epsilon^{2\beta}} + nh \log n + \frac{nh}{\epsilon^{2\beta}})$, $O(\frac{nh}{\epsilon^{2\beta}} \cdot M)$, $O(h)$ *and* ϵ, *respectively.*

We also present the approximate ratio of the path returned by *SE* in the following theorem.

Theorem 3.13. *The length of the path between two POIs, namely* s *and* t, *returned by* SE *is at most* $(1 + 2 \cdot \epsilon)$ *times the length of the shortest geodesic path from* s *to* t.

4 PROCESSING OTHER PROXIMITY QUERIES

In this section, we present how to process other proximity queries with our distance oracles and compare them with some existing studies. The other proximity queries studied in this paper include (1) the k nearest neighbor and farthest neighbor query and (2) the top-k closest pair and farthest pair query. They both have wide applications and here are some examples.

Applications of the k nearest neighbor and farthest neighbor query. (I) In the geographic information system (GIS), it is important for hikers to find k nearest or farthest POIs (i.e., the candidates of the hiking destinations) from their own locations in a given region (e.g., Zion National Park in U.S.) to plan a proper hiking route [38], where k nearest POIs correspond to easy hiking trials and k farthest ones correspond to challenging trials. (II) In the online 3D virtual game such as PokemanGo, each user may be interested in the k nearest or farthest portals/landmarks in a specific area to visit, where the k nearest neighbors corresponds to the portals in their vicinity and k farthest neighbors are the distant ones for the better explorations. (III) In the computer graphics and vision, each 3D model contains a set of

3D points and the k nearest neighbor and farthest neighbor search serves as a fundamental query in many computer graphics algorithms such as denoising and repairing [21, 22].

Applications of the top-k closest pair and farthest pair query. (I) In the geographic information system (GIS), the top-k closest pair and farthest pair query is a fundamental building block in the spatial join operation [9, 10] in the mountainous areas. (II) In the spatial recommendation, the k closest pair query is used to recommend resort and hotel pairs for the tourists so that they could find a resort and a corresponding hotel for the tour [9] in a mountainous regions such as Greece and Hong Kong. Besides, in an application where several source points and destination points are given and a hiking trial is to be designed, the k farthest pair query is used to recommend the possible candidates for the hiking trial to be developed since the distant pair could fully utilize the terrain surface. (III) In the spatial data mining, the inner-cluster distance and intra-cluster distance computation are frequently invoked for many clustering algorithms such as k-means, where the top-k closest pair and farthest pair query is required as a building block for the distance computation.

For the completeness, we studied both the monochromatic version and bichromatic version of the two queries mentioned above. The major idea of the query algorithm is to linearly scan each pair in the node pair set of SE and select the corresponding pairs according to the query. The following two subsections further detail the query algorithm and their accuracy and time complexity. In the two subsections, we adopt a new parameter ϵ' and let $\epsilon' = \frac{2\epsilon}{1-\epsilon}$, where ϵ is the error parameter of the distance oracle. Given an approximate algorithm, its approximate error bound is c if its approximate ratio is $1 + c$, where c is a non-negative real number.

4.1 k Nearest-Neighbor and k Farthest-Neighbor Query

We first present the bichromatic kNN query: given a point p in P and a set P' of points where $P' \subseteq P$, return a set of k points, says $X = \{q_1, q_2, ..., q_k\}$, which are k points in P' nearest to p, in other words, $\forall i \in [1, k], q_k \in P'$ and $\max_{i\in[1,k]} d_g(p, q_i) \leq \min_{o\in P'\setminus X} d_g(p, o)$.

To process the above query, we perform a distance query between p and any point in P' with the assistance of our distance oracle. Let $\widetilde{d_g}(p, q)$ denote the distance between p and q returned by the distance query. Finally, we return the list $KLIST'$ containing k points in P' such that $\max_{q\in KLILST'} \widetilde{d_g}(p, q) \leq \min_{q\in P'\setminus KLIST'} d_g(p, q)$. Thus, it takes $O(nt)$ time to process the query, where t is the distance query time. Since the distance query time is $O(h)$, the running time of our query algorithm is $O(hn)$. The following lemma shows the correctness of our algorithm.

Lemma 4.1. *The approximate ratio of our kNN algorithm is $1 + \epsilon'$, where $\epsilon' = \frac{2}{1-\epsilon}$ is the appro. error of our distance oracle algorithm and ϵ is the error parameter of the distance oracle.*

Proof. For the sake of space, the proof could be found in Appendx B. □

The monochromatic k NN query is a special case of the bichromatic kNN query when $P' = P$. The monochromatic k nearest neighbor query on a terrain surface [11, 13, 39] was studied. The experimental results of [39] showed that [39] is more efficient than [11, 13] in practice. Thus, we do not consider [11, 13] as our baseline. [39] proposed an index called the *Surface Index* (*SI*) and an algorithm for the monochromatic kNN query on a terrain surface. Its worst-case query time is $O(N^2)$ which is worse than ours.

Similar to the monochromatic and bichromatic kNN query, we can process the monochromatic and bichromatic approximate k farthest neighbor (kFN) query in the same method except that we finally return the k farthest points instead of the k nearest ones. The following lemma presents the accuracy of our algorithm.

Lemma 4.2. *The approximate ratio of our kFN algorithm is $1+\epsilon'$, where $\epsilon' = \frac{2}{1-\epsilon}$ is the appro. error of our distance oracle algorithm and ϵ is the error parameter of the distance oracle.*

Proof. For the sake of space, the proof could be found in Appendx B. □

4.2 Top-k Closest-Pair/Farthest-Pair Query Processing

We present the top-k closest-pair query (kCP) and top-k farthest-pair (kFP) query in this section.

We first present the top-k bichromatic closest-pair (BCP) query which is formulated as follows: Given two sets P_1 and P_2 where $P_1, P_2 \subset \overline{P}$ and $P_1 \cap P_2 = \emptyset$, return a list X such that $X \subseteq P_1 \times P_2$ and $\max_{<p,q> \in X} d_g(p,q) \leq \min_{<p,q> \in P_1 \times P_2 \setminus X} d_g(p,q)$, where $\times$ denotes the Cartesian product between two sets.

We denote the node pair set in our oracle as $\mathcal{O}$. Given a node O in our compressed partition tree, we denote $RS(O)$ as the set containing the centers of all the leaf nodes in the subtree rooted at O. To process the top-k BCP query, we do a linear scan on $\mathcal{O}$ and find out a list $\mathcal{L}$ containing pairs from $\mathcal{O}$, each pair $\langle A, B\rangle$ in which satisfies that $RS(A) \cap P_1 \neq \emptyset, RS(B) \cap P_2 \neq \emptyset$. Then, we sort all the pairs in the list $\mathcal{L}$ in the ascending order of the distance stored within them. Next, we initialize an integer variable x to be k and initialize an list KCP' to be an empty list. We proceed to process the sorted list $\mathcal{L}$ in several iterations. At each iteration, we extract the head $\langle O, O'\rangle$ of the list. If $|RS(O) \cap P_1| \cdot |RS(O') \cap P_2| \leq x$, we insert all the pairs in $(RS(O) \cap P_1) \times (RS(O') \cap P_2)$ into KCP' and decrease x by $|RS(O) \cap P_1| \cdot |RS(O') \cap P_2|$. Otherwise, we insert x pairs randomly selected from $(RS(O) \cap P_1) \times (RS(O') \cap P_2)$ into KCP' and output the KCP'. In order to check the set intersection efficiently, we can first mark all the nodes O such that $RS(O)$ intersects with P_1 and P_2 separately by a postorder traversal of the compressed partition tree. By Lemma 3.5, the size of the compressed partition tree is $O(n)$. Thus, this traversal takes $O(n)$ time. A linear scan of $\mathcal{O}$ takes $O(|\mathcal{O}|)$ time. Since k is very small compared with n, the following operations are dominated by the linear scan in terms of the running time. We conclude that the overall query time is $O(|\mathcal{O}|)$ which is equal to $O(\frac{n}{\epsilon^{2\beta}})$.

Lemma 4.3. *The approximate ratio of our top-k BCP query algorithm is $1+\epsilon'$, where $\epsilon' = \frac{2}{1-\epsilon}$ is the appro. error of the distance oracle algorithm and ϵ is the error parameter of the distance oracle.*

Proof. For the sake of space, the proof could be found in Appendix B. □

Let PP denote the set $\{\langle a,b\rangle | \langle a,b\rangle, \langle b,a\rangle \in P \times P, a \neq b\}$, where $\times$ denotes the Cartesian product. We proceed to consider the top-k monochromatic Closest-Pair (MCP) query which is formulated as follows: Given the set P of all POIs, return a list X of k point pairs such that $X \subseteq PP$ and $\max_{\langle p,q\rangle \in X} d_g(p,q) \leq \min_{\langle p,q\rangle \in PP \setminus X} d_g(p,q)$.

It could be checked that the processing algorithm and analysis of the top-k BCP query applies to the top-k MCP query if we replace P_1, P_2 by P and replace $P \times P$ by $\{\langle O, O'\rangle | \langle O, O'\rangle, \langle O', O\rangle \in P \times P\}$. It still guarantees the same approximate ratio. Its running time is $O(nh)$ and its approximate ratio (resp. appro. error) is $1+\epsilon' = \frac{1+\epsilon}{1-\epsilon}$ (resp. $\epsilon' = \frac{2\epsilon}{1-\epsilon}$).

Similar to the monochromatic and bichromatic top-k closest pair query, we could process the monochromatic and bichromatic top-k farthest pair (FP) query by using the same method except that we maintain the k farthest pairs instead of k closest ones. The following lemma shows the accuracy of our FP query processing algorithm.

Algo.	Oracle Building Time	Oracle Size	Shortest Distance Query Time
SP-Oracle [15]	$O(\frac{N}{\sin(\theta)\cdot\epsilon^2}\log^3(\frac{N}{\epsilon})\log^2\frac{1}{\epsilon})$	$O(\frac{N}{\sin(\theta)\cdot\epsilon^{1.5}}\cdot\log^2(\frac{N}{\epsilon})\log^2\frac{1}{\epsilon})$	$O(\frac{1}{\sin(\theta)\cdot\epsilon}\log\frac{1}{\epsilon}+\log\log(N+n))$
SE(Naive)	$O(\frac{nhN\log^2 N}{\epsilon^{2\beta}})$	$O(\frac{nh}{\epsilon^{2\beta}})$	$O(h^2)$
K-Algo [24]	–	–	$O(\frac{l_{max}^3 N}{(l_{min}\cdot\epsilon\cdot\sqrt{1-\cos\theta})^3}+\frac{l_{max}\cdot N}{\epsilon\cdot l_{min}\cdot\sqrt{1-\cos\theta}}\log(\frac{l_{max}\cdot N}{\epsilon\cdot l_{min}\cdot\sqrt{1-\cos\theta}}))$
SE	$O(\frac{N\log^2 N}{\epsilon^{2\beta}}+nh\log n+\frac{nh}{\epsilon^{2\beta}})$	$O(\frac{nh}{\epsilon^{2\beta}})$	$O(h)$

Table 1. Comparison of Different Methods for Shortest Distance Query with Error Bound ϵ (where $\beta \in [1.3, 1.5]$ and $h < 30$ in practice)

Dataset	No. of Vertices	Resolution	Region Covered	No. of POIs
BH	1.4M	10 meters	$14km \times 10km$	4k
EP	1.5M	10 meters	$10.7km \times 14km$	4k
SF	170k	30 meters	$14km \times 11.1km$	51k

Table 2. Dataset Statistics

Lemma 4.4. *The approximate ratio of our top-k FP query algorithm is $1+\epsilon'$, where $\epsilon' = \frac{2}{1-\epsilon}$ is the appro. error of the distance oracle algorithm and ϵ is the error parameter of the distance oracle.*

Proof. For the sake of space, the proof could be found in Appendix B. □

5 RELATED WORK AND BASELINES

In this section, we present the related work and baseline methods in Section 5.1 and Section 5.2, respectively.

5.1 Related Work

The existing studies of finding the *exact* geodesic distance and path between two vertices are [7, 32] and [45]. Their time complexities are $O(N^2 \log N)$, $O(N^2)$, $O(N \log^2 N)$ and $O(N^2 \log N)$, respectively, which are impractical even on moderate terrain data.

Motivated by the intrinsic expensive cost of computing exact geodesic distances, many existing studies focus on computing *approximate* geodesic distances and paths [24, 25, 30]. In [30], the authors studied the problem of finding an approximate geodesic shortest path which satisfies a *slope constraint*. In [25], the authors proposed an algorithm for finding a geodesic path between two points satisfying a condition on the terrain surface and computing the lower and upper bounds of the geodesic shortest distance based on the length of the path found, but the gap between the bounds depends on the structure of the terrain surface, and thus it could be very large implying that there exists no guarantee on the qualities of the bounds. In [24], the authors proposed a *Steiner point-based* algorithm introducing additional points called *Steiner points* on the surface of the terrace for finding an ϵ-approximate geodesic shortest path between two points, where ϵ is a user-specified parameter. The algorithm computes tighter lower and upper bounds of the geodesic distance than those of [25], which do not depend on the underlying terrain. According to the experimental results in [24], the algorithm ran more than 300 seconds even for a setting with a very loose error parameter $\epsilon = 0.25$. All of these algorithms compute the approximate geodesic distances *on-the-fly*, which is not efficient enough in (real-time) applications involving many distance and shortest path queries.

Algorithm	Oracle Building Time	Oracle Size	Query Time	Error	Exact or Appro.?	Index-Base or Index-Free?
MMP [32]	-	-	Large	-	Exact Algo.	Index-Free
CH [7]	-	-	Large	-	Exact Algo.	Index-Free
ICH [46]	-	-	Large	-	Exact Algo.	Index-Free
[25]	-	-	Large	Large	Appro. Algo.	Index-Free
K-Algo [24]	-	-	Large	Small	Appro. Algo.	Index-Free
SP-Oracle [15]	Large	Large	Medium	Small	Appro. Algo.	Index-Based
SE	Small	Small	Small	Small	Appro. Algo.	Index-Based

Table 3. Pros and Cons of Shortest Geodesic Distance and Path Query Algorithms

In order to answer the geodesic shortest path/distance and shortest path queries more efficiently, some existing studies aim at designing oracles [2, 4, 15, 23]. [23] proposed a data structure for the Single-Source All-Destination (SSAD) approximate geodesic shortest path queries, where the source point of each shortest path query is already given before the data structure is built. This data structure could answer any shortest path query from this fixed source point to any destination. However, this data structure is limited to a fixed source point. Even though different data structures from all possible source points could be built, the total space occupied by all these data structures is prohibitively large, which is not feasible in practice. [2, 4] designed an oracle for approximate geodesic shortest path queries and [15] designed an oracle for approximate geodesic shortest distance and shortest path queries. These two oracles share similar ideas, and the one in [15] is better in terms of oracle size and query time mainly because geodesic distance and shortest path queries are intrinsically easier than geodesic path queries. Specifically, the oracle in [15] has its space complexity of $O(\frac{N}{\sin(\theta)\cdot\epsilon^{1.5}} \cdot \log^2(\frac{N}{\epsilon})\log^2\frac{1}{\epsilon})$ and its query time complexity of $O(\frac{1}{\sin(\theta)\cdot\epsilon^1}\log\frac{1}{\epsilon} + \log\log N)$, where θ is the minimum inner angle of any face on the terrain surface.

As will be introduced later, we use this oracle as a baseline oracle for comparison, and our experimental results show that this oracle has a scalability issue due to its large oracle size, and its corresponding query time is significantly larger than that of our oracle.

We summarize the algorithms mentioned above and compare their pros and cons in Table 3. As the table shows, the first three algorithms [7, 32, 46] are index-free (i.e., on-the-fly) exact algorithm. They all provide exact geodesic distances and paths but they suffer from their prohibitively large query time. The fourth algorithm [25] and fifth algorithm [24] are index-free approximate algorithms whose query time is accelerated due to their tradeoff of their reduced accuracy. But, their query time is still not good enough. Besides, [25] suffers from its uncontrollable error bound which is data-dependent. The sixth one [15] is an index-based approximate algorithm. It has guarantee on the error bound and provides medium query time but still suffers from its prohibitively a large indexing cost (including oracle building time and oracle size) which renders itself to scale up to sizable datasets. Our oracle enjoys a small indexing cost and also a small query time and as such, it could scale up and it also provides guarantee on the error bound.

Some other related studies include those proximity queries relying on the geodesic shortest distance queries [12, 13, 39, 46, 47]. Specifically, [12, 13, 39] studied kNN queries, among which, [12] and [13] adopted a multi-resolution terrain model and used the low-resolution terrain information constructed in the model for pruning some unnecessary regions on the terrain surface in order to answer kNN queries efficiently, and the time complexity is $O(N^2)$. In the worst case, it takes $O(N^2)$ time to answer a kNN query, which is very costly. In [39], the authors proposed a Voronoi diagram-based method for kNN queries, However, it takes $O(N\log^2 N)$ to return an answer of a kNN query in the worst case. The major idea is to construct an approximate Voronoi diagram on the terrain surface which contains 2 regions for each POI. One is called a tight region, where every point in the region will have the POI as nearest neighbor for sure. The other is called loose region, where the POI of this region is not necessarily but possibly the nearest neighbor for any point inside. But any point outside the loose region won't have the POI as its nearest neighbor. To process a kNN query, it visits all the regions in the ascending order of their distance to the query point until k POIs are visited. [46] studied dynamic monitoring of the kNN queries which is a dynamic version of SI [39] and [47] studied reverse nearest neighbor queries.

Besides, some studies [6, 17, 34–37] focused on studying well-separated pairs. [6] studied it in the Euclidean space, [17] studied its dynamic case (e.g., insertion and deletion) and [34–36] studied it on road networks. Another is to incorporate the idea in distance join queries [33]. However, they are different from ours because we studied it in the terrain context and different contexts give different challenges (e.g., in the terrain context, how to build a distance oracle involving many expensive geodesic distance computations is very challenging).

Among these existing studies, only [39] and [46] focused on finding *exact* k-nearest neighbors but [12] and [13] returned *approximate* answers of k-nearest neighbors without any error guarantee as claimed in [39, 46].

We summarize the existing k nearest neighbor algorithms and compare them with our algorithm in Table 4. It is worth mentioning that SP-Oracle is adapted to handling this query since the k nearest neighbors could be found through multiple distance queries on SP-Oracle. As could be observed from the table, SP-Oracle (Adapted) suffers from its very large indexing cost (including oracle building time and oracle size) although it provides a controllable error bound. MR [12, 13] and SI [39] have better performance compared with SP-Oracle (Adapt) but their indexing cost and query time are still not good enough. Besides, MR [12, 13] suffers from its uncontrollable error which is data-dependent.

Algo.	Oracle Building Time	Oracle Size	Query Time	Appro. Error
SP-Oracle (Adapted) [15]	Large	Large	Medium	Small
MR[12, 13]	Medium	Medium	Medium	Large
SI [39]	Large	Medium	Medium	Small
SE	Small	Small	Small	Small

Table 4. Pros and Cons of k Nearest Neighbor Algorithms

However, to the best of our knowledge, there is no existing work about the bichromatic version of k-nearest neighbor queries in the context of terrain datasets.

There are other proximity queries related to our study. One example is monochromatic/bichromatic k-farthest neighbor queries and the other example is monochromatic/bichromatic closest/farthest pair queries. Similarly, to the best of our knowledge, there is no existing work about these queries.

5.2 Baseline Methods

In this section, we first present the baselines for the distance and path queries (Section 5.2.1), then give the baselines for other proximity queries studied in this paper (Section 5.2.2) and finally compare each baseline with *SE* in theory (Section 5.2.3).

5.2.1 Baselines for Distance and Path Queries. In this part, we first introduce two baseline oracles, namely the *Steiner point-based oracle* (in short, *SP-Oracle*) and the naive implementation of *SE* (in short, *SE(Naive)*).

Algo.	Oracle Building Time	Oracle Size	Query Time for kNN/kFN Query	Query Time for top-k CP/FP Query	(Appro.) Error
SP-Oracle [15]	$O(\frac{N}{\epsilon^2}\log^3(\frac{N}{\epsilon})\log^2\frac{1}{\epsilon})$	$O(\frac{N}{\epsilon^{1.5}}\log^2(\frac{N}{\epsilon})\log^2\frac{1}{\epsilon})$	$O(\frac{n}{\epsilon}\log\frac{1}{\epsilon} + n\log\log N)$	$O(\frac{n^2}{\epsilon}\log\frac{1}{\epsilon} + n^2\log\log N)$	$\epsilon' = \frac{2\epsilon}{1-\epsilon}$
SI [39]	$O(N^2\log N)$	$O(N)$	$O(N\log^2 N)$	Not Applicable	1
SE	$O(\frac{N\log^2 N}{\epsilon^{2\beta}} + nh\log n + \frac{nh}{\epsilon^{2\beta}})$	$O(\frac{nh}{\epsilon^{2\beta}})$	$O(nh)$	$O(\frac{n}{\epsilon^{2\beta}})$	$\epsilon' = \frac{2\epsilon}{1-\epsilon}$

Table 5. Comparison of Algorithms for kNN/FN and top-k BCP/FCP query with Error Parameter ϵ (where β is a positive real number between 1.3 and 1.5, and h is a positive integer smaller than 30 in practice)

Steiner Point-Based Oracle: The first baseline oracle is called the *Steiner point-based oracle* (in short, *SP-Oracle*) proposed in [15] which were originally proposed for vertex-to-vertex distance queries and could also be adapted for both POI-to-POI (P2P) distance queries and arbitrary point-to-arbitrary point (A2A) distance queries. Next, we describe how this adapted distance oracle [15] could handle A2A distance queries only (since A2A distance queries could be regarded as a general setting compared with P2P distance queries). Its major idea is as follows. It first introduces $O(\frac{1}{\sin(\theta)\cdot\sqrt{\epsilon}}\log\frac{1}{\epsilon})$ additional points called *Steiner points* on each face of the terrain surface and $O(\frac{N}{\sin(\theta)\cdot\epsilon}\log\frac{1}{\epsilon})$ Steiner edges connecting Steiner points on the same face, where θ is the minimum inner angle of any face on the terrain surface. It then constructs a graph, denoted by G_ϵ, where the set of vertices in the graph is the set containing all the Steiner points and all existing vertices and the set of edges in the graph is the set of all existing edges and all the additional edges added each with its weight equal to its corresponding *Euclidean distance*. *SP-Oracle* indexes the exact distances between any two Steiner points on G_ϵ. Consider an A2A distance query. Given two arbitrary points, namely s and t, on the surface of the terrain, *SP-Oracle* finds (1) a set X_s of Steiner points on the face containing s and its adjacent faces, and (2) another set X_t of Steiner points on the face containing t and its adjacent faces. Then, for each point p_s in X_s and each point p_t in X_t, it computes a distance equal to the sum of the Euclidean distance between s and p_s, the exact distance between p_s and p_t on G_ϵ and the Euclidean distance between p_t and t. Finally, it returns the smallest distance computed as the estimated geodesic distance between s and t. We present the oracle building time, oracle size, query time, and distance error bound of *SP-Oracle* in Table 1.

SE(Naive): The second baseline is called the naive method of *SE* (in short, *SE*(*Naive*)) which is exactly our *SE* with the naive method for the both the oracle construction and the query processing. We present the oracle building time, oracle size, query time, and distance error bound of *SE*(*Naive*) in Table 1.

We proceed to present the on-the-fly version of the existing algorithms for the distance and path queries.

On-the-fly Algorithm The *Kaul's algorithm* (in short, *K-Algo*) recently proposed in [24] could be used as the baseline algorithm which computes the approximate geodesic distance *on-the-fly* (since it is the best-known algorithm in the literature). Although *K-Algo* is a non-distance oracle algorithm, it is interesting to compare it with our *SE*. The time complexity of *K-Algo* is $O(\frac{l_{max}^3 N}{(l_{min}\cdot\epsilon\cdot\sqrt{1-\cos\theta})^3} + \frac{l_{max}\cdot N}{\epsilon\cdot l_{min}\cdot\sqrt{1-\cos\theta}}\log(\frac{l_{max}\cdot N}{\epsilon\cdot l_{min}\cdot\sqrt{1-\cos\theta}}))$[1] where l_{min} (resp., l_{max}) is the length of the shortest (resp., longest) edge and θ is the minimum inner angle of any face.

5.2.2 Baselines for Other Proximity Queries. For the k nearest neighbor query, we compare our method with the state-of-the-art algorithm in terms of both theory and empirical performance, namely Surface Index (SI) [39].

To the best of our knowledge, our method is the first algorithm for the top-k closest pair or farthest pair query processing. The only baseline that we consider is the SP-Oracle. Although it was designed for the distance and path query only, but it could be adapted as follows. SP-Oracle could process the query by materializing all pairwise distances and return the corresponding list of point pairs for each proximity query.

5.2.3 Comparison. We compare the oracle proposed in this paper, i.e., *SE*, and the three baselines, i.e., *SP-Oracle*, *SE*(*Naive*) and *K-Algo*, in terms of error bound, oracle building time, oracle size and query time, for the distance and path queries and the results are shown in Table 1. We highlight some of the comparison results as follows. Consider the error bound. Our *SE* and all baseline methods, namely *SP-Oracle*, *SE*(*Naive*) and *K-Algo*, have the same error bound equal to ϵ. Consider the oracle building time. As described before, we know that *SE* has a lower oracle building (or oracle construction) time complexity than *SE*(*Naive*). Besides, in our experimental results, the empirical oracle building time of *SE* is smaller than that of *SP-Oracle*. Consider the oracle size. The oracle size of *SE* is the same as that of *SE*(*Naive*). Besides, in our experimental results, the empirical oracle size of *SE* is smaller than that of *SP-Oracle*. Consider the query time. Since h is very small (at most 30 in our experimental results), *SE* has the lowest query time complexity compared with *SE*(*Naive*) and *SP-Oracle*. *K-algo* has the largest query time which is significantly larger than others.

For the k nearest neighbor and farthest neighbor query, we consider SI and SP-Oracle as our baselines and for the top-k closest pair and farthest pair query, we consider SP-Oracle as our baseline. The theoretical comparison is listed in Table 5. We highlight some results of the comparison as follows. Consider the approximate ratio. Our *SE* and *SP-Oracle* have the same appro. error equal to $\epsilon' = \frac{2\epsilon}{1-\epsilon}$. The error bound of *SI* is 0 since it is an exact algorithm. But ϵ' is small enough for many applications and as we will show in the experiment, the empirical error bound is quite small (smaller than 0.4 which means the appro. ratio is 1.4 only) even when ϵ is set to be as large as 0.25. Consider the oracle building time. As described before, we know that *SE* has a lower oracle building (or oracle construction) time complexity than *SI* since the building time of SI contains a N^2 term which is significantly large. Besides, in our experimental results, the empirical oracle building time of *SE* is smaller than that of *SP-Oracle*. Consider the oracle size. The oracle size of *SE* is the same as that of *SI* when n is much smaller than N which is typical for terrain data. Besides, in our experimental results, the empirical oracle size of *SE* is smaller than that of *SP-Oracle*. Consider the query time. Since h is very small (at most 30 in our experimental results), *SE* has the lowest query time complexity compared with *SI* and *SP-Oracle*.

[1]By Section 4.2 of [24], its running time is $O((N+N')(\log(N+N') + (\frac{l_{max}\cdot K}{l_{min}\sqrt{1-\cos\theta}})^2)$ where $N' = O(\frac{l_{max}\cdot K}{l_{min}\sqrt{1-\cos\theta}}N)$ and K is a parameter which is a positive number at least 1. By Theorem 1 of [24], we obtain that its error bound ϵ is equal to $\frac{1}{K-1}$. Thus, we obtain this time complexity.

6 EMPIRICAL STUDIES

6.1 Experimental Setup

We conducted our experiments on a Linux machine with 2.67 GHz CPU and 48GB memory. All algorithms were implemented in C++.

Datasets. Following some existing studies on terrain data [12, 30, 39], we used three real terrain datasets, namely BearHead (in short, BH), EaglePeak (in short, EP) and San Francisco South (in short, SF) and these datasets can be downloaded from http://data.geocomm.com/. For each of these terrain datasets, we extracted a set of POIs from the corresponding region in OpenStreetMap. Table 2 shows the dataset statistics. Besides, a smaller version of SF dataset which corresponds to a small sub-region of the SF dataset and contains 1k vertices and 60 POIs was also used since one of the baselines, *SE-Naive*, is not feasible on any of the full datasets due to its expensive cost of building an oracle.

Algorithms. We tested our distance and path oracle *SE* in the experiment. We set M to be 5 in the default setting. We also tested a degenerated version where $M = 0$ (i.e., in this case, *SE* is reduced to a distance oracle only), denoted by $SE_{M=0}$. For distance and path queries, we compared our oracle with the algorithms mentioned in Section 5.2. For other proximity queries, we compared our oracle with the algorithms mentioned in Table 5.

The query time reported corresponds to the average running time of 100 queries. We conducted experiments with the following queries: (1) the shortest distance query, (2) the shortest path query, (3) the monochromatic and bichromatic k nearest neighbor query, (4) the monochromatic and bichromatic k farthest neighbor query, (5) the top-k monochromatic and bichromatic closet pair query, and (6) the top-k monochromatic and bichromatic farthest pair query. For the shortest distance and path query, we evaluated our new oracle *SE* and three baselines, *SP-Oracle* [15], *K-Algo* [24] and *SE-Naive*, are studied in the experiments. For the monochromatic and bichromatic k nearest neighbor and farthest neighbor queries, we consider the baselines SP-Oracle [15] and the best-known algorithm for kNN queries, SI (Surface Index) [39] as shown in Table 5. As studied in [39], SI is the the best-known algorithm for kNN queries on terrain surfaces in terms of building time and query time. For the bichromatic top-k closest pair and farthest pair queries, we consider the baseline SP-Oracle [15] as shown in Table 5. Note that for the k nearest neighbor and farthest neighbor queries, the experimental results of [39] showed that [39] is more efficient than [11, 13] in practice. Thus, we do not consider [11, 13] as our baseline. For *SE*, we study two variations: one is $SE(Greedy)$ which is based on the greedy point selection strategy and the other is $SE(Random)$ which is based on the random point selection strategy.

Query Generation. Each P2P (V2V) shortest distance or path query was generated by randomly sampling two POIs (vertices) on the surface of a terrain, one as a source and the other as a destination. Each A2A shortest distance and path query was generated by randomly selecting two arbitrary points, one as a source and the other as a destination. To randomly select an arbitrary point, we first generated a 2D coordinate (x, y) which is a point randomly selected in the 2D rectangular region covered by the terrain and then computed the point on the terrain surface whose projection on the x-y plane is (x, y).

For each k nearest or farthest neighbor query, we randomly selected a POI as a query point. For each bichromatic k nearest or farthest neighbor query, besides the query point, we also randomly selected a set of POIs containing 1k POIs for the k nearest or farthest neighbor candidates. For each top-k bichormatic closest or farthest pair query, we randomly selected two disjoint sets of POIs each containing 1k POIs for the k closest or farthest pair

Dataset	max	min	avg.	std.
BH	16.57	0.82	7.8	3.33
EP	14.15	0.33	6.25	3.15
SF	16.92	0.48	7.09	3.6

Table 6. Statistics of Query Distances (km)

candidates. Note that the running time of each query reported in the paper is the average of 100 queries generated with the corresponding method mentioned above.

Table 6 shows the statistics of the query distances of all queries performed on each dataset as shown in Table 2.

Factors & Measurements. Five factors, namely ϵ (the error parameter), n (the number of POIs), N (the number of vertices in a terrain), M (the number of intermediate points associated with each node pair in SE), and k (the size of the output to be returned) in each Top-k query were studied. Four measurements, namely (1) *oracle building time* (which is the time for constructing the distance oracle), (2) *oracle size* (which is the space consumption of the distance oracle), (3) *query time* (which is the time for answering a distance or path query based on the oracle), and (4) *error* (which is the error of the distance returned based on the oracle) were used for evaluating the oracles. For the query time, 100 queries were answered and the average running time was returned.

6.2 Experimental Results

In this section, we present the results of P2P distance and path queries in Section 6.2.1, other experiments (e.g., V2V distance and path queries and A2A distance and path queries) in Section 6.2.2, the experiments on the monochromatic and bichromatic k nearest neighbor and farthest neighbor queries in Section 6.2.3, the experiments on the monochromatic and bichromatic closest-pair and farthest-pair queries in Section 6.2.4, a case study of our proximity query processing algorithms in Section 6.2.5 and a summary of the results in Section 6.2.6.

6.2.1 P2P Queries.

6.2.1 P2P Queries. In this section, we present the experimental results on P2P queries.

Effect of ϵ. We tested 5 different values of ϵ from $\{0.05, 0.1, 0.15, 0.2, 0.25\}$. Figure 10(a)-(d) show the results on the smaller version of the SF dataset. According to the results, (1) the building times of $SE(Greedy)$ and $SE(Random)$ are almost the same and are both smaller than those of *SP-Oracle* and *SE-Naive*, e.g., when $\epsilon = 0.05$, $SE(Greedy)$ and $SE(Random)$ have their building times 1 order (resp., at least 2 orders) smaller than that of *SP-Oracle* (resp., *SE-Naive*), (2) the sizes of $SE(Greedy)$, $SE(Random)$ and *SE-Navie* are 2-3 orders of magnitude smaller than that of *SP-Oracle*, (3) the query time of $SE(Greedy)$ is the smallest and about half of that of $SE(Random)$, and the query times of both $SE(Greedy)$ and $SE(Random)$ are orders of magnitude smaller than those of others, and (4) the errors of all oracles are very small and much smaller than the theoretical bound (which is ϵ).

Based on the results shown above, we adopt the following for the simplicity of presentation: (1) the results of error for the rest of experiments are omitted since the errors of all oracles are similar and very small (smaller than $\epsilon/10$) compared with the error bound, (2) the results of *SE-Naive* on any full datasets are not shown simply because it cannot be built within a reasonable amount of time, e.g., within a month, and (3) the results of $SE(Greedy)$ are omitted for the rest of experiments since $SE(Random)$ and $SE(Greedy)$ have similar performance and we omit $SE(Greedy)$ for the clarity and by SE, it means $SE(Random)$ for the remaining presentation.

The results on the other three datasets, namely BH, EP and SF, are shown in , Figure 11, Figure 12 and Figure 13 respectively. Note that in Figure 11 and Figure 12, the results of *SP-Oracle* for all settings of ϵ are not shown since the size of *SP-Oracle* exceeds our memory budget (i.e., 48GB).

Effect of n. We tested 5 different values of n from $\{60k, 90k, 120k, 150k, 180k\}$ and used the SF dataset for this experiment. As mentioned in Section 6.1, we have $51k$ POIs in the SF South dataset (170k vertices), and in order to obtain a set of the targeted number of POIs, we do as follows. Let n denote the targeted number of POIs we want to generate. Let P be the set of POIs that we have and n' be the number of POIs in P. We generate $(n - n')$ 2-dimensional points (x, y) based on a Normal distribution $N(\mu, \sigma^2)$, where $\mu = (\overline{x} = \frac{\sum_{p' \in P} x_{p'}}{n'}, \overline{y} = \frac{\sum_{p' \in P} y_{p'}}{n'})$ and $\sigma^2 = (\frac{1}{n}\sum_{p' \in P}(x_{p'} - \overline{x})^2, \frac{1}{n}\sum_{p' \in P}(y_{p'} - \overline{y})^2)$. If a generated point (x, y) is outside the range of the terrain, we simply discard it and re-do the process until a point within the range is generated. At the end, we project each generated point (x, y) to the surface of the terrain and take the projected point as a newly generated POI. The results are shown in Figure 14. According to the these results, $SE_{M=0}$ outperforms *SP-Oracle* in terms of oracle building time, oracle size and query time and significantly outperforms *K-Algo* in terms of query time. SE outperforms *SP-Oracle* in terms of building time, size and distance query time by 1-2 orders of time but is slower than *SP-Oracle* in terms of path query time by several times. But SE still has a better overall performance than *SP-Oracle*.

Effect of N. We tested 5 values of N from $\{0.5M, 1M, 1.5M, 2M, 2.5M\}$ on synthetic datasets. Each synthetic dataset with N vertices is a terrain surface from an enlarged BH dataset (4.2M vertices) simplified by a surface simplification algorithm [30]. Note that each simplified terrain surface covers the same region as the original BH dataset with a different simplification ratio and still has 4k POIs. The enlarged BH dataset was generated from the BH dataset as follows. On each face of BH, we added a new vertex on its geometric center and add a new edge between the new vertex and each of the three vertices on the face. The results are shown in Figure 15, where the results of *SP-Oracle* are not shown since the size of *SP-Oracle* exceeds our memory budget (i.e., 48GB).

Effect of M. We studied the effect of the number M of intermediate points associated with each node pair in SE. We tested 5 values of M from $\{0, 5, 10, 15, 20\}$ on the BH dataset. The results are shown in Figure 16, where the results of *SP-Oracle* are not shown since the size of *SP-Oracle* exceeds our memory budget (i.e., 48GB). From this figure, we could have the following observations: (1) the oracle building time of SE is almost intact in the setting of different M. This is because the running time of adding the intermediate points for each node pair is neglectable compared with the time of finding the shortest distance between the centers of the two node pairs. As such, adding the intermediate points just introduces negligible overhead for the building time. (2) But, the oracle size of SE increases linearly with the growth of M since each node pair in SE stores M intermediate points and the additional storage is required for larger M. (3) The distance query time is independent of M since the distance query algorithm does not get the intermediate points involved. (4) The path query time is monotonically decreasing with the growth of M but it decreases very little when M is larger than or equal to 5.

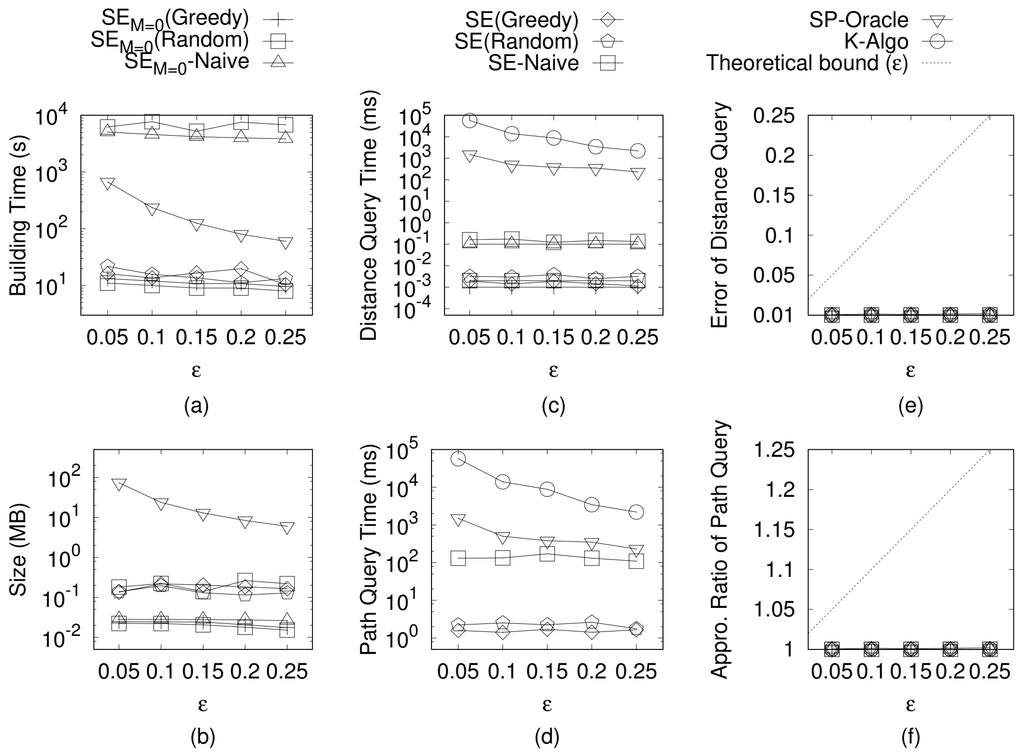

Fig. 10. Effect of ϵ on SF dataset (Smaller Version) (P2P Distance and Path Queries)

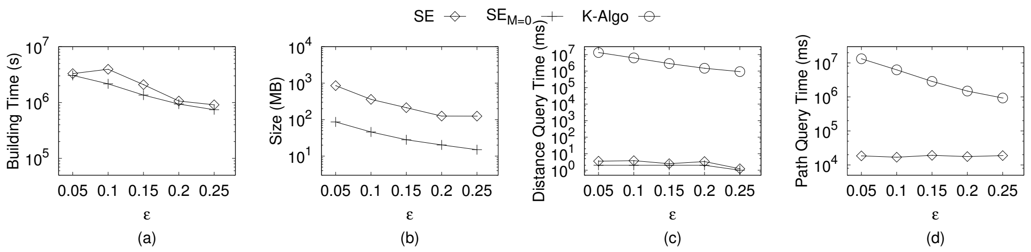

Fig. 11. Effect of ϵ on BH dataset (P2P Distance and Path Queries)

6.2.2 *Other Experiments on Distance and Path Queries.* In this section, we present the other experiments on geodesic shortest distance and path queries including the results of V2V queries, A2A queries and P2P queries in the case where $n > N$.

V2V Queries: In V2V queries, the original POIs are discarded, and we treat all vertices as POIs. We varied n and ϵ for the experiments. Consider the experiment studying the effect of n. Note that $N = n$ in this experiment. We tested 5 values of n (i.e., N) from $\{60k, 90k, 120k, 150k, 180k\}$ on synthetic datasets, and each synthetic dataset with N vertices corresponds to a sub-region of a SF dataset with a higher resolution (10m×10m resolution, 1M vertices). The results are shown in Figure 17, and according to the results,

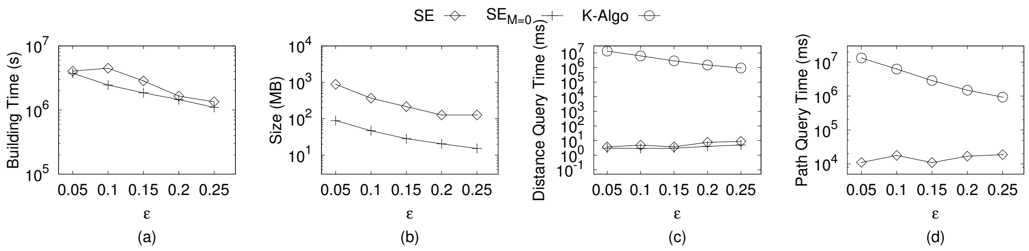

Fig. 12. Effect of ϵ on EaglePeak dataset (P2P Distance and Path Queries)

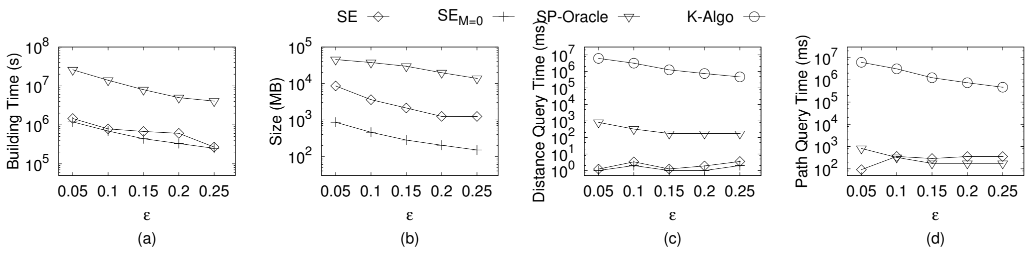

Fig. 13. Effect of ϵ on San Francisco South (P2P Distance and Path Queries)

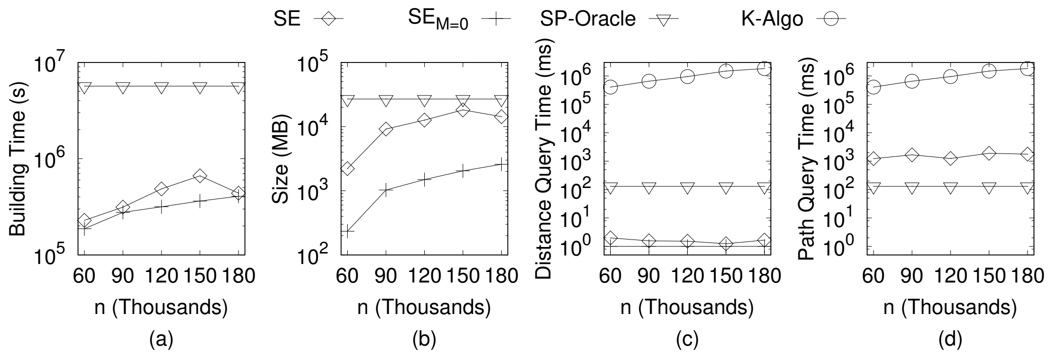

Fig. 14. Effect of n on SF dataset (P2P Distance and Path Queries)

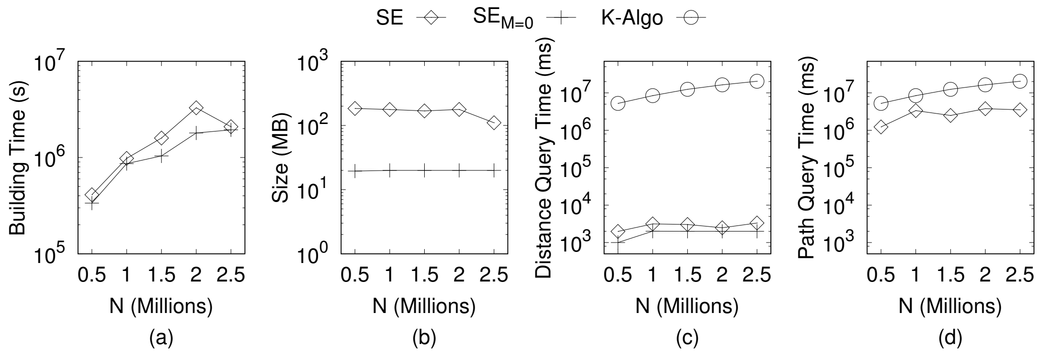

Fig. 15. Effect of N on BH dataset (P2P Distance and Path Queries)

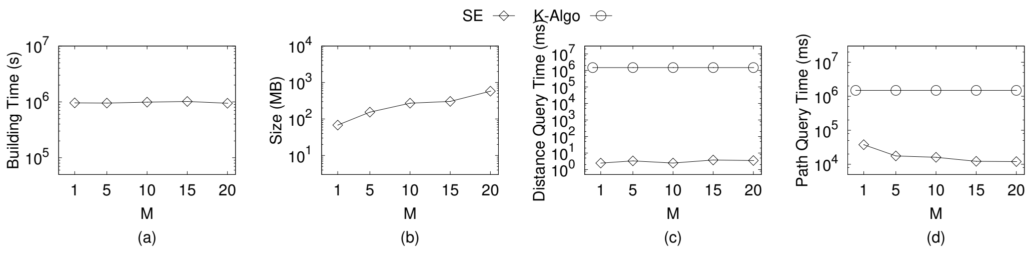

Fig. 16. Effect of M on BH dataset (P2P Distance and Path Queries)

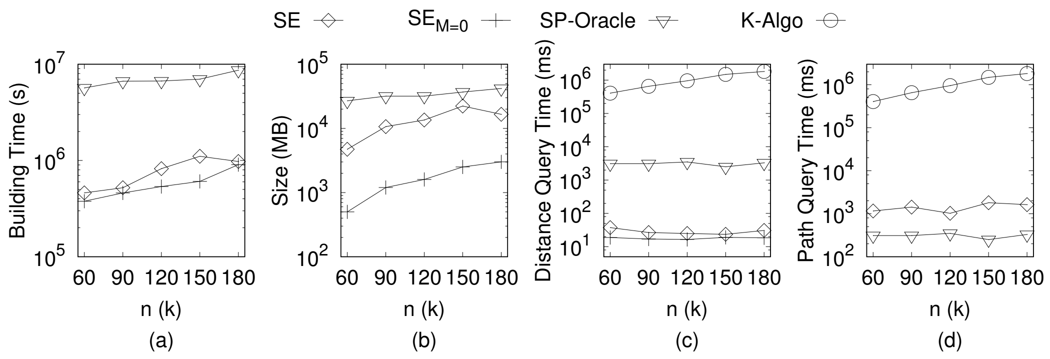

Fig. 17. Effect of n on SF dataset (V2V Distance and Path Queries)

SE has its building time and size both at least 1 order of magnitude smaller than *SP-Oracle* and its distance query time 2-3 (resp., 5-6) orders of magnitude smaller than that of *SP-Oracle* (resp., *K-Algo*). The path query time of *SE* and *SP-Oracle* is smaller than that of *K-Algo* by 3-4 orders of magnitudes. Although the path query time of *SE* is larger than that *SP-Oracle* by several times, *SE* outperforms *SP-Oracle* much more (i.e., more than 1 orders of magnitudes) in terms of building time and distance query time and also outperforms *SP-Oracle* in terms of oracle size. Thus, *SE* has the best overall performance.

We also conducted the experiment studying the effect of ϵ with values in $\{0.05, 0.1, 0.15, 0.2, 0.25\}$ on the smaller version of the SF dataset. The results are also similar. In particular, the query time of *SE* is 5-6 orders (resp., 6-8 orders) of magnitude smaller than that of *SP-Oracle* (resp., *K-Algo*).

Arbitrary Point to Arbitrary Point (A2A) Queries. We tested the A2A distance queries where the query point is not a POI but an arbitrary point on the terrain surface. We used the low resolution BH (resolution: 30 meter, 150k vertices) dataset by varying ϵ from $\{0.05, 0.1, 0.15, 0.2, 0.25\}$. Figure 18(a), (b), (c) and (d) shows the building time, oracle size, distance query time and path query time, respectively. According to the results, $SE_{M=0}$ outperforms *SP-Oracle* by several times in terms of building time and oracle size. *SE* has similar building time with $SE_{M=0}$ and *SP-Oracle* and SE has several times larger size than *SP-Oracle*. The distance query time of $SE_{M=0}$ and *SE* is 2-3 (resp., 5-6) orders of magnitude smaller than that of *SP-Oracle* (resp., *K-Algo*). The path query time of *SE* and *SP-Oracle* is very close which is 3-4 orders of magnitudes smaller than that of *K-Algo* and the path query time of *SE* is slightly smaller than that of *SP-Oracle*. Thus, we observe that *SE* has better

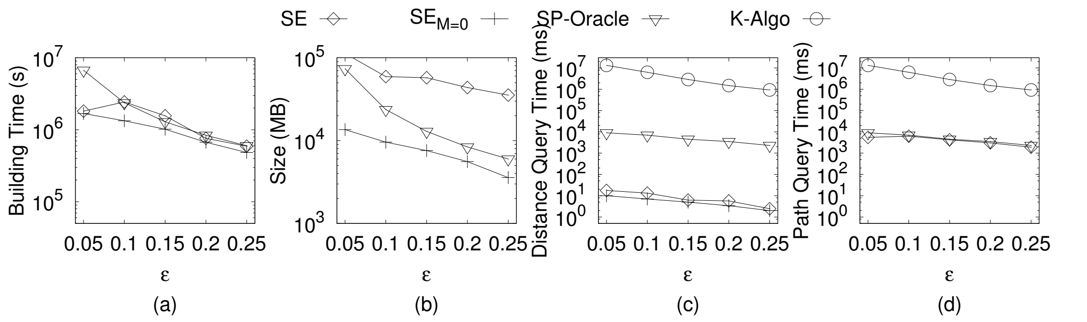

Fig. 18. A2A Queries

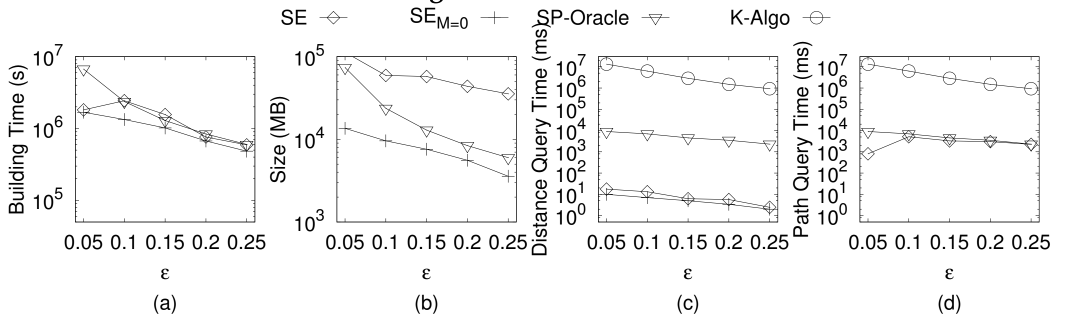

Fig. 19. P2P Queries In The Case $n > N$

overall performance than *SP-Oracle* since (1) the advantage of *SE* in the distance query time (which is 2-3 orders of magnitudes) outweighs its disadvantage in oracle size (which is several times only) compared with *SP-Oracle*, and (2) *SE* and *SP-Oracle* have very close performances in terms of building time and path query time.

P2P Queries In The Case $n > N$. We tested P2P queries of the case $n > N$ on the low resolution BH (resolution: 30 meter, 150k vertices) dataset by varying ϵ from $\{0.05, 0.1, 0.15, 0.2, 0.25\}$. We generated 1M POIs by the same method as mentioned in Section 6.2.1. Figure 19(a)(b)(c)(d) shows the building time, oracle size, distance query time and path query time, respectively. The result is similar to that of A2A query. Note that the building time and space of P2P Queries in the case $n > N$ is the same as those of A2A queries since each tested oracle is the same in the two queries.

6.2.3 Monochromatic and Bichromatic k Nearest Neighbor and Farthest Neighbor Query.

In this section, we first report the experimental results of the monochromatic and bichromatic k nearest neighbor queries.

Effect of ϵ. We tested 5 different values of ϵ (i.e., $0.05, 0.1, 0.15, 0.2, 0.25$). The results on the two types of queries in the datasets BearHead, EaglePeak and San Francisco South is shown in Figure 20 - Figure 25. As the figures show, the kNN query time of *SE* is smaller than that of SP-Oracle and the best-known algorithm, SI by several orders of magnitude in BH and EP. Although SE does not consistently outperforms the baselines in San Fransisco dataset, SE is still the fastest one in most cases in Figure 22 and Figure 25. When ϵ is smaller than 0.2, the appro. errors of *SE*, the best-known algorithm, SI and SP-Oracle are very close and are much smaller than the theoretical bound. When ϵ is at least 0.2, although the appro. error of *SE* is higher than that of the best-known algorithm, SI and SP-Oracle by a notable

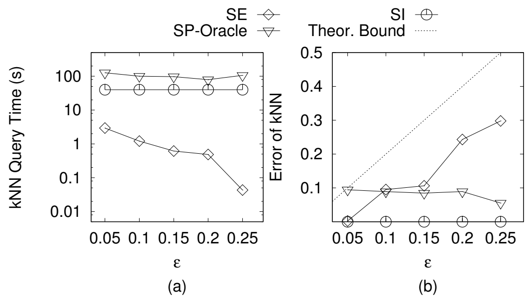

Fig. 20. Effect of ϵ on real dataset, BearHead, for monochromatic kNN

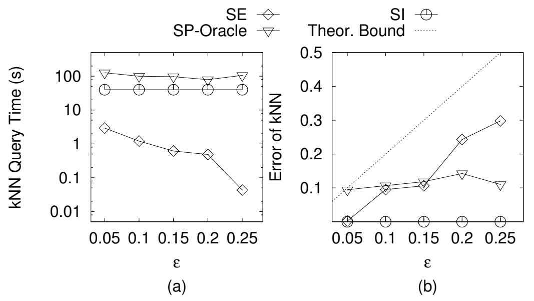

Fig. 21. Effect of ϵ on real dataset, EaglePeak, for monochromatic kNN

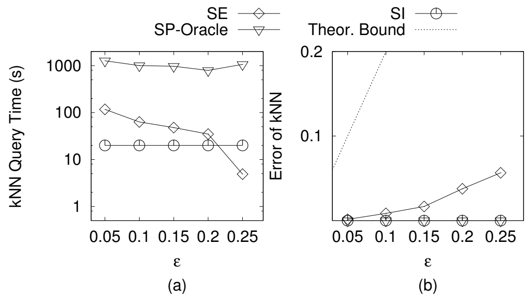

Fig. 22. Effect of ϵ on real dataset, San Francisco, for monochromatic kNN

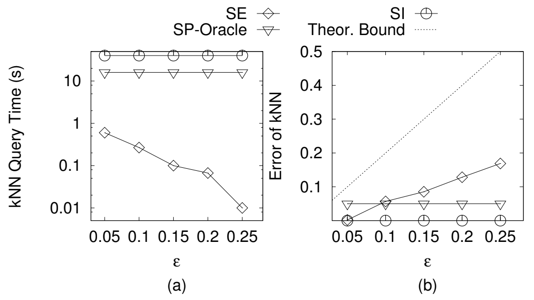

Fig. 23. Effect of ϵ on real dataset, BearHead, for bichromatic kNN

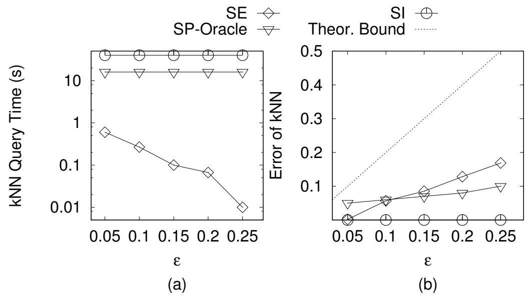

Fig. 24. Effect of ϵ on real dataset, EaglePeak, for bichromatic kNN

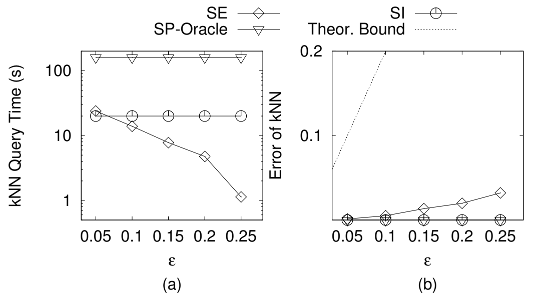

Fig. 25. Effect of ϵ on real dataset, San Francisco, for bichromatic kNN

margin, it is still obviously smaller than the theoretical bound (i.e., *SE* still satisfies accuracy requirement).

Effect of k. We tested 4 different values of k, namely $5, 10, 15, 20$. The results on the two types of queries in the datasets, BearHead, EaglePeak and San Francisco South are shown in Figure 26 - Figure 31. As the figures show, the kNN query time of *SE* is smaller than

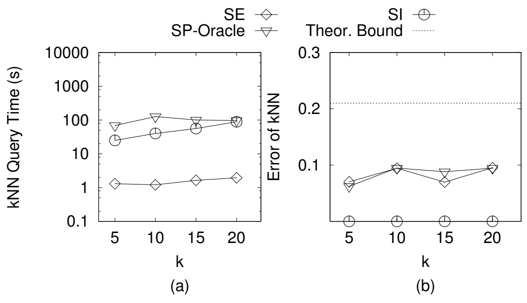

Fig. 26. Effect of k on real dataset, BearHead, for monochromatic kNN

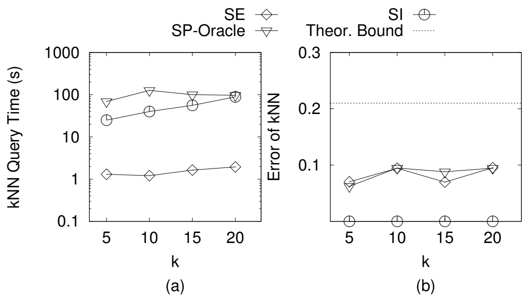

Fig. 27. Effect of k on real dataset, EaglePeak, for monochromatic kNN

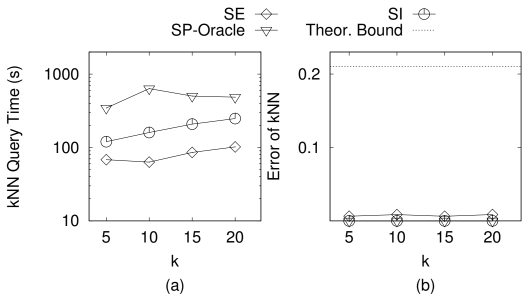

Fig. 28. Effect of k on real dataset, San Francisco South, for monochromatic kNN

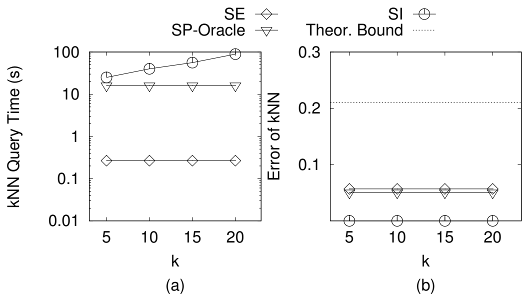

Fig. 29. Effect of k on real dataset, BearHead, for bichromatic kNN

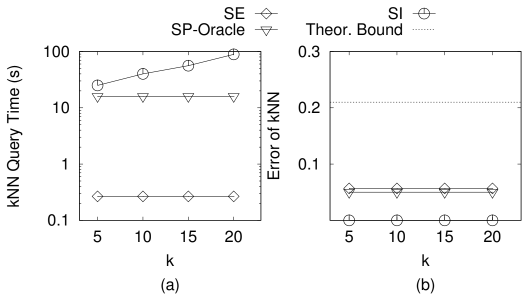

Fig. 30. Effect of k on real dataset, EaglePeak, for bichromatic kNN

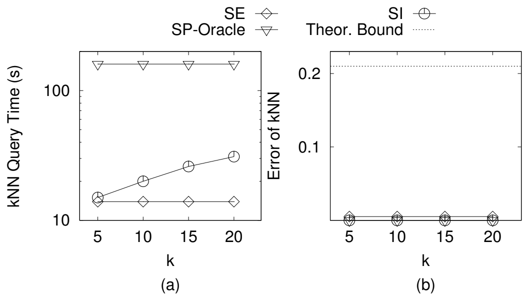

Fig. 31. Effect of k on real dataset, San Francisco South, for bichromatic kNN

that of SP-Oracle and the best-known algorithm, SI by several orders of magnitude in BH and EP. Although the speedup of SE compared with the baselines is not that significant in San Fransisco dataset, SE is still the fastest one. The appro. error of *SE*, the best-known algorithm, SI and SP-Oracle are very small in practice and much smaller than the theoretical bound.

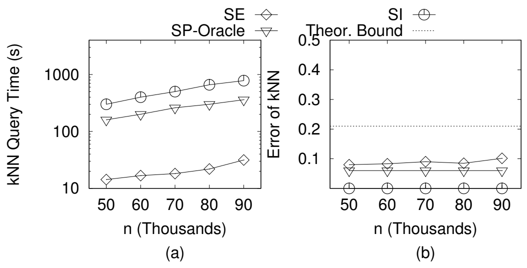

Fig. 32. Effect of n on dataset, BearHead, for bichromatic kNN

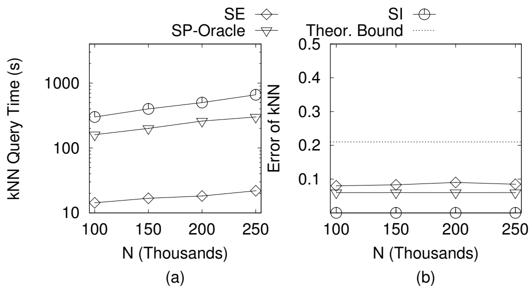

Fig. 33. Effect of N on dataset, BearHead, for bichromatic kNN

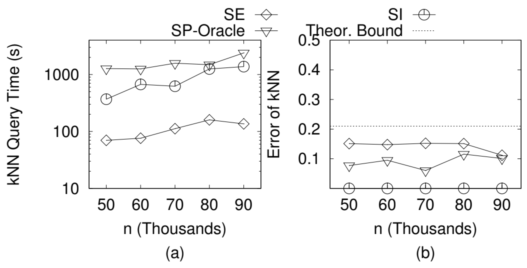

Fig. 34. Effect of n on dataset, BearHead, for monochromatic kNN

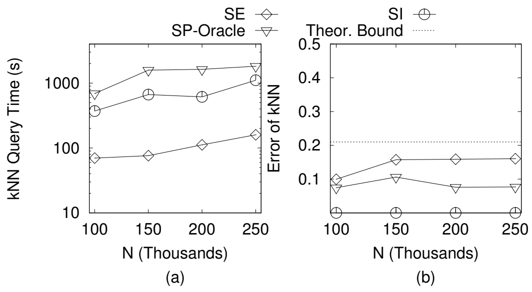

Fig. 35. Effect of N on dataset, BearHead, for monochromatic kNN

Effect of n. We studied the effect of the number of POIs n on a synthetic dataset. The values of n we tested are $50k, 60k, 70k, 80k, 90k$. We generated the synthetic POIs in the same way as stated in Section 6.2.1. The results are shown in Figure 32 and Figure 34. As figures shows, the kNN query time of *SE* is smaller than that of SP-Oracle and the best-known algorithm, SI by several times. The appro. error of *SE*, the best-known algorithm, SI and SP-Oracle are very small in practice and much smaller than the theoretical bound.

Effect of N. We studied the effect of the number of vertices N on the terrain surface on a synthetic dataset. The values of N we tested are $100k, 150k, 200k, 250k$. We generated the terrain data in the same way as stated in Section 6.2.1. The results are shown in Figure 33 and Figure 35. As figures shows, the kNN query time of *SE* is smaller than that of SP-Oracle and the best-known algorithm, SI by 0.5-3 orders of magnitude. The appro. error of *SE* is larger than that of SP-Oracle.

For the sake of space and the fact that the technique for farthest neighbor query is similar to that of the nearest neighbor query, we put the results of the farthest neighbor query into the appendix.

6.2.4 Top-k Bichromatic and Monochromatic Closest-Pair and Farthest-Pair Query. We first present the experimental results of the top-k bichromatic closest-pair (BCP) and monochromatic closest-pair (MCP) queries.

Effect of ϵ. We tested 5 different values of ϵ (i.e., $0.05, 0.1, 0.15, 0.2, 0.25$). The results of top-$k$ BCP query on BearHead, EaglePeak and San Fransisco are shown in Figure 37(a), Figure 38(a) and Figure 39(a). The results of top-k MCP query on BearHead, EaglePeak and San Fransisco are shown in Figure 41(a), Figure 42(a) and Figure 43(a). The query time of *SE* for both types of queries is very small in practice. The top-k BCP query and the top-k MCP query processing of SP-Oracle could not be finished within a reasonable time and thus, it is not shown in the figures. Since there is no exact algorithm for both types of queries, we are not able to calculate the appro. error.

Effect of k. We tested 4 different values of k, namely $5, 10, 15, 20$. The results of top-k BCP query on BearHead, EaglePeak and San Fransisco are shown in Figure 37(b), Figure 38(b) and Figure 39(b). The results of top-k MCP query on BearHead, EaglePeak and San Fransisco are shown in Figure 41(b), Figure 42(b) and Figure 43(b). The query time of *SE* for both types of queries is small in practice. The top-k BCP query and top-k MCP query processing of SP-Oracle could not be finished within a reasonable time and thus, it is not shown in the figure.

Effect of n. We studied the effect of the number of POIs n on a synthetic dataset. The values of n we tested are $50k, 60k, 70k, 80k, 90k$. The data is generated in the same way as stated in Section 6.2.1.

The results of top-k BCP query and top-k MCP query are shown in Figure 40(a) and Figure 44(a), respectively. As Figure 40(a) shows, the query time of *SE* for the top-k BCP query is small in practice. The top-k BCP query processing of SP-Oracle could not be finished within a reasonable time and thus it is not shown in the figure. We observe similar results for the top-k MCP query.

Effect of N. We studied the effect of the number of vertices N on the terrain surface on a synthetic dataset. The values of N we tested are $100k, 150k, 200k, 250k$. We generated the terrain data as stated in Section 6.2.1. The results of top-k BCP query and top-k MCP query are shown in Figure 40(b) and Figure 44(b), respectively. As Figure 40(b) shows, the query time of *SE* for the top-k BCP query is small in practice. The top-k BCP query processing of SP-Oracle could not be finished within a reasonable time and thus, it is not shown in the figure. We observe similar results for top-k MCP query.

For the sake of space and the fact that the technique for farthest-pair query is similar to that of closest-pair query, we put the results of the farthest-pair query into the appendix.

6.2.5 Case Study of Proximity Queries. In this section, we conduct a case study in one application scenario of *Path Advisor* [48] which is a widely used web-based and mobile app in a university[2]. This app provides location-based services on the campus of this university including the navigation and POI recommendation, etc. based on a 3D terrain surface in this university containing 1,789 vertices. We studied two types of commonly invoked queries. One is k nearest neighbor query and the other is shortest path query as follows.

Case Study of kNN query. In this case study, we instantiate this k nearest neighbor query as the query of 3 nearest canteens of the atrium, which is a landmark of the university and also the lobby of its main building. This query is a commonly invoked one for the newcomers of this university. Figure 36 demonstrates the user interface of the Path Advisor and the details of this query mentioned above. In this figure, there is a menu on the left side which provides the search option and also legends. On the right side, it shows the 3D terrain model of the campus. The query point (i.e., the location of the atrium) is marked with an icon

[2]Due to the double-blind policy, we do not write down the name of this university.

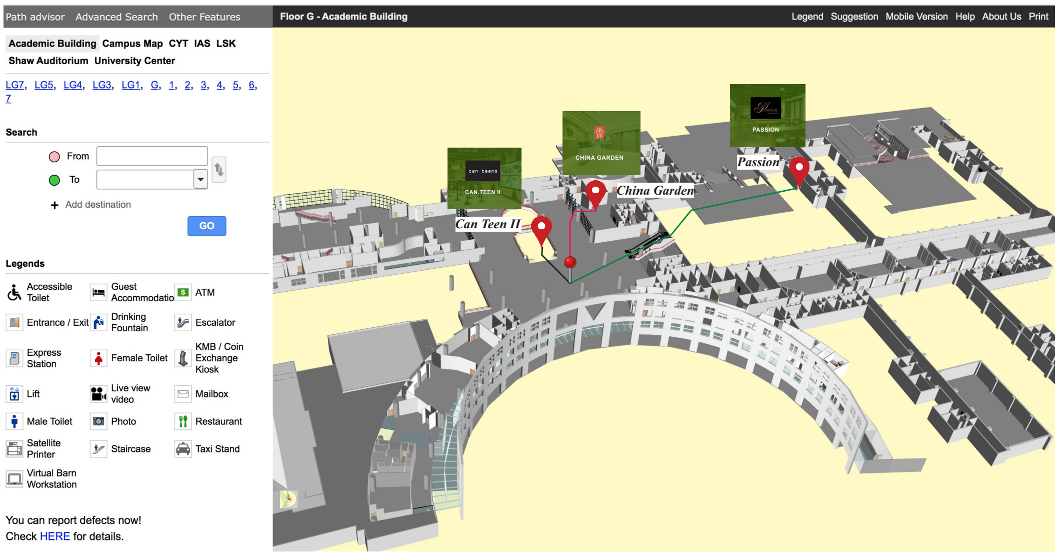

Fig. 36. An Illustration of Path Advisor in A University

of a round push pin and the 3 nearest canteens are shown by using the icon of location pinpoints. The three restaurants are *Can Teen 2*, *China Garden* and *Passion*, respectively. We report the building time, the space consumption, the query response time and the error of our algorithm and each baseline considered in our experiment as follows in Table 7. In our results, we observe that (1) K-Algo has a significantly high query time which is more than five seconds and intolerable for mobile app users and will worsen the user experience a lot; (2) SP-Oracle incurs a very large space consumption which is heavy-weight for mobile apps and its query time is still not good enough (i.e., more than one second) for the real-time application although it is smaller than that of K-Algo; (3) Our algorithm SE has the smallest building time, the smallest space consumption and the smallest query time. It incurs neglectable storage overhead and neglectable query response time, and highly boosts the quality of experience. Its building time, space consumption and query response time are smaller than that of SP-Oracle by several orders of magnitudes.

Algorithm	Building Time (s)	Space Consumption (KB)	Query Response Time (ms)	Error
K-Algo.	-	-	5,920	0.0002
SP-Oracle	2,160	150,000	1,580	0.0003
SE	50	1,058	0.254	0.0008

Table 7. Performances of All Algorithms in Our Case Study of kNN Query

Case Study of Shortest Path Query. We consider the shortest path query from the location of the atrium, which is a landmark of the university, and also the lobby of the main building to the canteen ‘Passion’. Figure 36 demonstrates the user interface of the Path Advisor and the details of this query mentioned above. In this figure, there is a menu on the left side which provides the search option and also legends. On the right side, it shows the 3D terrain model of the campus. The desired shortest path is shown in the green path in Figure 36. We report the experimental results of our algorithm and all baselines considered in Table 8 below. In

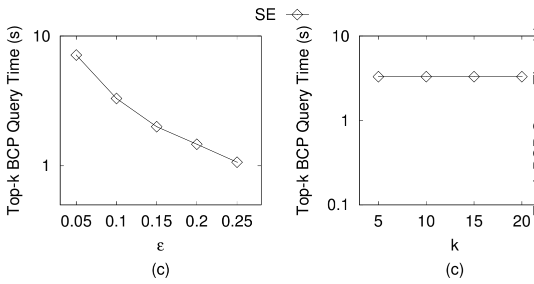

Fig. 37. **Effect of ϵ and k on real dataset, BearHead, for Top-k BCP**

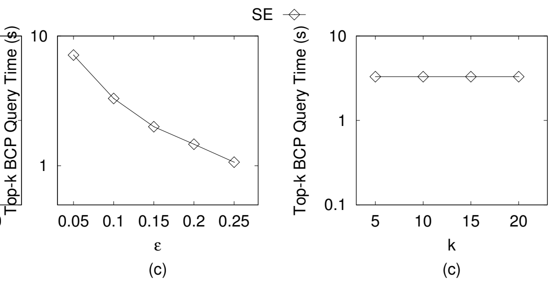

Fig. 38. **Effect of ϵ and k on real dataset, EaglePeak, for Top-k BCP**

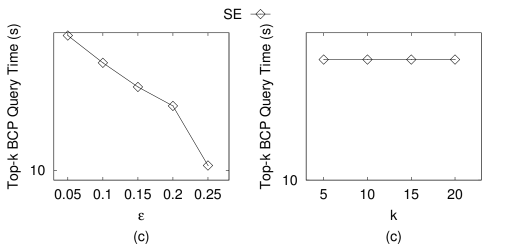

Fig. 39. **Effect of ϵ and k on real dataset, San Fransisco, for Top-k BCP**

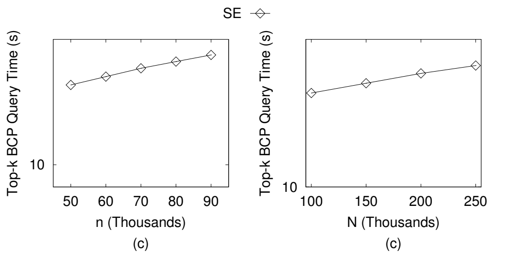

Fig. 40. **Effect of n and N on dataset, San Francisco South, for Top-k BCP query**

our results, we observe that (1) K-Algo suffers from its very large query time (i.e., larger than 5 seconds) which prevent from its usage in this Path Advisor system which requires real-time response; (2) SP-Oracle consumes too much memory and incurs intolerable space overhead for mobile apps. Besides, it is still not efficient enough for the query processing although it is smaller than that of K-Algo; (3) Our algorithm SE has the smallest building time, the smallest space consumption and the smallest query time which is smaller than that of SP-Oracle by orders of magnitudes. It is quite light-weight and enjoys its neglectable storage overhead and instant query response.

Algorithm	Building Time (s)	Space Consumption (KB)	Query Response Time (ms)	Error
K-Algo.	-	-	5,281	0.0035
SP-Oracle	2,160	150,000	687	0.0013
SE	50	1,058	0.087	0.0023

Table 8. **Performances of All Algorithms in Our Case Study of Shortest Path Query**

6.2.6 Experimental Result Summary. We conclude that our *SE* structure is efficient in terms of oracle building time, oracle size as well as space and is accurate and efficient for the distance query and path query. It is scalable to n, N and k. SP-Oracle is not efficient in terms of space which is memory infeasible for the three real datasets when the error parameter is sufficiently small. Besides, our oracle, *SE*, performs much faster than the state-of-the-art by

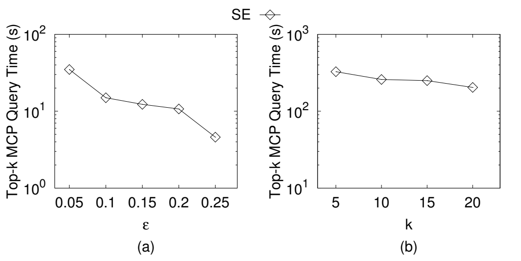

Fig. 41. **Effect of ϵ and k on real dataset, BearHead, for Top-k MCP**

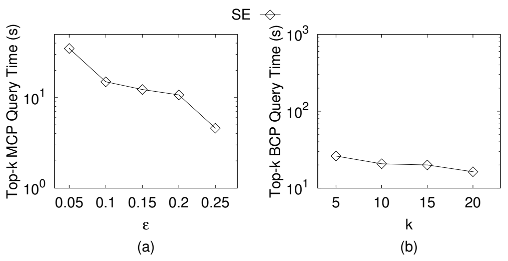

Fig. 42. **Effect of ϵ and k on real dataset, EaglePeak, for Top-k MCP**

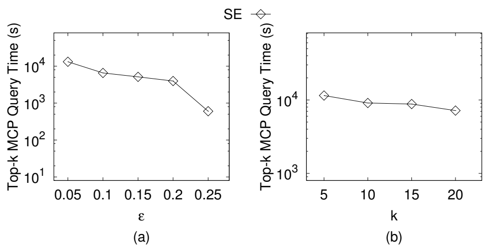

Fig. 43. **Effect of ϵ and k on real dataset, San Fransisco, for Top-k MCP**

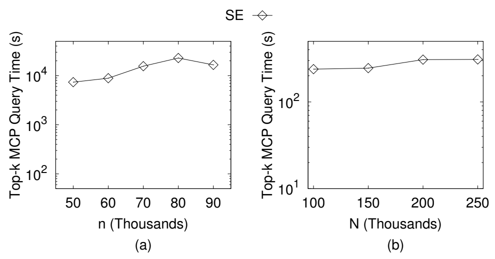

Fig. 44. **Effect of n and N on dataset, San Francisco South, for Top-k MCP query**

up to 2-6 orders of magnitude for distance query and path query. For the monochromatic and bichromatic k nearest neighbor and k farthest neighbor queries, our method outperforms the best-known algorithm, SI by up to 1-6 orders of magnitudes in terms of the query processing time. Our proposed algorithm is also efficient and scalable for the monochromatic and bichromatic top-k closest-pair and top-k farthest-pair queries.

7 CONCLUSION

In this paper, we studied several important spatial queries, including the shortest distance and path query, monotonic and bichromatic k nearest neighbor and farthest neighbor queries and monotonic and bichromatic top-k closest-pair and farthest-pair queries, which are fundamental to many other spatial queries and many data mining applications. We proposed a distance and path oracle called *SE* which has three good features: (1) low construction time, (2) small size and (3) low query time (compared with the state-of-the-art). Our experimental studies show that *SE* consistently outperforms the best existing algorithms in terms of all measurements for the queries studied. There are several interesting research directions. One of them is to study how to efficiently update the distance oracle when there is an update on some POIs.

Acknowledgements: We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Victor Junqiu Wei is supported by PolyU internal fund (1-BD47) under the research project (P0039657) of the Hong Kong Polytechnic University. The research of the HKUST side is supported by PRP/026/21FX. This research of Cheng Long is supported in part by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Awards MOE-T2EP20220-0011 and MOE-T2EP20221-0013). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore. The research of Hanan Samet was sponsored in part by the NSF under Grants IIS-18-16889, IIS-20-41415, and IIS-21-14451.

REFERENCES

[1] A. Al-Badarneh, H. Najadat, and A. Alraziqi. A classifier to detect tumor disease in mri brain images. In *ASONAM*, 2012.

[2] L. Aleksandrov, H. N. Djidjev, H. Guo, A. Maheshwari, D. Nussbaum, and J.-R. Sack. Algorithms for approximate shortest path queries on weighted polyhedral surfaces. In *Discrete & Computational Geometry*, 2010.

[3] L. Aleksandrov, H. N. Djidjev, H. Guo, A. Maheshwari, D. Nussbaum, and J.-R. Sack. Algorithms for approximate shortest path queries on weighted polyhedral surfaces. *Discrete & Computational Geometry*, 2010.

[4] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *JACM*, 2005.

[5] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, 2006.

[6] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *JACM*, 1995.

[7] J. Chen and Y. Han. Shortest paths on a polyhedron. In *SOCG*, 1990.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 3rd edition, 2009.

[9] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest pair queries in spatial databases. *ACM SIGMOD Record*, 2000.

[10] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Algorithms for processing k-closest-pair queries in spatial databases. *Data & Knowledge Engineering*, 2004.

[11] K. Deng, H. T. Shen, K. Xu, and X. Lin. Surface k-nn query processing. In *ICDE*, 2006.

[12] K. Deng and X. Zhou. Expansion-based algorithms for finding single pair shortest path on surface. In *WWGIS*. 2005.

[13] 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. *VLDBJ*, 2008.

[14] B. G. Dickson and P. Beier. Quantifying the influence of topographic position on cougar (puma concolor) movement in southern california, usa. *Journal of Zoology*, 2007.

[15] H. N. Djidjev and C. Sommer. Approximate distance queries for weighted polyhedral surfaces. In *ESA*. 2011.

[16] M. Fan, H. Qiao, and B. Zhang. Intrinsic dimension estimation of manifolds by incising balls. *Pattern Recognition*, 2009.

[17] J. Fischer and S. Har-Peled. Dynamic well-separated pair decomposition made easy. In *CCCG*, 2005.

[18] F. Fodor. The densest packing of 12 congruent circles in a circle. *Contributions to Algebra and Geometry*, 2000.

[19] J. Golay and M. Kanevski. A new estimator of intrinsic dimension based on the multipoint morisita index. *Pattern Recognition*, 2015.

[20] S. A. Huettel, A. W. Song, and G. McCarthy. Functional magnetic resonance imaging. In *Sinauer Associates*, 2004.

[21] J. Jia and C.-K. Tang. Image repairing: Robust image synthesis by adaptive nd tensor voting. In *CVPR*. IEEE, 2003.

[22] J. Jia and C.-K. Tang. Tensor voting for image correction by global and local intensity alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005.

[23] T. Kanai and H. Suzuki. Approximate shortest path on a polyhedral surface based on selective refinement of the discrete graph and its applications. In *GMPTA*, 2000.

[24] M. Kaul, R. C.-W. Wong, and C. S. Jensen. New lower and upper bounds for shortest distance queries on terrains. *VLDB*, 2015.

[25] M. Kaul, R. C.-W. Wong, B. Yang, and C. S. Jensen. Finding shortest paths on terrains by killing two birds with one stone. *VLDB*, 2013.

[26] B. Kégl. Intrinsic dimension estimation using packing numbers. In *NIPS*, 2002.

[27] M. Kortgen, G. J. Park, M. Novotni, and R. Klei. 3d shape matching with 3d shape contexts. In *CESCG*, 2003.

[28] J.-F. Lalonde, N. Vandapel, D. F. Huber, and M. Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of field robotics*, 2006.

[29] M. Lanthier, A. Maheshwari, and J.-R. Sack. Approximating shortest paths on weighted polyhedral surfaces. pages 527–562, 2001.

[30] L. Liu and R. C.-W. Wong. Finding shortest path on land surface. In *SIGMOD*, 2011.

[31] A. Mårell, J. P. Ball, and A. Hofgaard. Foraging and movement paths of female reindeer: insights from fractal analysis, correlated random walks, and lévy flights. *Canadian Journal of Zoology*, 2002.

[32] J. S. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 1987.

[33] J. Sankaranarayanan, H. Alborzi, and H. Samet. Distance join queries on spatial networks. *In* Proceedings of the 14th ACM International Symposium on Advances in Geographic Information Systems*, pages 211–218, Arlington, VA, November 2006.*

[34] J. Sankaranarayanan and H. Samet. Distance oracles for spatial networks. *In* Proceedings of the 25th IEEE International Conference on Data Engineering*, pages 652–663, Shanghai, China, April 2009.*

[35] J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. IEEE Transactions on Knowledge and Data Engineering*, 22*(*8*)*:1158–1175, August 2010.* (*Best Papers of ICDE 2009 Special Issue.*), pages 1158–1175.

[36] J. Sankaranarayanan and H. Samet. Roads belong in databases. IEEE Data Engineering Bulletin*, 33*(*2*)*:4–11, June 2010.*, page Invited paper.

[37] J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. PVLDB*, 2*(*1*)*:1210–1221, August 2009.*, page Also *Proceedings of the 35th International Conference on Very Large Data Bases* (*VLDB*).

[38] L. T. Sarjakoski, P. Kettunen, H.-M. Flink, M. Laakso, M. Rönneberg, and T. Sarjakoski. Analysis of verbal route descriptions and landmarks for hiking. *Personal and Ubiquitous Computing*, 2012.

[39] C. Shahabi, L.-A. Tang, and S. Xing. Indexing land surface for efficient knn query. *VLDB*, 2008.

[40] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *ECCV*, 2006.

[41] F. Tauheed, L. Biveinis, T. Heinis, F. Schurmann, H. Markram, and A. Ailamaki. Accelerating range queries for brain simulations. In *ICDE*, 2012.

[42] N. Tran, M. J. Dinneen, and S. Linz. Close weighted shortest paths on 3d terrain surfaces. In *The ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems 2021* (*SIGSPATIAL*), pages 597–607, 2020.

[43] N. Vandapel, R. R. Donamukkala, and M. Hebert. Unmanned ground vehicle navigation using aerial ladar data. *The International Journal of Robotics Research*, 2006.

[44] V. J. Wei, R. C.-W. Wong, C. Long, and D. M. Mount. Distance oracle on terrain surface. In *SIGMOD*, 2017.

[45] S.-Q. Xin and G.-J. Wang. Improving chen and han's algorithm on the discrete geodesic problem. *ACM Trans. Graph.*, 2009.

[46] S. Xing, C. Shahabi, and B. Pan. Continuous monitoring of nearest neighbors on land surface. In *VLDB*, 2009.

[47] D. Yan, Z. Zhao, and W. Ng. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *CIKM*, 2012.

[48] Y. Yan and R. C.-W. Wong. Path advisor: a multi-functional campus map tool for shortest path. *VLDB*, 2021.

A LARGEST CAPACITY DIMENSION

Consider a metric space X with a distance metric $d(\cdot,\cdot)$. Given a positive real number r, a set $Y \subseteq X$ is said to be *r-separated* if for any two distinct points $x, y \in Y$, $d(x,y) \geq r$. Given a positive real number r and a set $S \subseteq X$, the *r-packing number* of S, denoted by $M(r,S)$, is defined to be the maximum cardinality of an r-separated subset of S. Given a metric space X and a distance measure d, a set $S \subseteq X$ and two positive real numbers r_1 and r_2, the *scale-dependent capacity dimension* of S w.r.t. r_1 and r_2, denoted by $\mathcal{D}(S, r_1, r_2)$, is defined to be $-\frac{\log M(r_1,S)-\log M(r_2,S)}{\log r_1 - \log r_2}$ [26]. This dimension is used to measure the '*intrinsic dimension*' of a metric space. Many high-dimensional data points are believed to be distributed in a manifold with a low '*intrinsic dimension*'. Intuitively, the '*intrinsic dimension*' is the number of independent variables needed to represent the whole dataset. A good estimate of the '*intrinsic dimension*' could be used to set the input parameters of the dimension reduction algorithms (e.g., Principle Component Analysis). There are many different specific formulations of the '*intrinsic dimension*' capturing certain properties [5, 16, 19, 26]. The *scale-dependent capacity dimension* captures the geometric property of the data and provides a multi-resolution dimensionality which depends on the radius r. It measures the growth rate of $M(r,S)$ w.r.t. r of a subset S of X. In our context, the data space X is the set P of all the n POIs and the distance metric $d(\cdot,\cdot)$ is the geodesic distance $d_g(\cdot,\cdot)$. Then, we give the definition of a 'ball' on a terrain surface. Given a point $p \in P$ and a non-negative real number r, a ball centered at p with radius equal to r, denoted by $B(p,r)$, is defined to be a set of all POIs in the disk $D(p,r)$. Then, we give the definition of the *capacity dimension* of a ball $B(p,r)$ on the terrain surface which only measures the growth rate of $M(r, B(p,r))$ when r reduces from $2r$ to $\frac{r}{2}$.

Definition A.1. Given a ball $B(p,r)$ on a terrain surface, where p is a point in P and r is a positive real number, the capacity dimension of $B(p,r)$ is defined to be $\mathcal{D}(B(p,r), 2r, \frac{r}{2}) = -\frac{\log M(2r,B(p,r)) - \log M(\frac{r}{2},B(p,r))}{\log 2r - \log \frac{r}{2}} = 0.5 \log \frac{M(\frac{r}{2},B(p,r))}{M(2r,B(p,r))}$.

Consider a set of points whose pairwise geodesic distances are at least $2r$. Since the disk $D(p,r)$ could overlap with at most 2 of them, we obtain that $M(2r, B(p,r)) = 2$. Thus, for a given ball $B(p,r)$, the *capacity dimension* is equal to $0.5 \log \frac{M(\frac{r}{2},B(p,r)}{2})$. Thus, we obtain that given a disk $D(p,r)$, $2 \cdot 2^{2\mathcal{D}(B(p,r),2r,\frac{r}{2})}$ is the maximum number of POIs whose pairwise geodesic distances are at least $\frac{r}{2}$ such that the disk $D(p,r)$ contains them.

Next, we define the *largest capacity dimension* of a set P of POIs on a terrain surface to be $\max_{p \in P, r \in (0,\infty)} D(B(p,r), 2r, \frac{r}{2})$, denoted by β. By the definition of the *largest capacity dimension*, any disk $D(p,r)$, where p is a point in P and r is a positive real number, could contain at most $2 \cdot 2^{2\beta}$ (i.e. $O(2^{2\beta})$) POIs whose pairwise geodesic distance is at least $\frac{r}{2}$.

The definition of *largest capacity dimension* has an equivalent presentation as follows. Given a set P of POIs on a terrain surface, its *largest capacity dimension* β is the largest positive real number such that for any point p in P and any non-negative real number r, the disk $D(p,r)$ overlaps with at most $2 \cdot 2^{2\beta}$ disjoint disks each with radius at least $\frac{r}{4}$. [18] proved that in the 2D Euclidean space, a disk with radius r overlaps with at most 12 disjoint disks with radii equal to $\frac{r}{4.029}$. Thus, in the 2D Euclidean space, a disk with radius r overlaps with at most 12 disjoint disks with radii equal to $\frac{r}{4}$. Based on this, we obtain that in an extreme case where the terrain surface is a 2D plane, the *largest capacity dimension* β is at most 1.3. In general cases, intuitively, β is a little bit larger than 1.3, since the terrain surface could be regarded as a 2D surface with some fluctuations in terms of height.

B PROOF

Lemma B.1. *Given disks $D_1(c_1, r_1)$ and $D_2(c_2, r_2)$ where (1) c_1 and c_2 are two points in P and (2) r_1 and r_2 are two non-negative real numbers, for any point p_1 in D_1 and any point p_2 in D_2, $d_g(c_1, c_2)$ is an ϵ-approximate distance of $d_g(p_1, p_2)$ if $d_g(c_1, c_2) \geq (\frac{2}{\epsilon} + 2) \cdot \max\{r_1, r_2\}$.*

Proof. By Triangle Inequality, we obtain that $d_g(c_1, c_2) - d_g(p_1, c_1) - d_g(p_2, c_2) \leq d_g(p_1, p_2) \leq d_g(c_1, c_2) + d_g(p_1, c_1) + d_g(p_2, c_2)$. Thus, we obtain that $d_g(p_1, p_2) - r_1 - r_2 \leq d_g(c_1, c_2) \leq d_g(p_1, p_2) + r_1 + r_2$. We further obtain that $d_g(p_1, p_2) \geq (\frac{2}{\epsilon} + 2) \cdot \max\{r_1, r_2\} - r_1 - r_2 \geq (\frac{2}{\epsilon} + 2) \cdot \max\{r_1, r_2\} - 2\max\{r_1, r_2\} = \frac{2}{\epsilon} \cdot \max\{r_1, r_2\}$. By the two inequalities obtained above, we obtain that $d_g(p_1, p_2) - \epsilon \cdot d_g(p_1, p_2) \leq d_g(c_1, c_2) \leq d_g(p_1, p_2) + \epsilon \cdot d_g(p_1, p_2)$. □

Proof of Lemma 3.2. By Step (b) in the partition tree construction algorithm, the tree built satisfies Separation Property and Covering Property. Then, we prove that it also satisfies the distance property: Consider a node O and any of its descendants O'. Let $(O, O_1, O_2, O_3, ..., O_t, O')$ denote the path from O to O' in the partition tree. By Separation Property, we obtain that $r_{O_i} = \frac{r_O}{2^i}(1 \leq i \leq t), r_{O_t} = 2 \cdot r_{O'}$. By our building method, we obtain that $d_g(c_{O_i}, c_{O_{i+1}}) \leq r_{O_i}(1 \leq i \leq t-1), d_g(c_O, c_{O_1}) \leq r_O, d_g(c_{O_t}, c_{O'}) \leq r_{O_t}$. By Triangle Inequality, we obtain that $d_g(c_O, c_{O'}) \leq d_g(c_O, c_{O_1}) + \sum_{i=1}^{t-1} d_g(c_{O_i}, c_{O_{i+1}}) + d_g(c_{O_t}, c_{O'})$. By integrating all the inequalities above, we obtain that $d_g(c_O, c_{O'}) \leq \sum_{i=0}^{t} \frac{r_O}{2^i} \leq 2 \cdot r_O$. □

Proof of Lemma 3.3. Since $\forall i \in [0, h-1], r_i = 2 \cdot r_{i+1}$, we obtain that $h = \log \frac{r_0}{r_h}$. We obtain that $r_0 \leq \max_{p,q \in P} d_g(p, q)$ since r_0 is a geodesic distance between two POIs (the center of the root and its farthest neighbor). It is obvious that $r_{h-1} \geq \min_{p,q \in P} d_g(p, q)$, since otherwise, $\forall p \in P$, the disk $D(p, r_{h-1})$ contains only 1 POI and the construction algorithm stops at layer h-1. Since $r_h = \frac{r_{h-1}}{2}$, we obtain that $r_h \geq 0.5 \cdot \min_{p,q \in P} d_g(p, q)$ by contradiction.

Finally, we obtain that $h \leq \log \frac{\max_{p,q \in P} d_g(p,q)}{0.5 \cdot \min_{p,q \in P} d_g(p,q)}$. □

Proof of Theorem 3.6. By the algorithm for generating S, we obtain that every pair in S is well-separated at the end. Then, we prove that for any two points p and q in P, there is exactly one pair $\langle O, O' \rangle$ containing $\langle p, q \rangle$ in S. Consider each iteration of the procedure presented in Section 3.3. We proceed to prove the statement: there is exactly one node pair in S containing $\langle p, q \rangle$ at the end of the iteration if at the beginning of the iteration, there is exactly one node pair, denoted by $\langle O_1, O_2 \rangle$, in S containing $\langle p, q \rangle$. If $\langle O_1, O_2 \rangle$ is not extracted in the iteration, then $\langle O_1, O_2 \rangle$ is still the only one containing $\langle p, q \rangle$ at the end of the iteration. Otherwise, $\langle O_1, O_2 \rangle$ is extracted and w.l.o.g., we assume that O_2 is split. Since only one child of O_2 is O_q or an ancestor of O_q, exactly one newly inserted node pair contains $\langle p, q \rangle$. Besides, it must be true that at the beginning of the first iteration, exactly one node pair (i.e. $\langle O_{root}, O_{root} \rangle$) contains $\langle O_1, O_2 \rangle$. By induction, we obtain that at the end of the final iteration, exactly one node pair in S contains $\langle p, q \rangle$.

Consider the unique pair $\langle O, O' \rangle$ containing $\langle p, q \rangle$ in the node pair set of our SE. Since $\langle O, O' \rangle$ is well-separated, then $d_g(O, O') \geq (\frac{2}{\epsilon} + 2) \cdot \max\{r'_O, r'_{O'}\}$, where r'_O (resp., $r'_{O'}$) denote the radius of the enlarged disk of O (resp., O'). Since the enlarged disk of O (resp., O') contains p and q by Distance Property, we obtain that $d_g(O, O') = d_g(c_O, c_{O'})$ is an ϵ-approximate distance of $d_g(p, q)$ by Lemma B.1. □

Lemma B.2. *Consider a chain of node pairs $\langle O_1, O'_1 \rangle, \langle O_2, O'_2 \rangle, ..., \langle O_i, O'_i \rangle, ..., \langle O_m, O'_m \rangle$, where $\langle O_i, O'_i \rangle$ is generated by $\langle O_{i-1}, O'_{i-1} \rangle$ for each integer $i \in [2, m]$. Let $r_{\overline{O_i}}$ denote $\max\{r_{O_i}, r_{O'_i}\}$ for each integer $i \in [1, m]$. $\forall k, j \in [1, m]$, $r_{\overline{O_k}} \geq r_{\overline{O_j}}$ if and only if $k < j$.*

Proof of Lemma B.2. It is easy to see that $\forall k, j \in [1, m]$ where $k < j$, O_k (resp., O'_k) is O_j (resp., O'_j) or an ancestor of O_j (resp., O'_j) in $T_{compress}$. Thus, we obtain that $r_{O_k} \geq r_{O_j}$ and $r_{O'_k} \geq r_{O'_j}$. Since $r_{\overline{O_k}} = \max\{r_{O_k}, r_{O'_k}\}$ and $r_{\overline{O_j}} = \max\{r_{O_j}, r_{O'_j}\}$, we obtain that $r_{\overline{O_k}} \geq r_{\overline{O_j}}$. □

Proof of Lemma 3.8. Consider the chain of pairs $\langle O_1, O'_1\rangle, \langle O_2, O'_2\rangle, ..., \langle O_i, O'_i\rangle, ..., \langle O_m, O'_m\rangle$, where $O_1 = O'_1 = O_{root}$, $O_m = O$, $O'_m = O'$ and $\langle O_i, O'_i\rangle$ is *generated by* $\langle O_{i-1}, O'_{i-1}\rangle$ for all integer i in $[2, m]$ in our method of constructing the node pair set of SE. Consider the case that $\langle O, O'\rangle$ is a first-higher-layer node pair. We denote the parent of O' in $T_{compress}$ by $parent(O')$, there must exist an integer k such that $k \in [1, m-1]$ and $parent(O')$ is *split* from $\langle O_k, O'_k\rangle$. This is because otherwise, $\langle O, O'\rangle$ would not be generated. Consider the pair $\langle O_k, O'_k\rangle$ from which $parent(O')$ is split and thus, $r_{parent(O')} = \max\{r_{O_k}, r_{O'_k}\}$. By Lemma B.2, we obtain $r_{parent(O')} \geq \max\{r_O, r_{O'}\}$ and thus the layer containing $parent(O')$ is higher than or equal to that containing O. For the case that $\langle O, O'\rangle$ is a first-lower-layer node pair, the proof is symmetric and we omit the details. □

Proof of Lemma 3.11. Consider a node pair $\langle O, O'\rangle$ considered in the procedure described in Section 3.3, where $\langle O, O'\rangle \neq \langle O_{root}, O_{root}\rangle$. Let $parent(O)$ (resp., $parent(O')$) denote the parent of O (resp., O') in $T_{compress}$ if O (resp., O') is not the root node. It is obvious that $\langle O, O'\rangle$ is generated by $\langle O, parent(O')\rangle$ or $\langle parent(O), O'\rangle$.

W.l.o.g., we assume that $\langle O, O'\rangle$ is generated by $\langle parent(O), O'\rangle$. By Lemma B.2, we obtain that $r_{parent(O)} \leq r_{parent(O')}$ since $parent(O')$ is split before $\langle parent(O), O'\rangle$ is generated. By Lemma 3.8, we obtain that $r_{O'} \leq r_{parent(O)}$ and $r_O \leq r_{parent(O')}$. Let $\overline{O}$ denote the child of $parent(O)$ in the original partition tree T_{org} which is on the path from $parent(O)$ to O. There are two cases of $\overline{O}$ and we present that in both cases, it is true that $c_{\overline{O}} = c_O$. If $O = \overline{O}$, it is obvious that $c_{\overline{O}} = c_O$. Then, consider the case where $O \neq \overline{O}$. Since any node on the path from $\overline{O}$ to O excluding O must have only one child, we obtain that $c_{\overline{O}} = c_O$ by Step (i) of the building algorithm of T_{org}. Similarly, the child O_c of $parent(O')$ in the original partition tree T_{org} which is on the path from $parent(O')$ to O' has the same center with O. Since $r_{parent(O)} \leq r_{parent(O')}$, we obtain that $r_{\overline{O}} \leq r_{O_c}$. Then, consider the node $\overline{O}'$ on the path from O_c to O' in T_{org} which is on the same layer as $\overline{O}$ and has the same center as O_c (By Step (i), we could always find such a node $\overline{O}$). Since $c_{\overline{O}'} = c_{O'}$, we obtain that $d_g(\overline{O}, \overline{O}') = d_g(O, O')$. Since $\langle parent(O), O'\rangle$ is not a well-separated pair, we obtain that $d_g(parent(O), O') \leq 2(\frac{2}{\epsilon} + 2) \cdot \max\{r_{parent(O)}, r_{O'}\} = 2(\frac{2}{\epsilon} + 2)r_{parent(O)}$. Thus, we obtain that $d_g(parent(O), \overline{O}') \leq 2(\frac{2}{\epsilon} + 2)r_{parent(O)}$. Since $\overline{O}$ is a child of $parent(O)$ in the original partition tree T_{org}, we obtain that $d_g(parent(O), \overline{O}) \leq r_{parent(O)} = 2r_{\overline{O}}$. By triangle inequality, we obtain that $d_g(\overline{O}, \overline{O}') \leq d_g(parent(O), \overline{O}') + d_g(\overline{O}, parent(O)) \leq 2(\frac{2}{\epsilon} + 2)r_{parent(O)} + 2r_{\overline{O}} = 4(\frac{2}{\epsilon} + 2)r_{\overline{O}} + 2r_{\overline{O}} = (\frac{8}{\epsilon} + 10)r_{\overline{O}}$. Thus, $\langle \overline{O}, \overline{O}'\rangle$ must be an enhanced node pair, where $c_{\overline{O}} = c_O$ and $c_{\overline{O}'} = c_{O'}$. □

Lemma B.3. *The maximum number of child nodes of each node O in a partition tree or a compressed partition tree is $O(2^{2\beta})$.*

Proof. By the definition of the partition tree, the center of each children of O must lie in the disk $D(c_O, r_O)$. Besides, by the Separation Property, the minimum pairwise distance of all its children must be at most $\frac{r_O}{2}$. Thus, by the definition of the *largest capacity dimension* β, we obtain that the maximum number of children of each node O in a partition tree is $O(2^{2\beta})$. It is easy to see that the converting from a partition tree to a compressed partition tree does not change the number of children of any undeleted node. Thus, we obtain that the maximum number of children of each node O in a compressed partition tree is $O(2^{2\beta})$. □

Lemma B.4. *Any disk $D(c, r)$, where $c \in P$ and r is a non-negative real number, can hold at most $O((2^{2\beta})^i)$ points, the minimum pairwise geodesic distance of which is at least $\frac{r}{2^i}$.*

Proof. Consider a set $PSET$ of points in $D(c, r)$ such that their minimum pairwise geodesic distance is at least $\frac{r}{2^i}$. We first build a partition tree upon $PSET$ as follows: first, we create the root to be $O(c_O = c, r_O = r)$ instead of following Step(a) and the building procedure of other nodes is the same as Step(b). Since the radius of each non-root node is half of its parent's radius, we obtain that there are totally i layers in the tree. By Lemma B.3, we obtain that the number of children of any node in the compressed partition tree is at most $O(2^{2\beta})$. Thus, the number of nodes in Layer i is at most $O(2^{2\beta})$ times that in Layer $i-1$, where $0 < i \leq h$. Thus, we obtain that there are at most $O((2^{2\beta})^i)$ nodes in the Layer i. In other words, $PSET$ has at most $O((2^{2\beta})^i)$ points. □

Proof of Theorem 3.7. **Proof Sketch.** To give the intuition of the proof, we present an intermediate node pair set S' which is conceptual. Let T' denote the tree which is the same as the original partition tree except that the radius of each leaf node is 0. S' denote a node pair set built by the node pair generating algorithm presented in Section 3.3 which takes T' as input instead of the compressed partition tree. It is clear from a high-level intuition that the node pair set S of SE is not larger than S' and the number of the node pairs considered in the process of generating S is $O(|S|)$ (see the full proof for the details). In the following, we denote r_{O_x} as the radius of a node O_x in the original partition tree. Consider a node O in T' and a set $\mathcal{S}'(O)$ which is $\{O' | \langle O, O' \rangle$ or $\langle O', O \rangle$ is in S' and $r_O \geq r_{O'}\}$. Note that $\cup_{O \in T} \mathcal{S}'(O) = S'$. By the node pair generating algorithm, we obtain that for each node O' in $\mathcal{S}'(O)$, 1. O' is in the same layer as O or one layer lower than O (see Lemma B.5) 2. there is a upper bound on $d_g(O, O')$, i.e., O' lies in a disk D centered at $c_{O'}$ with a r_O- and ϵ-related radius (see Lemma B.6). Then, together with a property (see Lemma B.4) derived from β, we obtain that $|\mathcal{S}'(O)| = O((\frac{1}{\epsilon})^{2\beta})$ and thus $|S'| = O((\frac{1}{\epsilon})^{2\beta} nh)$.

Detailed Proof. Now, we delve into the detailed proof and adopt the same notations as shown in the proof sketch.

Lemma B.5. $\forall O' \in \mathcal{S}'(O), r_{O'} \leq r_O \leq 2 \cdot r_{O'}$

Proof. Since $parent(O')$ is split before the node pair $\langle O, O' \rangle$ is generated, by Lemma B.2, we obtain $r_{parent(O')} \geq r_O$.

Since $r_{parent(O')} = 2 \cdot r_{O'}$, we obtain that $r_{O'} \leq r_O \leq 2 \cdot r_{O'}$. □

Lemma B.6. $\forall O' \in \mathcal{S}'(O), d_g(c_O, c_{O'}) \leq (4\frac{2}{\epsilon} + 10) \cdot r_O$.

Proof. By our node pair set generating algorithm presented in Section 3.3, $\langle O, O' \rangle$ is generated by $\langle parent(O), O' \rangle$ or $\langle O, parent(O') \rangle$ and the node pair which generated $\langle O, O' \rangle$ is not well separated. Consider the case where $\langle O, O' \rangle$ is generated by $\langle O, parent(O') \rangle$ (the analysis of the case where $\langle O, O' \rangle$ is generated by $\langle parent(O), O' \rangle$ is symmetric, i.e., just with O and O' swapped, and has the same result and thus, we do not present this case for the sake of space). We obtain that $d_g(c_O, c_{parent(O')}) \leq 2 \cdot (\frac{2}{\epsilon} + 2) \cdot \max\{r_O, r_{parent(O')}\}$, where $r_{parent(O')} = 2r_{O'}$ by the definition of the partition tree. By Lemma B.5, we obtain that $d_g(c_O, c_{parent(O')}) \leq 2 \cdot (\frac{2}{\epsilon} + 2) \cdot r_{parent(O')}$. By Triangle Inequality, we obtain that $d_g(c_O, c_{O'}) \leq d_g(c_O, c_{parent(O')}) + d_g(c_{O'}, c_{parent(O')})$. By the definition of the partition tree, we obtain that

$d_g(c_{O'}, c_{parent(O')}) \leq r_{parent(O')}$. Thus, we obtain that $d_g(c_O, c_{O'}) \leq 2 \cdot (\frac{2}{\epsilon} + 2) \cdot r_{parent(O')} + r_{parent(O')}$. By Lemma B.5, we obtain that $d_g(c_O, c_{O'}) \leq (4\frac{2}{\epsilon} + 10) \cdot r_O$. □

Let $\mathcal{S}''$ be a point set containing the centers of all nodes in $\mathcal{S}'(O)$. By Lemma B.5, we obtain that O' is either in the same layer as O in the partition tree or one layer lower than O in the partition tree. Let $\mathcal{S}_1''$ (resp., $\mathcal{S}_2''$) denote $\{O''|O'' \in \mathcal{S}'', O''$ is in the same layer as $O\}$ (resp., $\{O''|O'' \in \mathcal{S}'', O''$ is one layer lower than $O\}$).

By the Separation Property, we obtain that the minimum pairwise geodesic distance of $\mathcal{S}_1''$ (resp., $\mathcal{S}_2''$) must be at least r_O (resp., $\frac{r_O}{2}$). By Lemma B.4, we obtain that the Disk $D(r_O, (4\frac{2}{\epsilon} + 10) \cdot r_O)$ can hold $O((2^{2\beta})^{\log(4\frac{2}{\epsilon}+10)})$ (resp., $O((2^{2\beta})^{\log(2(4\frac{2}{\epsilon}+10))})$) points whose minimum pairwise geodesic distance is at least r_O (resp., $\frac{r_O}{2}$). Thus, we obtain that $|\mathcal{S}'(O)| \leq 2 \cdot (2^{2\beta})^{\log(2\cdot(4\frac{2}{\epsilon}+10))} = O((\frac{1}{\epsilon})^{2\beta})$. There are at most nh such node O in T. Thus, we obtain that $|S'|$ is $O(\frac{nh}{\epsilon^{2\beta}})$ since $\cup_{O \in T} \mathcal{S}'(O) = S'$.

Next, we prove that $|S|$ is at most $|S'|$ (i.e. $O(\frac{nh}{\epsilon^{2\beta}})$), where S is the node pair set of SE. Consider a node pair $\langle O, O' \rangle$ in S. We denote the node in T' which comes from the same node in the partition tree T_{org} as O (resp., O') by O_x (resp., O_y). Since $\langle O, O' \rangle$ are well-separated, $\langle O_x, O_y \rangle$ are well-separated and thus, we could find a node pair $\langle \overline{O_x}, \overline{O_y} \rangle$ in S' containing $\langle O_x, O_y \rangle$. We call $\langle \overline{O_x}, \overline{O_y} \rangle$ the corresponding pair of $\langle O, O' \rangle$. Let O_p (resp., O_p') denote the node in T' which comes from the same node in T_{org} as the parent of O (resp., O') in $T_{compress}$. O_p and O_p' are not well separated, since otherwise, $\langle O, O' \rangle$ could not be generated. $\overline{O_x}$ (resp., $\overline{O_y}$) must be on the path from O_p (resp., O_p') to O_x (resp., O_y) and any node on the path excluding O_x and O_p (resp., O_y and O_p') must have only one child. Thus, we obtain that any two different node pairs in S must have different corresponding pairs. Thus, we obtain that $|S| \leq |S'| = O(\frac{nh}{\epsilon^{2\beta}})$. Since in each iteration of the procedure of generating S, we extract one pair and insert more than one pair to S. The number of node pairs considered must be at most 2 times the size of the final S and thus it is $O(\frac{nh}{\epsilon^{2\beta}})$. □

Lemma B.7. *The oracle building time of* SE *is* $O(\frac{N \log^2 N}{\epsilon^{2\beta}} + nh \log n + \frac{nh}{\epsilon^{2\beta}})$.

Proof. The oracle building time of SE consists of the time t_{org} of building the original partition tree T_{org}, the time t_{tran} of transforming the partition tree to the compressed partition tree $T_{compressed}$, the time t_{en} of creating all the enhanced edges and the time t_{np} of generating the node pair set of SE. t_{org} consists of the running time of B+-tree operations, the point selection algorithm and all SSAD algorithms invoked. Note that we index a set of points/nodes with a B+-tree and we find the parent of O in Step (iii) with a SSAD algorithm and $d_g(O, O_{parent})$ is at most $2r_O$ due to the covering property. The running time of all the B+-tree operations in t_{org} is $O(nh \log n)$ since there are h layers and each layer needs $O(n)$ insertion/search operation in B+-tree. For the point selection algorithm, the *random selection algorithm* takes $O(nh)$ time since there are at most nh nodes in T_{org} and each takes at most $O(1)$ time. The *greedy selection algorithm* takes $O(nh \log n)$ time since in each layer since it takes $O(n \log n)$ time to create the grid and corresponding B+-tree in each cell and there are at most $O(n)$ heap operations and B+-tree operations. It is obvious that the overall running time of all SSAD algorithms performed in t_{org} is smaller than t_{en}. t_{tran} is $O(nh)$ time since the partition tree has $O(nh)$ nodes. By Theorem 3.7, there are at most $O(\frac{nh}{\epsilon^{2\beta}})$ node pairs considered in the procedure described in Section 3.3 and thus, t_{np} is $O(\frac{nh}{\epsilon^{2\beta}})$. Next, we will analyze t_{en}.

We introduce a new parameter of the terrain surface, denoted by θ. θ is defined to be the largest positive real number such that the number of vertices on the terrain surface in a disk $D(c, r)$ is at least 2^θ times the number of vertices on the terrain surface contained in the disk $D(c, \frac{r}{2})$, where c is a POI on the terrain surface and r is a positive real number at most $\max_{p \in V} d_g(c, p)$. Thus, the number of vertices contained in a disk centered at any POI is at most $\frac{1}{2^\theta}$ times that in a disk with double radius and the same center. For the root node O_{root}, we expand the disk $D(c_{O_{root}}, r_0)$. For any node O in other layers, we expand the disk $D(c_O, l \cdot r_O)$, where $l = \frac{8}{\epsilon} + 10$. Note that $\forall i \in [\lceil \log l \rceil, h], l \cdot r_i \le r_0$. Since $D(c, r_0)$ is a sub-region on the terrain surface, where c is any point on the surface, we obtain that the vertices of terrain visited by SSAD algorithm invoked for each node in layer i is at most N, if $i \in [0, \lceil \log l \rceil - 1]$; otherwise, it is $\frac{N}{2^{\theta(i - \lceil \log l \rceil)}}$.

By Lemma B.3, there are at most $(2^{2\beta})^i$ nodes in Layer i. Thus, we obtain that t_{en} is at most

$$O(\sum_{i=0}^{\lceil \log l \rceil - 1} (2^{2\beta})^i N \log^2 N + \sum_{i=\lceil \log l \rceil}^{h} \frac{(2^{2\beta})^i N \log^2 N}{2^{2\theta(i - \lceil \log l \rceil)}}) = O(N \log^2 N \frac{(2^{2\beta})^{\lceil \log l \rceil} - 1}{2^{2\beta} - 1} + (2^{2\beta})^{\lceil \log l \rceil} \sum_{i=0}^{h - \lceil \log l \rceil} \frac{(2^{2\beta})^i N \log^2 N}{2^{2\theta i}})$$
$$= O((2^{2\beta})^{\lceil \log l \rceil} N \log^2 N \sum_{i=0}^{h} (\frac{2^{2\beta}}{2^{2\theta}})^i)$$

Our empirical study verified that $\theta \ge \beta$. Thus, we obtain that $\frac{2^{2\beta}}{2^{2\theta}} < 1$ and thus t_{en} is $O((2^{2\beta})^{\lceil \log l \rceil - 1} N \log^2 N) = O(\frac{N \log^2 N}{\epsilon^{2\beta}})$. Thus, we obtain that the oracle building time is $O(\frac{N \log^2 N}{\epsilon^{2\beta}} + nh \log n + \frac{nh}{\epsilon^{2\beta}})$. □

Proof of Theorem 3.12. By Lemma B.7, Theorem 3.7, and the analysis in Section 3.4, we obtain the result. □

Proof of Theorem 3.13. The path returned by our algorithm is $\Pi_g(s, c_O) \cdot \Pi_g(c_O, o_1) \cdot \Pi_g(o_1, o_2) \cdot \ldots\ldots \cdot \Pi_g(o_{M-1}, o_M) \cdot \Pi_g(o_M, c_{O'}) \cdot \Pi_g(c_{O'}, t)$, where $(o_1, o_2, \ldots\ldots, o_M) = \mathcal{L}_M(O, O')$ and (O, O') is the node pair in our node pair set containing s and t. By the definition of $\mathcal{L}_M(O, O')$, all the vertices in $\mathcal{L}_M(O, O')$ lie in the shortest geodesic path from c_O to $c_{O'}$ and thus, $\Pi_g(c_O, o_1) \cdot \Pi_g(o_1, o_2) \cdot \ldots\ldots \cdot \Pi_g(o_{M-1}, o_M) \cdot \Pi_g(o_M, c_{O'})$ is equal to $\Pi_g(c_O, c_{O'})$. Then, we obtain that the length of the path returned by our algorithm is equal to $d_g(s, c_O) + d_g(c_O, c_{O'}) + d_g(c_{O'}, t)$. By triangle inequality, we obtain that

$$d_g(s, t) \ge -d_g(s, c_O) + d_g(c_O, c_{O'}) - d_g(c_{O'}, t)$$

Since $\langle O, O' \rangle$ is well-separated, then $d_g(c_O, c_{O'}) \ge (\frac{2}{\epsilon} + 2) \cdot \max\{r'_O, r'_{O'}\}$, where r'_O (resp., $r'_{O'}$) denote the radius of the enlarged disk of O (resp., O'). Since the enlarged disk of O (resp., O') contains s and t by Distance Property, we obtain that $d_g(c_O, c_{O'}) \ge (\frac{2}{\epsilon} + 2) \cdot \max\{r'_O, r'_{O'}\} \ge (\frac{2}{\epsilon} + 2) \cdot \max\{d_g(s, c_O), d_g(c_{O'}, t)\}$. Thus, we obtain that

$$\begin{aligned} d_g(s, t) \ge & - d_g(s, c_O) + d_g(c_O, c_{O'}) - d_g(c_{O'}, t) \\ \ge & d_g(c_O, c_{O'}) - 2 \cdot \max\{d_g(s, c_O), d_g(c_{O'}, t)\} \\ \ge & (1 - \frac{2}{\frac{2}{\epsilon} + 2}) \cdot d_g(c_O, c_{O'}) \end{aligned} \tag{1}$$

$$\begin{aligned} & d_g(s, c_O) + d_g(c_O, c_{O'}) + d_g(c_{O'}, t) \\ \le & d_g(c_O, c_{O'}) + 2 \cdot \max\{d_g(s, c_O), d_g(c_{O'}, t)\} \\ \le & (1 + \frac{2}{\frac{2}{\epsilon} + 2}) \cdot d_g(c_O, c_{O'}) \end{aligned} \tag{2}$$

By Inequality 1 and Inequality 2, we obtain that

$$\frac{d_g(s,c_O)+d_g(c_O,c_{O'})+d_g(c_{O'},t)}{d_g(s,t)} \leq \frac{(1+\frac{2}{\frac{2}{\epsilon}+2})\cdot d_g(c_O,c_{O'})}{(1-\frac{2}{\frac{2}{\epsilon}+2})\cdot d_g(c_O,c_{O'})} = 1+2\cdot\epsilon \quad (3)$$

□

Proof of Lemma 4.1. Let $\widetilde{d_g}(p,q)$ denote the distance between p and q returned by the distance query. Let q'_k (resp.,o'_k) denote the point in $KLIST'$ such that $\widetilde{d_g}(p,q'_k) = \max_{q\in KLIST'} \widetilde{d_g}(p,q)$ (resp., $d_g(p,o'_k) = \max_{q\in KLIST'} d_g(p,q)$). Let $KLIST$ denote the list containing the exact kNN of q in P. Let q_k (resp., o_k) denote a point in $KLIST$ such that $\widetilde{d_g}(p,q_k) = \max_{q\in KLIST} \widetilde{d_g}(p,q)$ (resp., $d_g(p,o_k) = \max_{q\in KLIST} d_g(p,q)$). Recall that the approximate ratio α is $\frac{d_g(p,o'_k)}{d_g(p,o_k)}$. Since the distance error bound is ϵ, we obtain that $\widetilde{d_g}(p,o'_k) \geq (1-\epsilon)d_g(p,o'_k)$. Thus, we obtain that $\alpha \leq \frac{\widetilde{d_g}(p,o'_k)}{d_g(p,o_k)(1-\epsilon)}$. By the definition of o_k, we obtain that $d_g(p,o_k) \geq d_g(p,q_k)$. Thus, we obtain that $\alpha \leq \frac{\widetilde{d_g}(p,o'_k)}{d_g(p,q_k)(1-\epsilon)}$. Since the distance error bound is ϵ, we obtain that $\widetilde{d_g}(p,q_k) \leq (1+\epsilon)d_g(p,q_k)$. Then, we obtain that $\alpha \leq \frac{\widetilde{d_g}(p,q'_k)(1+\epsilon)}{\widetilde{d_g}(p,q_k)(1-\epsilon)}$. By our algorithm, we obtain that $\widetilde{d_g}(p,q'_k) \leq \widetilde{d_g}(p,q_k)$. Thus, we obtain that $\alpha \leq \frac{1+\epsilon}{1-\epsilon} = 1+\frac{2\epsilon}{1-\epsilon} = 1+\epsilon'$. □

Proof of Lemma 4.2. Let $\widetilde{d_g}(p,q)$ denote the distance between p and q returned by the distance query. Let q'_k (resp.,o'_k) denote the point in $KLIST'$ such that $\widetilde{d_g}(p,q'_k) = \min_{q\in KLIST'} \widetilde{d_g}(p,q)$ (resp., $d_g(p,o'_k) = \min_{q\in KLIST'} d_g(p,q)$). Let $KLIST$ denote the list containing the exact kNN of p in P. Let q_k (resp., o_k) denote a point in $KLIST$ such that $\widetilde{d_g}(p,q_k) = \min_{q\in KLIST} \widetilde{d_g}(p,q)$ (resp., $d_g(p,o_k) = \min_{q\in KLIST} d_g(p,q)$). Recall that the approximate ratio α is $\frac{d_g(p,o_k)}{d_g(p,o'_k)}$. Since the distance error bound is ϵ, we obtain that $\widetilde{d_g}(p,o'_k) \leq (1+\epsilon)d_g(p,o'_k)$. Thus, we obtain that $\alpha \leq \frac{d_g(p,o_k)(1+\epsilon)}{\widetilde{d_g}(p,o_k)}$. By the definition of o_k, we obtain that $d_g(p,o_k) \leq d_g(p,q_k)$. Thus, we obtain that $\alpha \leq \frac{d_g(p,q_k)(1+\epsilon)}{\widetilde{d_g}(p,o_k)}$. Since the distance error bound is ϵ, we obtain that $\widetilde{d_g}(p,q_k) \geq (1-\epsilon)d_g(p,q_k)$. Then, we obtain that $\alpha \leq \frac{\widetilde{d_g}(p,q'_k)(1+\epsilon)}{\widetilde{d_g}(p,q_k)(1-\epsilon)}$. By our algorithm, we obtain that $\widetilde{d_g}(p,q'_k) \leq \widetilde{d_g}(p,q_k)$. Thus, we obtain that $\alpha \leq \frac{1+\epsilon}{1-\epsilon} = 1+\frac{2\epsilon}{1-\epsilon} = 1+\epsilon'$. □

Proof of Lemma 4.3. Let $\widetilde{d_g}(p,q)$ denote the distance between p and q returned by the distance query. Let KCP denote a list containing the exact top-k closest pairs and let KCP' denote a list containing the k pairs returned by our algorithm. Let $\langle p_k,q_k\rangle$ (resp., $\langle u_k,v_k\rangle$) denote the a pair in KCP such that $d_g(p_k,q_k) = \max_{\langle p,q\rangle\in KCP} d_g(p,q)$ (resp., $\widetilde{d_g}(u_k,v_k) = \max_{\langle p,q\rangle\in KCP} \widetilde{d_g}(p,q)$). Let $\langle p'_k,q'_k\rangle$ (resp., $\langle u'_k,v'_k\rangle$) denote the pair in KCP' such that $d_g(p'_k,q'_k) = \max_{\langle p,q\rangle\in KCP'} d_g(p,q)$ (resp., $\widetilde{d_g}(u'_k,v'_k) = \max_{\langle p,q\rangle\in KCP} \widetilde{d_g}(p,q)$). Recall that the approximate ratio $\alpha = \frac{d_g(p'_k,q'_k)}{d_g(p_k,q_k)}$. Since the distance error bound is ϵ, we obtain that $\widetilde{d_g}(p'_k,q'_k) \geq d_g(p'_k,q'_k)(1-\epsilon)$. Thus, we obtain that $\alpha \leq \frac{\widetilde{d_g}(p'_k,q'_k)}{d_g(p_k,q_k)(1-\epsilon)}$. By the definition of $\langle p_k,q_k\rangle$, we obtain that $d_g(p_k,q_k) \geq d_g(u_k,v_k)$. Thus, we obtain that $\alpha \leq \frac{\widetilde{d_g}(p'_k,q'_k)}{d_g(u_k,v_k)(1-\epsilon)}$. Since the distance error bound is ϵ, we obtain that $\widetilde{d_g}(u_k,v_k) \leq d_g(u_k,v_k)(1+\epsilon)$. Thus, we obtain that $\alpha \leq \frac{\widetilde{d_g}(p'_k,q'_k)(1+\epsilon)}{\widetilde{d_g}(u_k,v_k)(1-\epsilon)}$. By the definition

Dataset	No. of Vertices	No. of POIs	Region Covered	Resolution
GunnisonForest (GF)	101,583	4,234	10038 km^2	300 meters
LaramieMountain (LM)	101,627	3,896	12400 km^2	300 meters
OkanoganForest (OF)	102,171	5,235	10651 km^2	300 meters
RockyMountain (RM)	101,520	4,098	10828 km^2	300 meters

Table 9. Statistics of Additional Terrain Datasets

Dataset	max	min	avg.	std.
GF	157.46	0.27	36.30	39.77
LM	190.48	3.35	42.44	45.46
OF	208.37	2.10	44.24	47.53
RM	169.39	2.20	38.61	41.52

Table 10. Statistics of Query Distances (km) of Additional Datasets

of $\langle u'_k, v'_k \rangle$, we obtain that $\widetilde{d_g}(u'_k, v'_k) \geq \widetilde{d_g}(p'_k, q'_k)$. Thus, we obtain that $\alpha \leq \frac{\widetilde{d_g}(u'_k,v'_k)(1+\epsilon)}{\widetilde{d_g}(u_k,v_k)(1-\epsilon)}$. By our algorithm, we obtain that $\widetilde{d_g}(u'_k, v'_k) \leq \widetilde{d_g}(u_k, v_k)$. Thus, we obtain that $\alpha \leq \frac{1+\epsilon}{1-\epsilon} = 1 + \frac{2\epsilon}{1-\epsilon} = 1 + \epsilon'$. □

Proof of Lemma 4.4. Let $\widetilde{d_g}(p, q)$ denote the distance between p and q returned by the distance query. Let KCP denote a list containing the exact top-k farthest pairs and let KCP' denote a list containing the k pairs returned by our algorithm. Let KFP denote a list containing the exact top-k farthest pair query. Let $\langle p_k, q_k \rangle$ (resp., $\langle u_k, v_k \rangle$) denote the a pair in KCP such that $d_g(p_k, q_k) = \min_{\langle p,q\rangle \in KCP} d_g(p, q)$ (resp., $\widetilde{d_g}(u_k, v_k) = \min_{\langle p,q\rangle \in KCP} \widetilde{d_g}(p, q)$). Let $\langle p'_k, q'_k \rangle$ (resp., $\langle u'_k, v'_k \rangle$) denote the pair in KCP' such that $d_g(p'_k, q'_k) = \min_{\langle p,q\rangle \in KCP'} d_g(p, q)$ (resp., $\widetilde{d_g}(u'_k, v'_k) = \min_{\langle p,q\rangle \in KCP} \widetilde{d_g}(p, q)$). Recall that the approximate ratio $\alpha = \frac{d_g(p_k,q_k)}{d_g(p'_k,q'_k)}$. Since the distance error bound is ϵ, we obtain that $\widetilde{d_g}(p'_k, q'_k) \leq d_g(p'_k, q'_k)(1 + \epsilon)$. Thus, we obtain that $\alpha \leq \frac{d_g(p_k,q_k)(1+\epsilon)}{\widetilde{d_g}(p'_k,q'_k)}$. By the definition of $\langle p_k, q_k \rangle$, we obtain that $d_g(p_k, q_k) \leq d_g(u_k, v_k)$. Thus, we obtain that $\alpha \leq \frac{d_g(u_k,v_k)(1+\epsilon)}{\widetilde{d_g}(p'_k,q'_k)}$. Since the distance error bound is ϵ, we obtain that $\widetilde{d_g}(u_k, v_k) \geq d_g(u_k, v_k)(1-\epsilon)$. Thus, we obtain that $\alpha \leq \frac{\widetilde{d_g}(u_k,v_k)(1+\epsilon)}{\widetilde{d_g}(p'_k,q'_k)(1-\epsilon)}$. By the definition of $\langle u'_k, v'_k \rangle$, we obtain that $\widetilde{d_g}(u'_k, v'_k) \leq \widetilde{d_g}(p'_k, q'_k)$. Thus, we obtain that $\alpha \leq \frac{\widetilde{d_g}(u_k,v_k)(1+\epsilon)}{\widetilde{d_g}(u'_k,v'_k)(1-\epsilon)}$. By our algorithm, we obtain that $\widetilde{d_g}(u'_k, v'_k) \geq \widetilde{d_g}(u_k, v_k)$. Thus, we obtain that $\alpha \leq \frac{1+\epsilon}{1-\epsilon} = 1 + \frac{2\epsilon}{1-\epsilon} = 1 + \epsilon'$. □

C EXPERIMENTS ON ADDITIONAL DATASETS

Additional Datasets. We also collected four additional real terrain datasets, namely GunnisonForest (in short, GF), LaramieMountain (in short, LM), OkanoganForest (in short, OF), and RockyMountain (in short, RM) and these datasets can be downloaded from https://doi.org/10.5069/G98K778D. For each of these terrain datasets, we extracted a set of POIs from the corresponding region in OpenStreetMap. Table 9 shows the dataset statistics and Table 10 shows the statistics of the query distances on these datasets. As could be

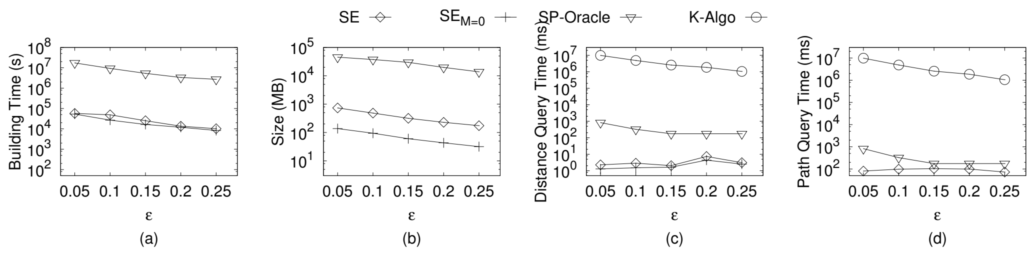

Fig. 45. Effect of ϵ on GF dataset (P2P Distance and Path Queries)

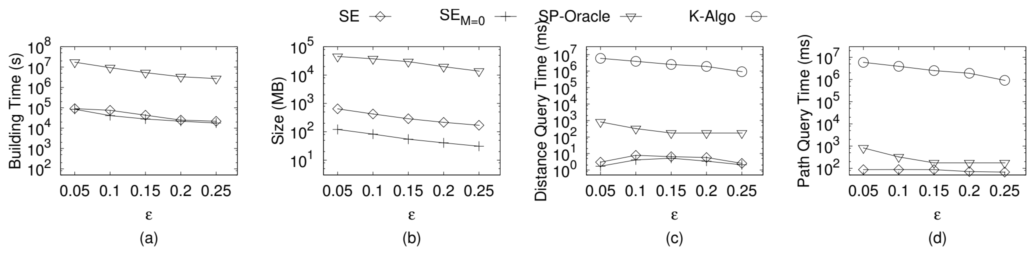

Fig. 46. Effect of ϵ on LM dataset (P2P Distance and Path Queries)

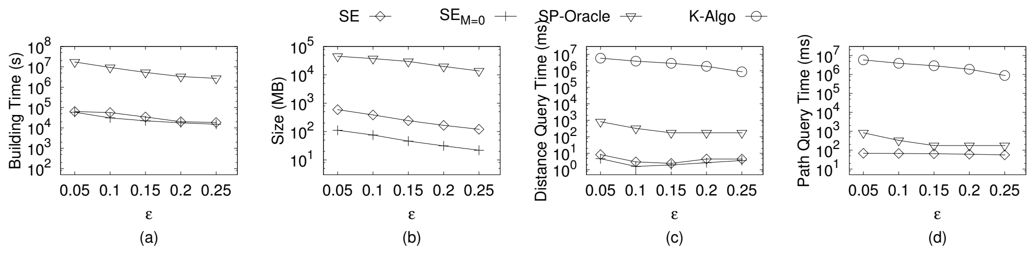

Fig. 47. Effect of ϵ on OF dataset (P2P Distance and Path Queries)

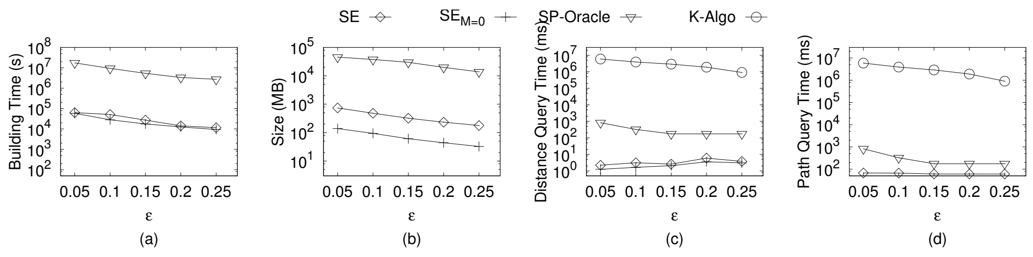

Fig. 48. Effect of ϵ on RM dataset (P2P Distance and Path Queries)

observed from the two tables, the four additional datasets have different properties (i.e., regions covered, query distances) from the BH, EP and SF datasets.

The results on the other four datasets, namely GF, LM, OF and RM, are shown in Figure 45, Figure 46, Figure 47 and Figure 48, respectively. As could be observed from the figures, our oracle *SE* outperforms *SP-Oracle* in terms of building time, oracle size and distance query time by orders of magnitudes. The path query time of *SE* is smaller than that of *SP-Oracle* by several times. *SE* also outperforms *K-Algo* in terms of distance query time and path query time by several orders of magnitudes. Note that due to the different properties of the additional datasets, the curve of each measurement (esp., distance query time) on any additional dataset looks different from that of BH, EP and SF. Specifically, the curves on the additional datasets look smoother than that of BH, EP and SF. This is because the resolution of each additional dataset (i.e., 300 meters) is 10-30 times larger than that of BH, EP and SF. In other words, the resolution of each additional dataset has a lower resolution (or a coarser resolution). In general, a dataset with a coarser resolution means that some detailed up-and-down details could not be represented in this dataset. Instead, the detailed up-and-down details are represented by a flat triangle. Thus, it is more likely that in a dataset with a coarser resolution, each POI (or the center of each node in our partition tree) is not located at the up-and-down detailed terrain (and it is located at the flat triangle). Thus, the results obtained from the POIs chosen from the dataset with a coarser resolution could be stabler or smoother. In conclusion, each additional dataset (with a coarser resolution) is more robust to the factors in our algorithm and more robust to the randomness of the locations of the query points.

D EXPERIMENTS ON EFFECT OF ROOT SELECTION METHODS

We study the effect of the root selection methods by testing P2P queries on the low resolution BH (resolution: 30 meter, 150k vertices) dataset by varying ϵ from $\{0.05, 0.1, 0.15, 0.2, 0.25\}$. The low resolution BH dataset has the same set of POIs and the same region covered as the BH dataset mentioned in Section 6.1. But, the low resolution BH dataset has different resolution and fewer vertices. In the vanilla SE, we randomly select one of the POIs as the center of the root node of partition tree. In this study, we also developed a new algorithm for the selection of the center of the root node which better optimizes the selection by using the information on the x-y plane. In the selection, it first finds the geometric center c of the projections of all POIs on the x-y plane. Then, it finds the POI p whose projection point on x-y plane is closest to c and it finally assigns p to be the center of the root node. Figure 49 demonstrates the results. In this figure, *SE* is our vanilla SE algorithm and *SE-root* is the same as *SE* except that it uses the our newly developed algorithm to select the center of the root node in the partition tree instead of random selection. As could be observed from this figure, *SE* and *SE-root* have neglectable differences which shows that the selection of the center of the root node in the partition tree has a minor effect on the performance of our algorithm. We also provide the analysis of this result from the perspective of theory as follows. Although the selection of the center of the root node in our partition tree determines r_0, in the worse case where a boundary point is selected, r_0 is at most 2 times larger than its optimal value, as a result of which, the height of the partition tree is increased by at most one and the performance only has a neglectable degradation. Thus, we conclude that it is a minor factor in our algorithm.

E EXPERIMENTS ON EFFECT OF RADIUS RATIO OF PARTITION TREE

We study the effect of the radius ratio (denoted by α) of the partition tree by testing P2P queries on the low resolution BH (resolution: 30 meter, 150k vertices) dataset by varying α from $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. ϵ is set to be 0.25 in this study. Note that the radius ratio is the ratio of the radii of two adjacent layers in the partition tree. The low resolution BH

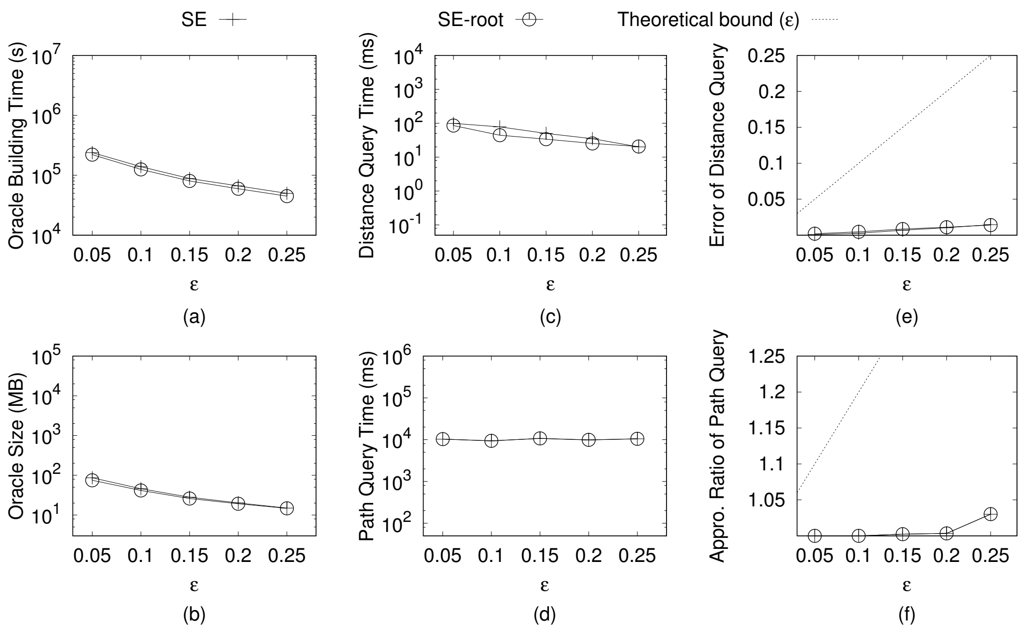

Fig. 49. Effect of Root Selection Methods on BH dataset (P2P Distance and Path Queries)

dataset has the same set of POIs and the same region covered as the BH dataset mentioned in Section 6.1. But, the low resolution BH dataset has different resolution and fewer vertices.

The result is shown in Figure 50. Besides the oracle building time, oracle size, distance query time, error of distance query, path query time and approximate ratio of path query, we also tested the depth of the compressed partition tree and the number of node pairs under different values of α. The results are shown in Figure 50(a) - Figure 50(h), respectively.

As could be observed from this figure, the depth of the compressed partition tree is monotonically increasing with the increase of α (by Figure 50(g)) since (1) the radius of the bottom layer of the partition tree must be smaller than the minimum pairwise distance between all POIs, and (2) for a larger α, it requires more layers to make the radius of the bottom layer smaller than the minimum pairwise distance between all POIs. As such, the oracle building time and the distance query time are also monotonically increasing with the increase of α by Figure 50(a) and Figure 50(b). Secondly, the number of node pairs in *SE* is monotonically decreasing with the increase of α (by Figure 50(h)). This is because a smaller α results in fewer layers in the compressed partition tree, as a result of which, the oracle size is monotonically decreasing with the increase of α by Figure 50(d). Besides, the error of the distance query is also monotonically increasing by Figure 50(c) since the fewer node pairs that SE has, the fewer distances pre-computed for the query processing.

We also normalized the y values tested in each figure from Figure 50(a) to Figure 50(h) by using the method of Min-max feature scaling (i.e., the normalized value y' of y is calculated as $\frac{y-y_{min}}{y_{max}-y_{min}}$, where y_{max} (resp. y_{min}) is the maximum (resp. minimum) value of y). Then, at each value of α we tested, we average all normalized y values in the Figure 50(a) - Figure 50(h) at the same α to obtain the average normalized score under that α. Figure 50(i) reports the result of the average normalized score of all values for α, in which the average normalized score achieves its minimum at 0.5. Thus, we conclude that *SE* has the best overall performance when α is equal to 0.5.

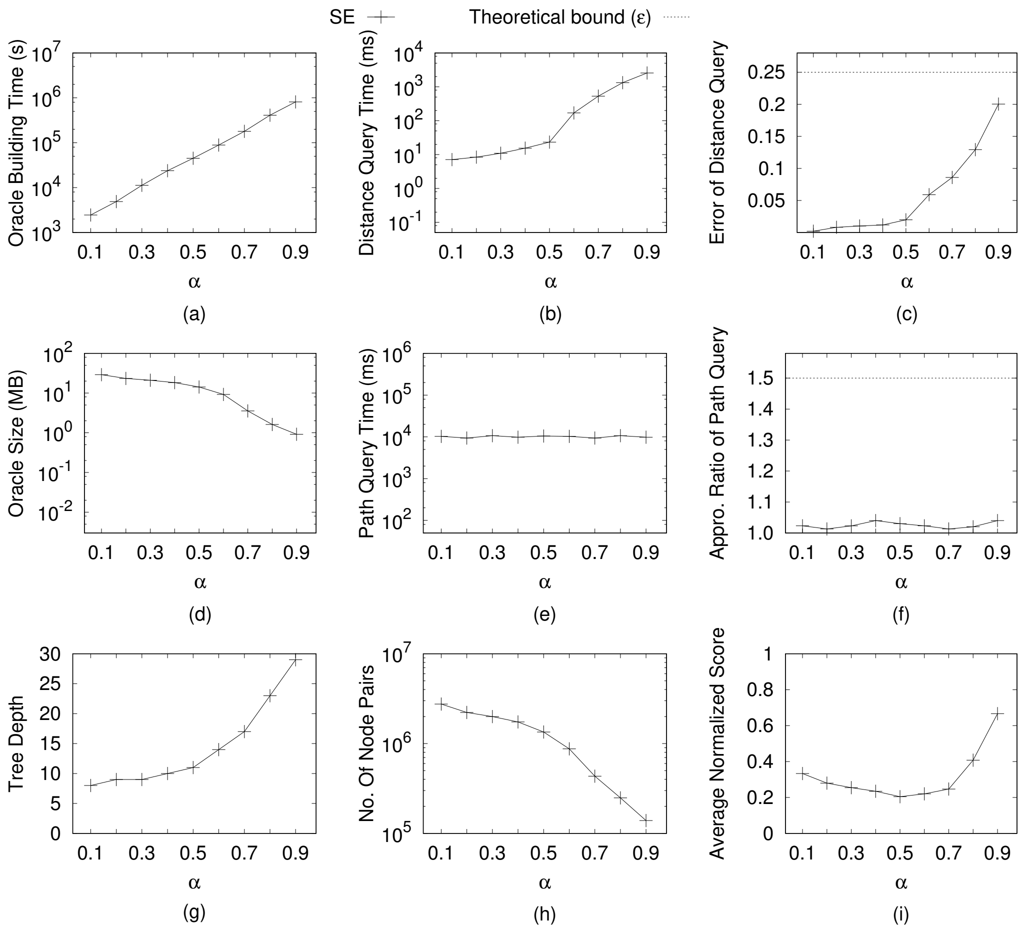

Fig. 50. Effect of Radius Ratio on BH dataset (P2P Distance and Path Queries)

F A2A DISTANCE QUERY PROCESSING

We present an oracle to answer the arbitrary point-to-arbitrary point (A2A) distance query based on our proposed distance oracle *SE*. This oracle is the same as that presented in Section 3 except that it takes some Steiner points introduced as input instead of all POIs, where Steiner points are introduced by the method proposed in [15] (there are $O(\frac{N}{\sin(\theta)\cdot\sqrt{\epsilon}}\log\frac{1}{\epsilon})$ Steiner points, where θ is the minimum inner angle of any face). Then, we present the query processing. Given two arbitrary points s and t, we first find the neighborhood of s (resp., t), denoted by $\mathcal{N}(s)$ (resp., $\mathcal{N}(t)$) (It is a set of Steiner points on the same face containing s (resp., t) and its adjacent face(s) [15]. Finally, we return

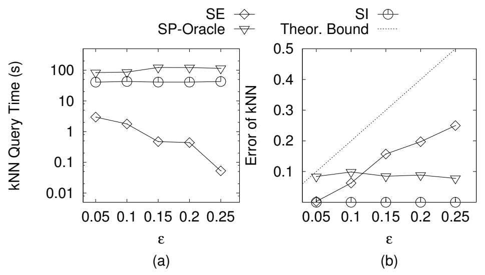

Fig. 51. Effect of ϵ on real dataset, BearHead, for monotonic kFN

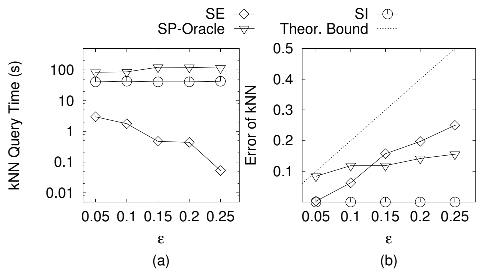

Fig. 52. Effect of ϵ on real dataset, EaglePeak, for monotonic kFN

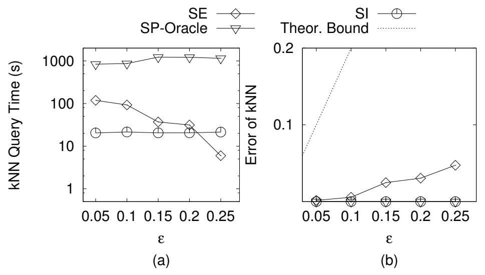

Fig. 53. Effect of ϵ on real dataset, San Francisco, for monotonic kFN

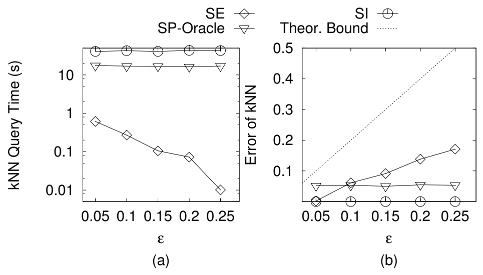

Fig. 54. Effect of ϵ on real dataset, BearHead, for bichromatic kFN

$\widetilde{d_g}(s,t) = \min_{p\in\mathcal{N}(s),q\in\mathcal{N}(t)}[d_g(s,p) + \widetilde{d_g}(p,q) + d_g(q,t)]$, where $d_g(s,p)$ and $d_g(q,t)$ could be computed in constant time by SSAD algorithm and $\widetilde{d_g}(p,q)$ is the distance between p and q estimated by the oracle constructed. By [15], $|\mathcal{N}(s)| \cdot |\mathcal{N}(t)| = \frac{1}{\sin(\theta)\cdot\epsilon}$ and if $\widetilde{d_g}(p,q)$ is an ϵ-approximate distance of $d_g(p,q)$, then $\widetilde{d_g}(s,t)$ is also an ϵ-approximate distance of $d_g(s,t)$. By Theorem 3.12, we obtain that for any two Steiner points p and q, $\widetilde{d_g}(p,q)$ is an ϵ-approximate distance of $d_g(p,q)$ and it takes $O(h)$ time to compute $\widetilde{d_g}(p,q)$ and the building time (resp., oracle size) is $O(\frac{N\log^2 N}{\epsilon^{2\beta}} + \frac{Nh}{\sin(\theta)\sqrt{\epsilon}}\cdot\log\frac{1}{\epsilon}\cdot\log\frac{N\log\frac{1}{\epsilon}}{\sin(\theta)\sqrt{\epsilon}} + \frac{Nh}{\sin(\theta)\sqrt{\epsilon}\cdot\epsilon^{2\beta}}\cdot\log\frac{1}{\epsilon})$ (resp., $O(\frac{Nh}{\sin(\theta)\sqrt{\epsilon}\cdot\epsilon^{2\beta}}\cdot\log\frac{1}{\epsilon})$). Thus, we obtain that for any two arbitrary points s and t, the oracle gives an ϵ-approximate distance of $d_g(s,t)$ and the query time is $O(\frac{h}{\sin(\theta)\cdot\epsilon})$.

G EXPERIMENT ON K FARTHEST NEIGHBOR AND TOP-K FARTHEST PAIR QUERIES

G.1 Monochromatic and Bichromatic k Farthest Neighbor Query

In this section, we report the experimental results of the monotonic and bichromatic k farthest neighbor (FN) queries.

Effect of ϵ. We tested 5 different values of ϵ (i.e., $0.05, 0.1, 0.15, 0.2, 0.25$). The results on the two types of queries in the datasets, BearHead, EaglePeak and San Francisco South are shown in Figure 51 - Figure 56. As the figures show, the kFN query time of *SE* is smaller than that of SP-Oracle and SI by several orders of magnitude in BH and EP. Although SE

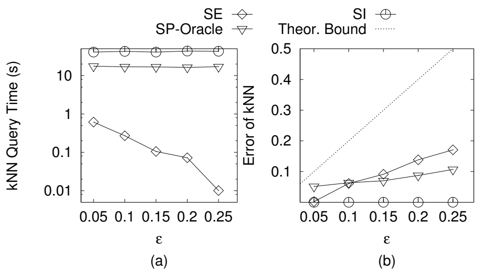

Fig. 55. Effect of ϵ on real dataset, EaglePeak, for bichromatic kFN

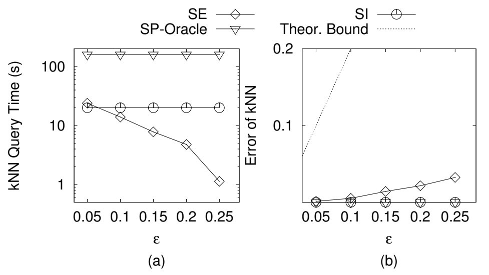

Fig. 56. Effect of ϵ on real dataset, San Francisco, for bichromatic kFN

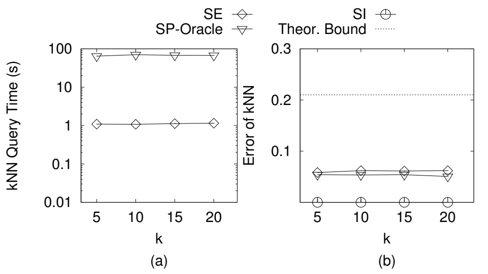

Fig. 57. Effect of k on real dataset, BearHead, for monotonic kFN

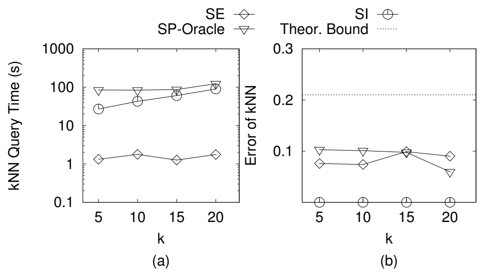

Fig. 58. Effect of k on real dataset, EaglePeak, for monotonic kFN

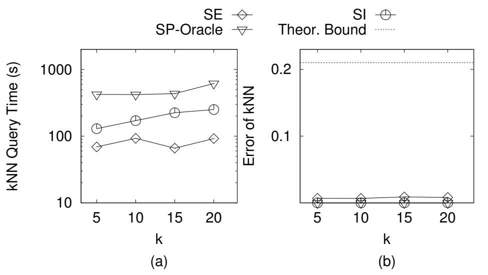

Fig. 59. Effect of k on real dataset, San Francisco South, for monotonic kFN

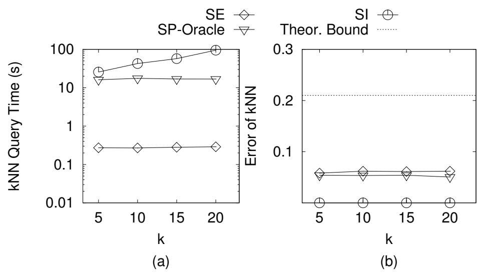

Fig. 60. Effect of k on real dataset, BearHead, for bichromatic kFN

could not consistently outperform baseline algorithms in San Fransisco dataset, SE is still the fastest one in most cases in Figure 53 and Figure 56. The appro. errors of *SE*, SI and SP-Oracle are very similar and are much smaller than the theoretical bound.

Effect of k. We tested 4 different values of k, namely $5, 10, 15, 20$. The results on the two types of queries in the datasets, BearHead, EaglePeak and San Francisco South are shown in Figure 57 - Figure 62. As the figures show, the kNN query time of *SE* is smaller than that of SP-Oracle and SI by several orders of magnitude in BH and EP. Although the speedup of SE compared with the baselines is not that significant in San Fransisco dataset, SE is still the

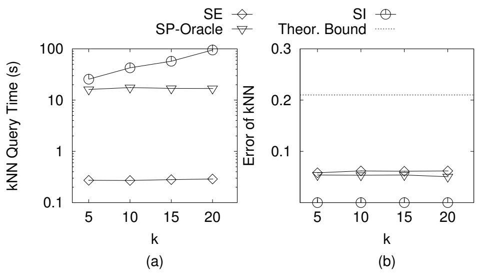

Fig. 61. Effect of k on real dataset, EaglePeak, for bichromatic kFN

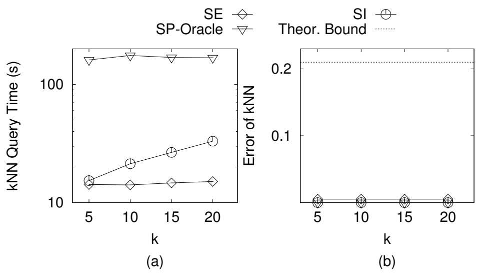

Fig. 62. Effect of k on real dataset, San Francisco South, for bichromatic kFN

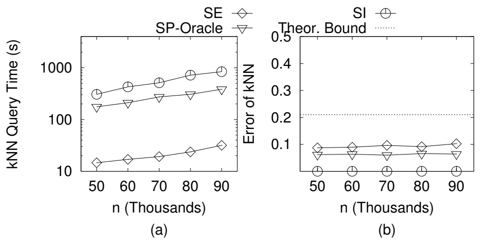

Fig. 63. Effect of n on dataset, BearHead, for bichromatic kFN

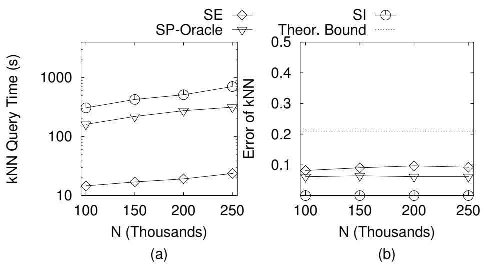

Fig. 64. Effect of N on dataset, BearHead, for bichromatic kFN

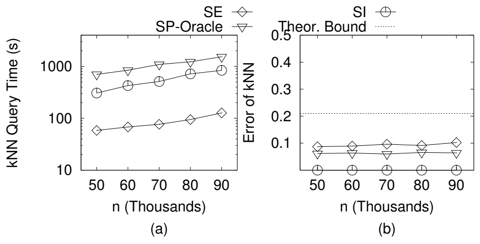

Fig. 65. Effect of n on dataset, BearHead, for monotonic kFN

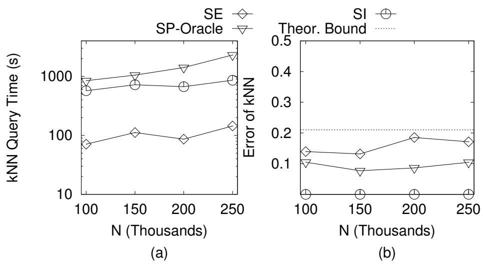

Fig. 66. Effect of N on dataset, BearHead, for monotonic kFN

fastest one. The appro. error of *SE*, SI and SP-Oracle are very small in practice and much smaller than the theoretical bound.

Effect of n. We studied the effect of the number of POIs n on a synthetic dataset. The values of n we tested are $50k, 60k, 70k, 80k, 90k$. We generate the synthetic POIs in the same way as stated in Section 6.2.1. The results are shown in Figure 63 and Figure 65. As figures shows, the kFN query time of either *SE* is smaller than that of SP-Oracle and SI by several times. The appro. error of *SE*, SI and SP-Oracle are very small in practice and much smaller than the theoretical bound.

Effect of N. We studied the effect of the number of vertices N on the terrain surface on a synthetic dataset. The values of N we tested are $100k, 150k, 200k, 250k$. We generated the terrain data in the same way as stated in Section 6.2.1. The results are shown in Figure 64 and Figure 66. As figures shows, the kFN query time of SE is smaller than that of SP-Oracle and SI by 0.5-3 orders of magnitude. The appro. error of SE is larger than that of SP-Oracle but is much smaller than the theoretical bound.

G.2 Top-k Bichromatic and Monotonic Farthest-Pair Query

We present the experimental results of the top-k bichromatic farthest-pair (BFP) and monotonic farthest-pair (MFP) queries.

Effect of ϵ. We tested 5 different values of ϵ (i.e., $0.05, 0.1, 0.15, 0.2, 0.25$). The results of the top-k BFP query on BearHead, EaglePeak and San Fransisco are shown in Figure 67(a), Figure 68(a) and Figure 69(a). The results of the top-k MFP query on BearHead, EaglePeak and San Fransisco are shown in Figure 71(a), Figure 72(a) and Figure 73(a). The query time of SE for both types of queries is very small in practice. The top-k BFP query and the top-k MFP query processing of SP-Oracle could not be finished within a reasonable time and thus it is not shown in the figure. As the figures show, the query time of SE is smaller than that of baselines by 1-3 orders of magnitude. Since there is no exact algorithm for the closest pair query, we are not able to calculate the appro. error.

Effect of k. We tested 4 different values of k, namely $5, 10, 15, 20$. The results of the top-k BFP query on BearHead, EaglePeak and San Fransisco are shown in Figure 67(b), Figure 68(b) and Figure 69(b). The results of the top-k MFP query on BearHead, EaglePeak and San Fransisco are shown in Figure 71(b), Figure 72(b) and Figure 73(b). The query time of SE for both types of queries is small in practice. The top-k BFP query and the top-k MFP processing of SP-Oracle could not be finished within a reasonable time and thus it is not shown in the figure.

Effect of n. We studied the effect of the number of POIs n on a synthetic dataset. The values of n we tested are $50k, 60k, 70k, 80k, 90k$. The data is generated in the same way as stated in Section 6.2.1.

The results of the top-k BFP and the top-k MFP query are shown in Figure 70(a) and Figure 74(a). As Figure 70(a) shows, the query time of SE for the top-k BFP query is small in practice. The top-k BFP query processing of SP-Oracle could not be finished within a reasonable time and thus it is not shown in the figure. We observe similar result for the top-k MFP query.

Effect of N. We studied the effect of the number of vertices N on the terrain surface on a synthetic dataset. The values of N we tested are $100k, 150k, 200k, 250k$. We generated the terrain data as stated in Section 6.2.1. The results of the top-k BFP and the top-k MFP query are shown in Figure 70(b) and Figure 74(b). As Figure 70(b) shows, the query time of SE for the top-k BFP query is small in practice. The top-k BFP query processing of SP-Oracle could not be finished within a reasonable time and thus it is not shown in the figure. We observe similar result for the top-k MFP query.

H DISCUSSION FOR CASE WHEN $n > N$

When $n > N$, we adopt the same distance oracle described in Appendix F, which is POI-independent. This distance oracle could answer not only A2A distance queries but also V2V distance queries and P2P distance queries (because A2A distance queries could be regarded as a general setting compared with V2V distance queries and P2P distance queries).

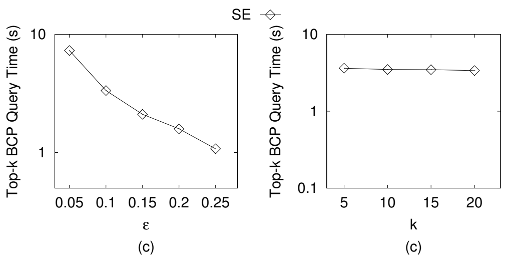

Fig. 67. **Effect of ϵ and k on real dataset, BearHead, for Top-k BFP**

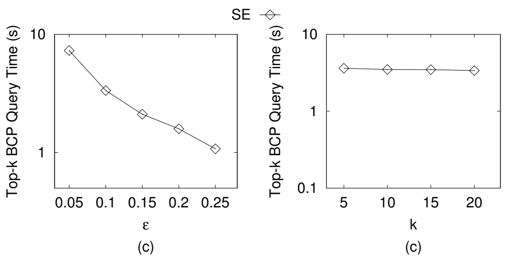

Fig. 68. **Effect of ϵ and k on real dataset, EaglePeak, for Top-k BFP**

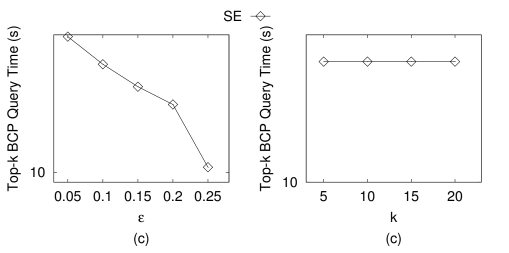

Fig. 69. **Effect of ϵ and k on real dataset, San Fransisco, for Top-k BFP Query**

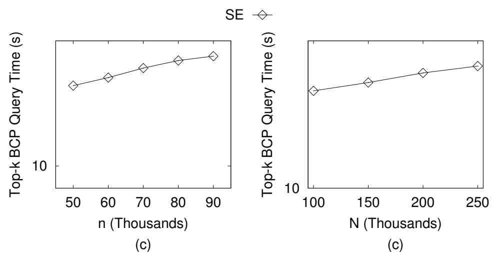

Fig. 70. **Effect of n and N on dataset, San Francisco South, for Top-k BFP Query**

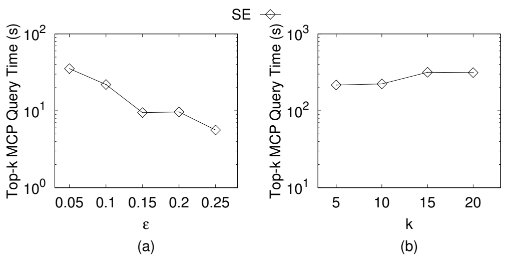

Fig. 71. **Effect of ϵ and k on real dataset, BearHead, for Top-k MFP Query**

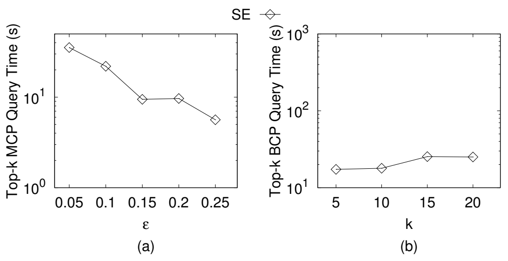

Fig. 72. **Effect of ϵ and k on real dataset, EaglePeak, for Top-k MFP Query**

I EXTENSION TO WEIGHTED TERRAIN SURFACE

Weighted terrain surface is a variant of the terrain surface studied in the paper. The same as the terrain surface which is studied in this paper, a weighted terrain surface also has a set of vertices, a set of edges and a set of faces. Each edge is adjacent to two vertices and each face has three adjacent vertices and three adjacent edges. Each vertex $v \in V$ has three coordinate values, denoted by x_v, y_v and z_v. But differently, in the weighted terrain surface, each face is assigned a positive real-valued weight and the weight information captures different travel cost (e.g., travel effort and energy consumption, etc.) in different faces and

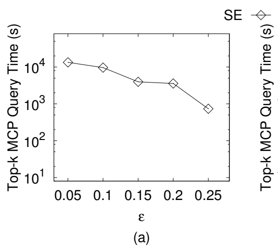

Fig. 73. Effect of ϵ and k on real dataset, San Fransisco, for Top-k MFP Query

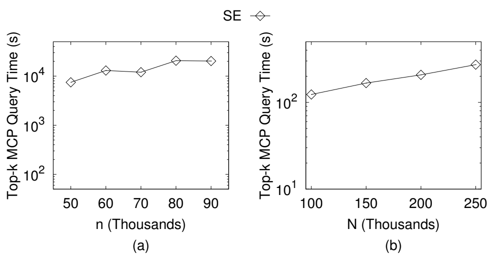

Fig. 74. Effect of n and N on dataset, San Francisco South, for Top-k MFP Query

the weight information encodes many different features such as slope, terrain type (e.g., sand, wet land), obstacles, etc. [3, 15, 29]. Consider two points s, t and let $\pi_g(s,t)$ denote a path between them on a weighted terrain surface. Formally, $\pi_g(s,t)$ consists of a sequence X of line segments and each segment lies on a unique face of the terrain surface. Given a line segment $x \in X$, we denote the unique face that x lies on by f_x and we denote the length of x by $l(x)$. The length of $\pi_g(s,t)$ on the weighted terrain surface, denoted by $l(\pi_{wg}(s,t))$, is defined to be $\sum_{x \in X}(w(f_x) \cdot l(x))$ which is the sum over the product of the length of each line segment in X and the weight of the face it lies on. The *geodesic shortest path on the weighted terrain surface* between s and t, denoted by $\Pi_{wg}(s,t)$, is defined to be the path between s and t on the terrain surface with the smallest length (i.e., $\Pi_{wg}(s,t) = \arg\min_{\pi_g(s,t)}\{l(\pi_g(s,t))\}$). The *weighted geodesic distance* between s and t, denoted by $d_{wg}(s,t)$, is defined to be the length of $\Pi_{wg}(s,t)$.

Although our technique is originally designed for the unweighted terrain, it paves the way to the research on weighted terrain surface and our technique could be easily adapted to the weighted terrain surface with some minor modifications. The adaptation is demonstrated as follows. In a word, the adaptation simply replaces the geodesic distance and path computation involved in the original *SE* by the corresponding weighted geodesic distance computation and the computation of the shortest path on weighted terrain surface in the preprocessing phase. As a result of that, *SE* would pre-compute and store several weighted geodesic distances in place of the original geodesic distances and the operations in the query phase keeps intact. Specifically, there are three subroutines in the pre-computation of the original*SE* which requires the geodesic distance or path computation and they are disk computation, enhance edge computation and node pair generation. In the implementation level, they all invoke the single-source all-destination (SSAD) algorithm for this geodesic distance and path computation. Thus, we replace the SSAD algorithm for the geodesic distance and path computation by the corresponding SSAD algorithm for the weighted geodesic distance and geodesic path computation on weighted terrain surfaces [4]. We also conducted empirical study on this weighted terrain surface. We used the low resolution BH (resolution: 30 meter, 150k vertices) for this experiment. Note that the dataset only provides the information of vertices, edges and faces and does not contain the weight information of each face. For the experiment on the weighted terrain surface, we adopted the method in [42] to generate the weight for each face. The result is shown in Table 11 and the error parameter ϵ is set to be the default value (i.e., 0.2). We compared *SE* with [4] (which is the best existing on-the-fly algorithm for the weighted geodesic distance computation) and

SP-Oracle [15] (which is the best existing index-based algorithm for the weighted geodesic distance computation) in this experiment. Note that *SP-Oracle* could also be adapted to weighted terrain surfaces [15]. As could be observed from this table, *SE* significantly outperforms *SP-Oracle* in terms of building time, space consumption and query time. The query time of *SE* is several orders of magnitude smaller than that of [4]. The three algorithms tested all have neglectable error which is much smaller than the error parameter.

Algorithm	Building Time (s)	Oracle Size (MB)	Distance Query Time (ms)	Path Query Time (ms)	Error
[4]	-	-	1,982,097	1,997,234	0.001
SP-Oracle [15]	598,231	89,337	5,387	5,674	0.002
SE	42,031	5,345	29	146	0.03

Table 11. Performances of All Algorithms on Weighted Terrain Surface (BH, low resolution)