Efficient Proximity Queries on Simplified Height Maps

Anonymous
Anonymous
Anonymous

Anonymous
Anonymous
Anonymous

ABSTRACT

Performing proximity queries on a 3D surface has gained significant attention from both academic and industry, where the height map is one fundamental 3D surface representation with many advantages over others such as the point cloud and *Triangular-Irregular Network* (*TIN*). In this paper, we study the shortest path query on a height map. Since performing proximity queries using the shortest path on a height map is costly, we propose a simplification algorithm on the height map to accelerate it. We also propose a shortest path query algorithm and algorithms for answering proximity queries on the original/simplified height map. Our experiments show that our simplification algorithm is up to 21 times and 5 times (resp. 412 times and 7 times) better than the best-known adapted point cloud (resp. *TIN*) simplification algorithm in terms of the simplification time and output size (the size of the simplified surface), respectively. Performing proximity queries on our simplified height map is up to 5 times and 1,340 times quicker than on the simplified point cloud and the simplified *TIN* with an error at most 10%, respectively.

ACM Reference Format:
Anonymous and Anonymous. 2025. Efficient Proximity Queries on Simplified Height Maps. In *Proceedings of 2026 International Conference on Management of Data (SIGMOD '26).* ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Performing proximity queries on a 3D surface has gained significant attention from both academic and industry [62, 69]. Academic researchers studied different types of proximity queries [30, 31, 49, 56, 62, 65, 66, 69], including *shortest path queries* [27, 42, 43, 47, 48, 51, 52, 61–64, 67–70], *k-Nearest Neighbor* (*kNN*) *queries* [30, 31, 56, 62, 65] and *range queries* [49, 58]. In industry, Google Earth [9] and Metaverse [15] employ shortest paths passing on 3D surfaces (e.g., Earth and virtual reality) for user navigation.

Height map, point cloud and *TIN*: There are different representations of a 3D surface, including *height map*, *point cloud* [69] and *Triangular-Irregular Network* (*TIN*) [62, 65, 66]. Figure 1 (a) shows a 3D surface in a 20km × 20km region in Gates of the Arctic [54] national park, USA. Figure 1 (b) shows the height map representation of this surface. Consider a 2D plane with 9 × 9 grid cells in this region. Each cell has *2D coordinate values* representing 2D coordinate values of its center point, and a *grayscale pixel color* representing its *elevation value* (e.g., calculated using a simple linear interpolation based on the pixel color), meaning the height projected from this center point on the 3D surface. If this value is larger, this pixel's color is brighter. Besides, each cell has 8 neighbors (shown in blue points of Figure 1 (b)). All these cells form a height map. Figure 1 (c) shows this height map in bird's eye view. Figure 1 (d) shows the point cloud representation of this surface. Each cell in the height map could be *one-to-one* mapped to a 3D point where the x- and y- coordinate values of this point are the 2D coordinate values of the center point of this cell, and the z-coordinate of this point is the elevation value of this cell [25, 45, 60, 72] (this is the best-known exact height map to point cloud conversion algorithm). This runs in $O(n)$ times, where n is the number of cells in the height map. The best-known approximate conversion algorithm uses machine learning approach, e.g., uniform random sampling [37, 46] for acceleration, but the converted point cloud is an approximated representation of the height map, since it randomly selects some (not all) cells for mapping. This runs in $O(n_r)$ times, where n_r is the number of randomly selected cells. Figure 1 (e) shows the *TIN* representation of this surface. A *TIN* has a set of contiguous triangulated *faces*, where each face has three *edges* connecting at three *vertices*. In practice, the *TIN* is converted from the point cloud [69] via *triangulation* [35, 59, 69] where all vertices of faces are the points in the point cloud. This runs in $O(n)$ times. If the triangulation is applied on the approximated point cloud, this runs in $O(n_r)$ times.

1.1 Advantages of Height Map

Height maps offer several advantages over point clouds and *TIN*s.

(1) Compared with point cloud datasets, there are more height map datasets available (e.g., there are 50M height map datasets but only 20M point cloud datasets in an open data 3D surface dataset platform called OpenDEM [16]), with four reasons.

(i) *Longer history of the height map.* The height map and point cloud were introduced in 1884 [2] and 1960 [12], respectively. So, more height map datasets are available due to the earlier adoption.

(ii) *Lower cost of obtaining a height map dataset.* The height map dataset could be obtained from either *optical* images of cost USD $25 [20] captured by an *optical* satellite or *radar* images of cost USD $3,300 [8] captured by a *radar* satellite. But, the point cloud dataset could be obtained only from *radar* images captured by a *radar* satellite (if no conversion operation from the height map to the point cloud is involved). The image cost difference is probably due to the satellite launching cost difference: USD $0.4 billion [13] for an optical satellite and USD $1.5 billion [1] for a radar satellite.

(iii) *More region coverage of the height map datasets.* Since optical and radar satellites cover 100% [3] and 80% [19] of Earth's land area, respectively, height map datasets cover more regions compared with point cloud datasets. For example, high-latitude regions (e.g., Gates of the Arctic national park in the northern part of Alaska, USA) are regions covered by height map datasets but not point

SIGMOD '26, May 31 – June 5, 2026, Bengaluru India

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

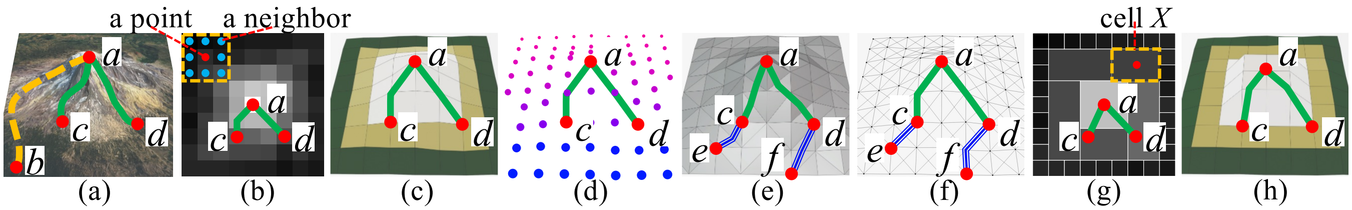


Figure 1: Paths passing on (a) a 3D surface, (b) a height map, (c) a height map in bird's eye view, (d) a point cloud, (e) a *TIN*, (f) a height map graph, (g) a simplified height map and (h) a simplified height map in bird's eye view

cloud datasets [21]. We perform a snowfall evacuation case study there, using the only available height map datasets for evacuation.

(iv) *Large conversion time from height map datasets to point cloud datasets.* In our experiment, converting height map datasets to point cloud datasets [25, 45, 60, 72] for radar satellites' uncovered region takes 21 years[1]. Although it is fast (e.g., 42s for a region of 1km^2) for a small area and we need it once, since we capture the height map *after* snowfall due to avalanches in our evacuation case study (i.e., the height map dataset is updated), and the weather may change suddenly in 1s [4] (complicating rescue efforts), we aim to avoid this to minimize sudden weather changes and save more lives.

(2) Compared with *TIN* datasets (i.e., usually converted from point cloud datasets), height map datasets are easier to access since satellites can capture them directly, so more height map datasets are available, and the 4 reasons above also apply to *TIN*s. Height maps also use less hard disk space, since they store cell information, while *TIN*s store vertex, edge and face information. Our experiments show that a height map with 25M cells and a *TIN* (converted from the height map) require 600MB and 13.6GB, respectively.

1.2 Our Focus

1.2.1 ***Height map shortest path query.*** In this paper, we study the shortest path query on the height map. There are two issues.

(1) *How to find the shortest path passing on the height map.* To the best of our knowledge, there is no existing study finding the shortest path *directly* on a height map. Most (if not all) algorithms [45, 53, 60, 72] adapt shortest path algorithms on the point cloud or *TIN* by converting the height map to point cloud and *TIN*, and then perform the shortest path query on the converted point cloud [69] or *TIN* [27, 42–44, 48, 63, 64, 68, 70]. In this paper, we propose a *height map graph* in Figure 1 (f) (constructed from the height map). Intuitively, for each cell in the height map, we construct a corresponding 3D vertex in the graph. For each pair of neighboring cells, we create an edge between their corresponding vertices with a weight equal to the Euclidean distance between them. Based on this graph, we could find the shortest path by Dijkstra's algorithm [32]. Our experiments show that computing the shortest path passing on a height map with 0.5M cells needs 3s, but computing the shortest path passing on a point cloud (see Figure 1 (d)) converted from this height map [69] needs 3.4s due to data conversion. Besides, computing the shortest surface path passing on a *TIN* (see Figure 1 (e)) converted from this height map [27, 63, 70] needs 280s $\approx$ 4.6 min, since the height map's structure is simpler.

(2) *How to improve it in other proximity queries.* In our experiments, using shortest paths to answer *kNN* or range queries for 10k possible query objects on a height map with 50k cells both need 4,400s $\approx$ 1.2 hours, which is quite long. Thus, we propose a *simplification* process on the height map.

1.2.2 ***Height map simplification.*** In this paper, we also study how to simplify the height map. If we merge *nearby* cells with similar elevation values (considered as *redundant* information) into one cell, then the number of cells is reduced and Dijkstra's algorithm on this simplified height map is faster. Figure 1 (g) shows a simplified height map of the same surface, where cell X is merged from 6 cells (whose elevation value is the average of these 6 cells). Figure 1 (h) shows this simplified height map in bird's eye view. Consider a pair of points a and c. There is a relative error called the *distance error ratio* of the distance calculated by a studied algorithm compared with the ground-truth or optimal distance, i.e., the *approximate* shortest distance between a and c on the simplified height map in Figure 1 (g) compared with the (*exact*) shortest distance between a and c on the original height map in Figure 1 (b).

Given an error parameter $\epsilon \in [0, 1]$, we study how to simplify the height map so that the distance error ratio for any pairs of points on the original height map is at most ϵ. There are two challenges.

(1) *How to simplify the height map with a small size efficiently.* To the best of our knowledge, there is no existing study focusing on simplifying a height map. The only closely related work are the simplification algorithms on the point cloud [23, 69] or *TIN* [31, 39, 43, 47]. We adapt them by converting the height map to point cloud and *TIN*, and then performing the original simplification algorithms on the converted point cloud and *TIN*. But, the size of the simplified point cloud and *TIN* are large since they do not consider any optimization techniques, resulting in large shortest path query times on their simplified point cloud and *TIN*. The simplification time of the point cloud (resp. *TIN*) simplification algorithms are large since they are randomized algorithms without any pruning technique (resp. involve expensive *TIN re-triangulation* [35, 59, 69]). We need to avoid this when simplifying the height map.

(2) *How to define the neighborhoods of cells in the simplified height map.* In the original height map (Figure 1 (b)), it is clear to understand the neighborhoods of each cell. But, in the simplified height map (Figure 1 (g)), since each *merged* cell can be adjacent to many

[1] Since the total Earth's land area is 149M km^2 [7], the total areas covered by optical satellites but not radar satellites are 30M km^2 (i.e., 149M km^2 $\times$ (100% - 80%)). In our experiment, converting a height map dataset in a region of 1 km^2 (with 3m $\times$ 3m resolution) to a point cloud dataset takes 42s. Thus, the conversion time is 42s/km^2 $\times$ 30M km^2 = 1.26 $\times 10^9$s $\approx$ 21 years.

different cells, we need to define clearly neighborhoods of each (merged/non-merged) cell for the shortest path query.

1.3 Contribution and Organization

We summarize our contributions as follows.

(1) We are the first to study the shortest path query directly on the height map. We also adopt a height map simplification process so that the distance error ratio for any pair of points on the original height map is at most ϵ. We show that this process is *NP-hard*.

(2) We propose an ϵ-approximate height map simplification algorithm called *Height Map Simplification Algorithm* (*HM-Simplify*), which can (i) significantly reduce the number of cells of the simplified height map, i.e., reduce the output size (the size of the simplified height map), to further reduce the shortest path query time on the simplified height map using a novel cell merging technique (by using cell information of height maps), and (ii) efficiently reduce the simplification time using an efficient checking technique during simplification (by using neighbor information of height maps). It clearly defines neighborhoods of cells in the simplified height map. We also propose a shortest path query algorithm called *Height Map Shortest Path Query Algorithm* (*HM-SP*) on the original/simplified height map, which can efficiently reduce the shortest path query time on the simplified height map using an efficient implicit edge insertion technique (by using neighbor information of height maps and the single-source-all-destination feature of Dijkstra's algorithm). We also design algorithms for answering *kNN* and range queries on the original/simplified height map, which can also efficiently reduce the proximity query time on the original/simplified height map using an efficient parallel computation technique (by using the single-source-all-destination feature of Dijkstra's algorithm). Our experimental ablation studies show these improvements.

(3) We give theoretical analysis on algorithm *HM-Simplify*'s simplification time, number of cells in the simplified height map, output size and error guarantee, algorithm *HM-SP*'s shortest path query time, memory usage and error guarantee, and proximity query algorithms' query time and error guarantee.

(4) Algorithm *HM-Simplify* outperforms the best-known adapted point cloud [23, 69] and *TIN* [40, 43] simplification algorithm concerning the simplification time and output size. Performing proximity queries on the simplified height map is much quicker than the best-known algorithms [27, 63, 69, 70] on the simplified point cloud and the simplified *TIN*. Our experiments show that given a height map with 50k cells, the simplification time and output size are 250s ≈ 4.6 min and 0.07MB for algorithm *HM-Simplify*, but are 5,250s ≈ 1.5 hours and 0.35MB for the best-known adapted point cloud simplification algorithm [23, 69], and 103,000s ≈ 1.2 days and 0.5MB for the best-known adapted *TIN* simplification algorithm [40, 43]. The proximity query time of 10k objects is 50s on the simplified height map, 250s ≈ 4.2 min on the simplified point cloud and 67,000s ≈ 18.6 hours on the simplified *TIN*.

The remainder of the paper is organized as follows. Section 2 gives the problem definition. Section 3 covers the related work. Section 4 presents our algorithms. Section 5 discusses the experimental results and Section 6 concludes the paper.

2 PROBLEM DEFINITION

2.1 Notation and Definitions

2.1.1 ***Height map***. Consider a height map $H = (C, N(\cdot))$ on a 2D plane containing a set of *cells* C with size n, and a *neighbor cells* (*hash*) table [29] $N(\cdot)$. In H, each cell $c \in C$ has 2D coordinate values (representing 2D coordinate values of its center point) and a grayscale pixel color (representing its *elevation value*), denoted as $c.x$, $c.y$ and $c.z$, respectively. Given cell $c \in C$, $N(c)$ returns c's *neighbor cells* in $O(1)$ time, and it is initialized to be c's nearest 8 surrounding cells on H. Figure 2 (a) shows a height map with 9 cells. For point p on cell c, 6 orange and 2 red points form $N(c)$.

We define the *height map graph* of H, says G, as follows. Let $G.V$ and $G.E$ be G's vertices and edges, respectively. For each cell $c \in C$, we create a vertex v_c whose x-, y- and z-coordinate values are defined to $c.x$, $c.y$ and $c.z$ of c, respectively. All vertices created form $G.V$. For each cell $c \in C$ and each cell $c' \in N(c)$, we create an edge between vertex v_c and vertex $v_{c'}$ (corresponding to c and c') with a weight equal to the Euclidean distance between v_c and $v_{c'}$. All edges created form $G.E$. c and c' are said to be *adjacent* if there is an edge between v_c and $v_{c'}$ in G. The graphs in Figures 2 (a) and (b) are the height map graph of H on the 2D plane and in a 3D space, respectively. Given a pair of points s and t on H, let $\Pi(s, t|H)$ be the (*exact*) *shortest path* between them passing on (G of) H. Let $|\cdot|$ be a path's distance (e.g., $\Pi(s, t|H)$'s distance is denoted by $|\Pi(s, t|H)|$). Figures 2 (a) and (b) show $\Pi(s, t|H)$ in green line.

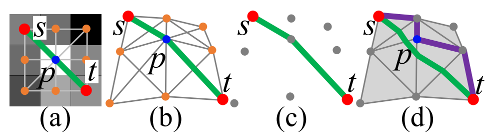


Figure 2: Paths passing on (a) a height map, (b) a height map graph, (c) a point cloud and (d) a *TIN*

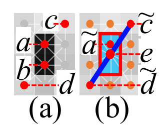


Figure 3: Cell merging

2.1.2 ***Simplified height map***. Given a height map H, we can obtain a simplified height map $\widetilde{H} = (\widetilde{C}, \widetilde{N}(\cdot))$ by merging some adjacent cells (deleting these cells and adding a new *larger* cell covering these cells for replacement) in H, where $\widetilde{C}$ and $\widetilde{N}(\cdot)$ are initialized as C and $N(\cdot)$, and are updated during simplification. Given ϵ, we can perform the merging operation if, after merging, any pairs of points on H have distance error ratios at most ϵ. Figures 3 (a) and (b) are original and simplified height maps H and $\widetilde{H}$, where the blue cell in $\widetilde{H}$ is the *larger* cell merged from 2 cells in H.

We refer a cell in H that is deleted from (resp. remaining in) $\widetilde{H}$ during simplification as a *deleted* (resp. *remaining*) cell, and a cell in $\widetilde{H}$ that covers some adjacent deleted and/or previously added cells as an *added* cell. We say that these adjacent deleted cells *belong* to the added cell. A *property of a deleted cell* is that each deleted cell only belongs to one added cell. In Figures 3 (a) and (b), we merge cells a and b to cell e, 10 orange and red points (around e) form all cells in $\widetilde{N}(e)$, $\{a, b\}$ are deleted cells, all other cells in C except $\{a, b\}$ are remaining cells, e is an added cell, and $\{a, b\}$ belong to e. The coordinate and elevation values of the added cell are weighted average values of those of the adjacent deleted cells (if these adjacent deleted cells contain a previously added cell c, the weight is the number of cells in H belonging to c; otherwise, the weight is 1). In Figures 3 (b), we use the coordinate and elevation values of a, b with weight equal to 1 to calculate the corresponding values of e. If we keep merging e with other cells, the weight of e is 2, since the number of cells in H belonging to e is 2. We denote a set of remaining cells and added cells as C_{rema} and C_{add}, so $\widetilde{C} = C_{rema} \cup C_{add}$. A set of deleted cells is denoted as $C - C_{rema}$.

Given a cell $c \in H$, we define the *estimated cell* of c (on $\widetilde{H}$), denoted by $\widetilde{c}$, such that $\widetilde{c}.x = c.x$, $\widetilde{c}.y = c.y$, $\widetilde{c}.z$ is the elevation value of the added cell such that c belongs to (if c is a deleted cell), or $\widetilde{c} = c$ (if c is a remaining cell). In Figure 3 (b), since a is a deleted cell, a belongs to e, we have $\widetilde{a}.x = a.x$, $\widetilde{a}.y = a.y$ and $\widetilde{a}.z = e.z$.

Similar to G, let $\widetilde{G}$ be the simplified height map graph of $\widetilde{H}$. We use $\widetilde{C}$ and $\widetilde{N}(\cdot)$ to substitute C and $N(\cdot)$ in the definition of G to obtain $\widetilde{G}$'s vertices and edges The graphs in Figures 3 (a) and (b) are original and simplified height graphs on the 2D plane. Given a pair of points $\widetilde{s}$ and $\widetilde{t}$ on $\widetilde{H}$, let $\Pi(\widetilde{s},\widetilde{t}|\widetilde{H})$ be the *approximate shortest path* between them passing on ($\widetilde{G}$ of) $\widetilde{H}$. Figure 3 (b) shows $\Pi(\widetilde{c},\widetilde{d}|\widetilde{H})$ in blue line. A notation table can be found in the appendix of Table 4.

2.2 Problem

We introduce the concept of *ϵ-approximate simplified height map* in Definition 1 to describe that $\widetilde{H}$ guarantees that for any pair of points on H, their distance error ratio is at most ϵ.

Definition 1 (ϵ-Approximate Simplified Height Map Definition). *Given H, $\widetilde{H}$ and ϵ, $\widetilde{H}$ is said to be an ϵ-approximate simplified height map of H (or $\widetilde{H}$ is said to be an ϵ-approximation of H) if and only if for any pairs of points s and t on H,*

$$(1-\epsilon)|\Pi(s,t|H)| \le |\Pi(\widetilde{s},\widetilde{t}|\widetilde{H})| \le (1+\epsilon)|\Pi(s,t|H)|. \quad (1)$$

We have the following problem.

Problem 1 (Height Map Simplification Problem). *Given H and ϵ, we want to find an ϵ-approximate simplified height map $\widetilde{H}$ of H with the minimum number of cells.*

The following theorem shows this problem is *NP-hard*.

Theorem 2.1. *The height map simplification problem is NP-hard.*

Proof Sketch. We can transform Minimum T-Spanner Problem [26] (*NP-complete* problem) to the Height Map Simplification Problem in polynomial time, to prove that it is *NP-hard*. The detailed proof appears in the appendix. □

3 RELATED WORK

3.1 Point Cloud and *TIN*

Let P be a point cloud converted from H by cell mapping [25, 45, 60, 72], and T be a *TIN* converted from P by point triangulation [35, 59, 69]. Given a pair of points s and t on P, let $\Pi(s,t|P)$ be the *shortest path* between them passing on (*point cloud graph* [69] of) P. The height map graph and point cloud graph are the same. Given a pair of vertices s and t on T, let $\Pi(s,t|T)$ and $\Pi_N(s,t|T)$ be the *shortest surface path* [43] (passing on faces of T) and *shortest network path* [43] (passing on edges of T) between them, whose distances are called the *shortest surface* and *network distance*, respectively. Let θ be the smallest interior angle of a triangle of T. Figure 2 (c) shows a point cloud with $\Pi(s,t|P)$ in green line, and Figure 2 (d) shows a *TIN* with $\Pi(s,t|T)$ in green line and $\Pi_N(s,t|T)$ in purple line.

Given a pair of points s and t on a height map, since the height map graph is the same as the point cloud graph, we know $|\Pi(s,t|H)| = |\Pi(s,t|P)|$. According to Lemma 4.3 of study [69], we know $|\Pi(s,t|H)| \le \alpha \cdot |\Pi(s,t|T)|$, where $\alpha = \max\{\frac{2}{\sin\theta}, \frac{1}{\sin\theta\cos\theta}\}$, and $|\Pi(s,t|H)| \le |\Pi_N(s,t|T)|$. But, $|\Pi(s,t|H)|$ can be larger or smaller than $|\Pi(s,t|T)|$. In Figures 1 (e) and (f) (see blue lines), $|\Pi(c,e|T)| > |\Pi(c,e|H)|$, but $|\Pi(d,f|T)| < |\Pi(d,f|H)|$.

3.2 Height Map Shortest Path Query Algorithms

There is no existing study finding the shortest path *directly* on a height map. Existing studies [45, 53, 60, 72] adapt shortest path algorithms on the point cloud or *TIN* by converting the height map to a point cloud or a *TIN*, and then computing the shortest path passing on the converted point cloud [69] or *TIN* [27, 42–44, 48, 63, 64, 68, 70] (by defining their 3D surfaces first, e.g., point clouds and *TIN*s, and find paths under their 3D surfaces).

3.2.1 ***Point cloud shortest path query algorithm.*** The best-known exact point cloud shortest path query algorithm called *Point Cloud Shortest Path Query Algorithm* (*PC-SP*) [69] uses Dijkstra's algorithm on the point cloud graph for querying in $O(n \log n)$ time.

3.2.2 ***TIN shortest surface path query algorithms.*** (1) *Exact algorithms*: Two studies use continuous Dijkstra's [48] and checking window [64] algorithms for querying both in $O(n^2 \log n)$ time. The best-known exact *TIN* shortest surface path query algorithm called *TIN Exact Shortest Surface Path Query Algorithm* (*TIN-ESSP*) [27, 63, 70] uses a line to connect the source and destination on a 2D *TIN* unfolded by the 3D *TIN*, for querying in $O(n^2)$ time.

(2) *Approximate algorithms*: All algorithms [42, 44, 68] use discrete Steiner points to construct a graph and use Dijkstra's algorithm it for querying. The best-known $(1+\epsilon)$-approximate *TIN* shortest surface path query algorithm called *TIN Approximate Shortest Surface Path Query Algorithm* (*TIN-ASSP*) [42, 68] runs in $O(\frac{l_{max} n}{\epsilon l_{min}\sqrt{1-\cos\theta}} \log(\frac{l_{max} n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$ time, where l_{max}/l_{min} are the longest/shortest edge's length of the *TIN*, respectively.

3.2.3 ***TIN shortest network path query algorithm.*** Network paths are surface paths restricted to *TIN*'s edge without traversing the faces, resulting in an approximate path. The best-known approximate *TIN* shortest network path query algorithm called *TIN Shortest Network Path Query Algorithm* (*TIN-SNP*) [43] uses Dijkstra's algorithm on *TIN*'s edge for querying in $O(n \log n)$ time.

Adaptations: (1) Given a *Height Map*, we adapt these four best-known point cloud or *TIN* shortest path query algorithms to be algorithms *PC-SP-Adapt(HM)* [69], *TIN-ESSP-Adapt(HM)* [27, 63, 70], *TIN-ASSP-Adapt(HM)* [42, 68] and *TIN-SNP-Adapt(HM)* [43], by converting the height map to a point cloud or a *TIN*, and then computing the shortest path passing on the point cloud or *TIN*. (2) Given a height map without data conversion, algorithm *TIN-ESSP* cannot be directly adapted to the height map since no face can be unfolded in a height map. But, algorithms *PC-SP*, *TIN-ASSP* and *TIN-SNP* can be directly adapted to the height map (by constructing a height map graph), and they become algorithm *HM-SP* (since they are Dijkstra's algorithms).

Drawback: All algorithms are very slow. Our experiments show that for a height map with 50k cells, answering *kNN* queries for all 10k objects needs 4,400s ≈ 1.2 hours, 380,000s ≈ 4.3 days, 70,000s ≈ 19.4 hours and 33,000s ≈ 9.2 hours for algorithms *PC-SP-Adapt(HM)*, *TIN-ESSP-Adapt(HM)*, *TIN-ASSP-Adapt(HM)* and *TIN-SNP-Adapt(HM)*, respectively.

3.3 Height Map Simplification Algorithms

There is no existing study focusing on simplifying a height map. The only closely related work are simplification algorithms on the point cloud [23, 69] or *TIN* [31, 39, 43, 47], where they randomly remove points in a point cloud, or remove a vertex v in a *TIN* and use *triangulation* [35, 59, 69] to form new faces among the adjacent vertices of v, respectively. Among these algorithms, *Point Cloud Simplification Algorithm* (*PC-Simplify*) [23, 69] is the best-known point cloud simplification algorithm, *TIN shortest Network distance Simplification Algorithm* (*TIN-NSimplify*) [43] is the most efficient *TIN* simplification algorithm such that each *TIN* simplification iteration checks for any pairs of vertices on the original *TIN*, whether the relative error of shortest network distances between them on the simplified and original *TIN* are at most ϵ. By using shortest *surface* distances, we obtain the best-known *TIN* simplification algorithm *TIN shortest Surface distance Simplification Algorithm* (*TIN-SSimplify*) [40, 43]. They can be adapted to the height map by converting the height map to a point cloud or a *TIN* first, then applying them for point cloud or *TIN* simplification.

3.3.1 ***Algorithm PC-Simplify***. Since study [23] finds a *TIN* with a minimum number of vertices without *TIN* triangulation, it indeed is a point cloud simplification algorithm. Each point cloud simplification iteration checks whether the z-coordinate value difference of each point on the original and simplified point cloud is at most ϵ. We retain the simplification process and construct a point cloud graph [69], so when it removes a point p, we remove p's adjacent edges in the graph, and check for *any* pairs of points (we can simplify it to check "any pairs of *adjacent* points of p"), whether the relative error of shortest distances between them on the simplified and original point cloud are at most ϵ. Its simplification time is $O(n^2 \log n)$ and output size is $O(n)$.

3.3.2 ***Algorithm TIN-NSimplify***. We can simplify the relative error checks from "*any* pairs of vertices" to "any pairs of *adjacent* vertices of the removed vertex v" for acceleration. Its simplification time is $O(n^2 \log n)$ and output size is $O(n)$.

3.3.3 ***Algorithm TIN-SSimplify***. We can simplify the relative error checks from "*any* pairs of vertices" to "*arbitrary* pairs of points[2] on the *adjacent* faces of v" for acceleration. We further simplify it by placing Steiner points on the *adjacent* faces of v (using any-to-any points *TIN* shortest surface path query technique [40]), and check related to "any pairs of Steiner points". Its simplification time is $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}} \log \frac{1}{\epsilon})$ and output size is $O(n)$.

[2]Given a pair of vertices not adjacent to v, the shortest surface path between them may pass on the adjacent faces of v but not on the adjacent vertices of v. So, only checking the shortest surface distance between any pairs of *adjacent* vertices of v is not sufficient.

Adaptations: (1) Given a *Height Map*, we adapt these three algorithms to be algorithms *PC-Simplify-Adapt(HM)* [23, 69], *TIN-NSimplify-Adapt(HM)* [43] and *TIN-SSimplify-Adapt(HM)* [40, 43], by converting the height map to a point cloud or a *TIN*, and then applying the corresponding algorithms for point cloud or *TIN* simplification. (2) Given a height map without data conversion, algorithm *PC-Simplify* can be directly adapted to the height map (by constructing a height map graph), and its performance is the same as algorithm *PC-Simplify* on point cloud (which has a large simplification time due to its randomized simplification process [23] that lacks pruning techniques). But, algorithms *TIN-NSimplify* and *TIN-SSimplify* cannot, since no vertices can be deleted nor no new faces can be created in a height map.

Drawbacks: (1) *Large output size*: All algorithms do not have optimization techniques during simplification, so their simplified point cloud and *TIN* have a large size, resulting in large shortest path query time on the simplified point cloud and *TIN*. (2) *Large simplification time*: Algorithm *PC-Simplify-Adapt(HM)* is a randomized algorithm without any pruning technique, and algorithms *TIN-NSimplify-Adapt(HM)* and *TIN-SSimplify-Adapt(HM)* involve an expensive operation of re-triangulation during *TIN* simplification, resulting in a large simplification time. Our experiments show that for a height map with 50k cells, the simplification time of algorithms *PC-Simplify-Adapt(HM)*, *TIN-NSimplify-Adapt(HM)*, *TIN-SSimplify-Adapt(HM)* and *HM-Simplify* are 5,250s ≈ 1.5 hours, 7,100s ≈ 2 hours, 103,000s ≈ 1.2 days and 250s ≈ 4.6 min, respectively. The *kNN* query time of 10k objects on the simplified point cloud, *TIN*, or height map are 250s ≈ 4.2 min, 16,800s ≈ 4.7 hours, 67,000s ≈ 18.6 hours and 50s, respectively.

4 METHODOLOGY

4.1 Overview

4.1.1 ***Two phases***. There are two phases for our framework.

(1) **Simplification phase using algorithm *HM-Simplify***: In Figure 4 and Figures 5 (a) - (f), given a height map H, we generate a simplified height map $\widetilde{H}$ by iteratively merge some adjacent cells whenever $\widetilde{H}$ is an ϵ-approximation of H (H is deleted).

(2) **Shortest path query phase using algorithm *HM-SP***: In Figure 4 and Figure 5 (g), given $\widetilde{H}$, a pair of points s and t on H, we first calculate s and t's estimated points $\widetilde{s}$ and $\widetilde{t}$ on $\widetilde{H}$, and then use Dijkstra's algorithm [32] on $\widetilde{H}$ to compute $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$.

4.1.2 ***Two components***. There are two *hash tables* involved.

(1) **The containing table $O(\cdot)$**: Given an added cell c in $\widetilde{H}$, $O(c)$ returns the set of deleted cells $\{p_1, p_2, \dots\}$ in H belonging to c in $O(1)$ time. In Figure 5 (b), we merge $\{a, b\}$ to cell c, the deleted cells $\{a, b\}$ belongs to the added cell c, so $O(c) = \{a, b\}$.

(2) **The belonging table $O^{-1}(\cdot)$**: Given a deleted cell c in H, $O^{-1}(c)$ returns the added cell c' in $\widetilde{H}$ such that c belongs to c' in $O(1)$ time. In Figure 5 (b), $O^{-1}(a) = c$.

4.2 Key Idea

4.2.1 ***Significant output size reducing for algorithm HM-Simplify***. It can significantly reduce the number of cells in $\widetilde{H}$, i.e., reduce output size, to further reduce the shortest path query time

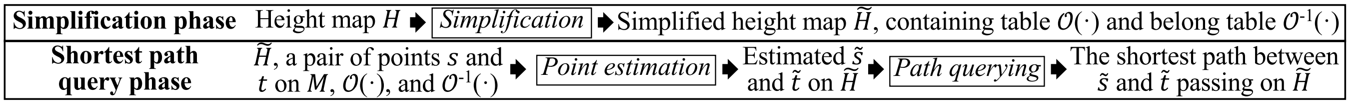
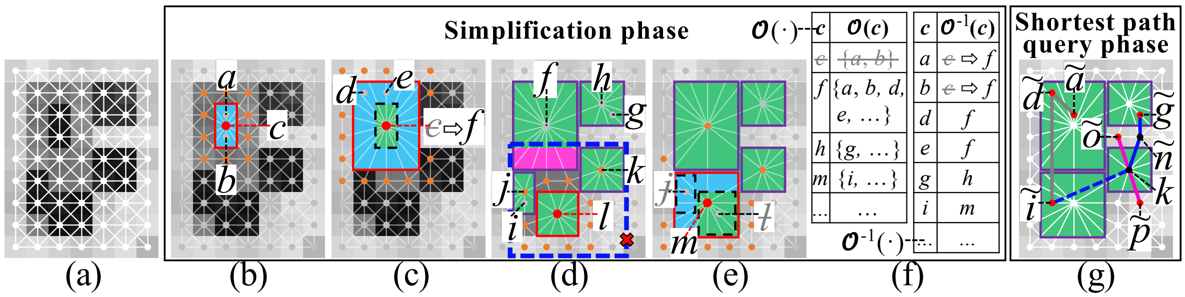


Figure 4: Overview of algorithm *HM-Simplify* and *HM-SP*



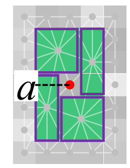


Figure 5: Details of algorithm *HM-Simplify* and *HM-SP*

Figure 6: No further merging

of algorithm *HM-SP* on $\widetilde{H}$ using *a novel cell merging technique* with two merging types:

(1) **Merge two cells**: We start by choosing two adjacent remaining and non-boundary (not lying on the boundary of H) cells with the smallest elevation value variance. In Figure 5 (b), we merge cells $\{a, b\}$ to cell c, and obtain $\widetilde{H}$. If $\widetilde{H}$ is an ϵ-approximation of H, we confirm this merging and proceed to the next merging type. If not, we terminate the algorithm.

(2) **Merge added cell with neighbor cells**: Given an added cell c from the previous merge, we merge c with its neighbor cells, i.e., expand c's non-boundary neighbor cells into left, right, top and/or bottom directions to reduce the number of cells in $\widetilde{H}$, where expanding left (resp. right) covers neighbors cells with x-coordinate value smaller (resp. larger) than c, and expanding top (resp. bottom) covers neighbors cells with y-coordinate value larger (resp. smaller) than c, and the origin is set at left-bottom side of H. Let Direction, i.e., $Dir = \{(L, R, T, B), (L, R, T, \cdot), (L, R, \cdot, B), (L, \cdot, T, B), (\cdot, R, T, B), (L, R, \cdot, \cdot), (L, \cdot, T, \cdot), (L, \cdot, \cdot, B), (\cdot, R, T, \cdot), (\cdot, R, \cdot, B), (\cdot, \cdot, T, B), (L, \cdot, \cdot, \cdot), (\cdot, R, \cdot, \cdot), (\cdot, \cdot, T, \cdot), (\cdot, \cdot, \cdot, B)\}$ be the expanded directions, where L, R, T, B means that we expand c into *left*, *right*, *top* and *bottom* directions, and $\cdot$ means no expansion in that direction. For example, $Dir[1] = (L, R, T, \cdot)$ means that we cover c's neighbors cells with x-coordinate value smaller and larger than c, and y-coordinate value larger than c.

In Figure 5 (c), we merge c with $\{d, e, \dots\}$ to form f, i.e., expand c into (L, R, T, B) directions, and obtain $\widetilde{H}$. If $\widetilde{H}$ is an ϵ-approximation of H, we confirm this merging and repeat. If not, we go back to the *two cells merging* type by selecting two new cells. In Figure 5 (d), we merge l with $\{j, k, \dots\}$ to form a potential newly added cell with a blue frame, i.e., expand l into (L, R, T, B) directions. For one of l's neighbor cells, i.e., k, we cover it as a whole to reduce the number of cells in $\widetilde{H}$. But, four pink deleted cells will belong to both f and the potential newly added cell, violating the property of the deleted cell in Section 2.1.2 that each deleted cell only belongs to one added cell, i.e., we do not want the potential newly added cell to overlap with any added cell f. So, we expand l into the direction of other elements in Dir. In Figure 5 (e), we merge l with $\{j, \dots\}$ to form m, i.e., expand l into $(L, \cdot, T, \cdot)$ directions, and obtain $\widetilde{H}$. If $\widetilde{H}$ is an ϵ-approximation of H, we confirm this merging and repeat. If not, we go back to the *two cells merging* type. Similarly, we cannot expand four green added cells to cover a in Figure 6.

4.2.2 *Efficient simplification for algorithm HM-Simplify*.

There are two reasons why it has a small simplification time.

(1) **Efficient height map shortest path query**: We use efficient algorithm *HM-SP* to check whether $\widetilde{H}$ is an ϵ-approximation of H.

(2) **Efficient ϵ-approximate simplified height map checking**: Checking whether $\widetilde{H}$ is an ϵ-approximation of H involves checking whether Inequality 1 is satisfied for *all* points on H, this naive method is time-consuming. Instead, *our efficient checking technique* only checks distances related to newly added cells' *neighbor* cells.

4.3 Simplification Phase

We illustrate the simplification phase using algorithm *HM-Simplify* in Algorithm 1, which uses Algorithm 2 for 2 times. Algorithm 1 shows the two merging types, and Algorithm 2 clearly updates the neighborhoods of each cell, and other related components.

4.3.1 *Detail and example for Algorithm 1*.

In each simplification iteration, let $\widehat{C} = \{p_1, p_2, \dots\}$ be a set of adjacent cells to be merged, and c_{add} be an added cell merged from cells in $\widehat{C}$. Let *FindTwoCell* $(\widetilde{H})$ be a function that returns two adjacent remaining and non-boundary cells in $\widetilde{H}$ with the smallest elevation values variance, and *FindAddedCellNeig* $(\widetilde{H}, c_{add}, i)$ be a function that returns a set of cells in $\widetilde{H}$ including c_{add} and its expanded non-boundary neighbor cells (as a whole) in $Dir[i]$ directions without violating the property of deleted cells. Both functions return *NULL* if such cells do not exist. The following shows Algorithm 1 with an example.

(1) *Merge two cells* (lines 2-5): In Figures 5 (a) and (b), $\widehat{C}$ = *FindTwoCell* $(\widetilde{H})$ = $\{a, b\}$, and we can merge cells in $\widehat{C}$, suppose that *UpdateSatisfy* is *True*, we obtain $c_{add} = c$ and $\widetilde{H}$.

Algorithm 1 *HM-Simplify* (H)

Input: $H = (C, N(\cdot) = \emptyset)$
Output: $\widetilde{H}$, $O(\cdot)$ and $O^{-1}(\cdot)$

1: initialize $N(\cdot)$ using C, $C_{rema} \leftarrow C$, $C_{add} \leftarrow \emptyset$, $\widetilde{N}(\cdot) \leftarrow N(\cdot)$, $O(\cdot) \leftarrow \emptyset$, $O^{-1}(\cdot) \leftarrow \emptyset$
2: $\widehat{C} \leftarrow$ *FindTwoCell* $(\widetilde{H} = (C_{rema} \cup c_{add}, \widetilde{N}(\cdot)))$
3: **while** $\widehat{C}$ is *NON-NULL* **do**
4: $\quad c_{add} \leftarrow \emptyset$
5: $\quad$**if** *UpdateSatisfy* $(C_{rema}, C_{add}, \widetilde{N}(\cdot), \widehat{C}, c_{add}, O(\cdot), O^{-1}(\cdot))$ **then**
6: $\quad\quad i \leftarrow 0$
7: $\quad\quad$**while** $i < 15$ **do**
8: $\quad\quad\quad \widehat{C} \leftarrow$ *FindAddedCellNeig* $(\widetilde{H} = (C_{rema} \cup c_{add}, \widetilde{N}(\cdot)), c_{add}, i)$
9: $\quad\quad\quad$**if** $\widehat{C}$ is *NON-NULL AND UpdateSatisfy* $(C_{rema}, C_{add}, \widetilde{N}, \widehat{C}, c_{add}, O(\cdot), O^{-1}(\cdot))$ **then**
10: $\quad\quad\quad\quad i \leftarrow 0$
11: $\quad\quad\quad$**else**
12: $\quad\quad\quad\quad i \leftarrow i + 1$
13: $\quad \widehat{C} \leftarrow$ *FindTwoCell* $(\widetilde{H} = (C_{rema} \cup c_{add}, \widetilde{N}(\cdot)))$
14: **return** $\widetilde{H} = (C_{rema} \cup C_{add}, \widetilde{N}(\cdot))$, $O(\cdot)$ and $O^{-1}(\cdot)$

(2) *Merge added cell with neighbor cells* (lines 6-12). In Figure 5 (c), $c_{add} = c, i = 0 < 15$, $\widehat{C}$ = *FindAddedCellNeig* $(\widetilde{H}, c_{add}, i)$ $= \{c, d, e, \dots\}$, i.e., we can expand c into $Dir[0] = (L, R, T, B)$ directions, suppose that *UpdateSatisfy* is *True*, we obtain $c_{add} = f$ and $\widetilde{H}$, and we set $i = 0$. We repeat it, suppose that after we expand f into the direction of other elements in *Dir*, *UpdateSatisfy* is always *False*, we exit this loop. The following is the iteration.

(3) *Merge two or added cell with neighbor cells* (lines 2-13): In Figure 5 (d), we obtain h, j, k, l and $\widetilde{H}$. Then, we further process l.

(4) *Merge added cell with neighbor cells* (lines 6-9): In Figure 5 (d), $c_{add} = l, i = 0 < 15$, we want to use *FindAddedCellNeig* $(\widetilde{H}, c_{add}, i)$ to expand l into $Dir[0] = (L, R, T, B)$ directions (to include $\{j, k, \dots\}$). We get the potential newly added cell with a blue frame. But, four pink deleted cells will belong to both f and the newly added cell, violating the property of the deleted cell, so such cells do not exist and $\widehat{C}$ is *NULL*. We repeat the process until $\widehat{C}$ is *NON-NULL*. In Figure 5 (e), $c_{add} = l, i = 6 < 15$, $\widehat{C}$ = *FindAddedCellNeig* $(\widetilde{H}, c_{add}, i)$ $= \{l, j, \dots\}$, i.e., we can expand l into $Dir[6] = (L, \cdot, T, \cdot)$ directions, suppose that *UpdateSatisfy* is *True*, we obtain $c_{add} = m$ and $\widetilde{H}$.

4.3.2 *Detail and example for Algorithm 2.*

The following shows Algorithm 2 with an example. Figure 5 (b) and (c) illustrate steps 1–4 and 5–8, respectively. Figures 5 (d) and (e) are similar.

(1) *Update $O'(\cdot)$ and $O^{-1\prime}(\cdot)$* (lines 3-9): $c_{add} = c$ and $\widehat{C} = \{a, b\}$, since all cells in $\widehat{C}$ are in C_{rema}, we have $O'(c) = \{a, b\}$, $O^{-1\prime}(a) = c$, $O^{-1\prime}(b) = c$.

(2) *Update neighbor cells* (lines 10-13): We update c and cells represented in orange points as neighbors of each other.

(3) *Update $\widetilde{H}'$* (lines 14-19): $\{a, b\}$ are deleted from C'_{rema} and c is added into C'_{add}, so $C'_{add} = \{c\}$.

(4) *Check ϵ-approximate simplified height map* (lines 20-22): Suppose that $\widetilde{H}'$ is an ϵ-approximation of H, we have $C_{rema} = C \setminus \{a, b\}$, $C_{add} = \{c\}$, updated $\widetilde{N}(\cdot)$ and $\widetilde{H}$. In Figure 5 (f), we update $O(\cdot)$ and $O^{-1}(\cdot)$. The following is the iteration.

(5) *Update $O'(\cdot)$ and $O^{-1\prime}(\cdot)$* (lines 3-9): $c_{add} = f$, $\widehat{C} = \{f, d, e, \dots\}$ and $C_{add} = \{c\}$, since for cells in $\widehat{C}$, c is in C_{add} and other cells are in C_{rema}, we have $O'(f) = \{a, b, d, e, \dots\}$, $O^{-1\prime}(a) = f$, $O^{-1\prime}(b) = f$, $O^{-1\prime}(d) = f$, $O^{-1\prime}(e) = f$, …, and delete $O'(c)$.

Algorithm 2 *UpdateSatisfy* $(C_{rema}, C_{add}, \widetilde{N}(\cdot), \widehat{C}, c_{add}, O(\cdot), O^{-1}(\cdot))$

Input: $C_{rema}, C_{add}, \widetilde{N}(\cdot), \widehat{C}, c_{add}, O(\cdot)$ and $O^{-1}(\cdot)$
Output: updated $C_{rema}, C_{add}, \widetilde{N}(\cdot), O(\cdot), O^{-1}(\cdot)$, and whether the updated height map is an ϵ-approximation of H

1: $C'_{rema} \leftarrow C_{rema}$, $C'_{add} \leftarrow C_{add}$, $\widetilde{N}'(\cdot) \leftarrow \widetilde{N}(\cdot)$, $\widetilde{N}'(C_{add}) \leftarrow \emptyset$, $O'(\cdot) \leftarrow O(\cdot)$, $O^{-1\prime}(\cdot) \leftarrow O^{-1}(\cdot)$
2: merge cells in $\widehat{C}$ to form cell c_{add}
3: **for** each $c \in \widehat{C}$ **do**
4: $\quad$**if** $c \in C_{rema}$ **then**
5: $\quad\quad O'(c_{add}) \leftarrow O'(c_{add}) \cup \{p\}$, $O^{-1\prime}(c) \leftarrow \{c_{add}\}$
6: $\quad$**else if** $c \in C_{add}$ **then**
7: $\quad\quad$**for** each $c' \in O'(c)$ **do**
8: $\quad\quad\quad O'(c_{add}) \leftarrow O'(c_{add}) \cup \{c'\}$, $O^{-1\prime}(c') \leftarrow \{c_{add}\}$
9: $\quad\quad O'(\cdot) \leftarrow O'(\cdot) - \{O'(c)\}$
10: **for** each $c \in \widehat{C}$ **do**
11: $\quad$**for** each $c' \in N(c)$ such that $c' \notin \widehat{C}$ **do**
12: $\quad\quad \widetilde{N}'(c_{add}) \leftarrow \widetilde{N}'(c_{add}) \cup \{c'\}$, $\widetilde{N}'(c') \leftarrow \widetilde{N}'(c') - c \cup \{c_{add}\}$
13: clear $\widetilde{N}'(c)$ for each $c \in \widehat{C}$
14: **for** each $c \in \widehat{C}$ **do**
15: $\quad$**if** $c \in C_{rema}$ **then**
16: $\quad\quad C'_{rema} \leftarrow C'_{rema} - \{p\}$
17: $\quad$**else if** $c \in C_{add}$ **then**
18: $\quad\quad C'_{add} \leftarrow C'_{add} - \{p\}$
19: $C'_{add} \leftarrow C'_{add} \cup \{c_{add}\}$
20: **if** $\widetilde{H}' = (C'_{rema} \cup C'_{add}, \widetilde{N}'(\cdot))$ is an ϵ-approximation of H **then**
21: $\quad C_{rema} \leftarrow C'_{rema}$, $C_{add} \leftarrow C'_{add}$, $\widetilde{N}(\cdot) \leftarrow \widetilde{N}'(\cdot)$, $O(\cdot) \leftarrow O'(\cdot)$, $O^{-1}(\cdot) \leftarrow O^{-1\prime}(\cdot)$
22: $\quad$**return** *True*
23: **return** *False*

(6) *Update neighbor cells* (lines 10-13): We update f and cells represented in orange points as neighbors of each other.

(7) *Update $\widetilde{H}'$* (lines 14-19): $\{d, e, \dots\}$ are deleted from C'_{rema}, c is deleted from C'_{add}, and f is added into C'_{add}, so $C'_{add} = \{f\}$.

(8) *Check ϵ-approximate simplified height map* (lines 20-22): Suppose that $\widetilde{H}'$ is an ϵ-approximaion of H, we have $C_{rema} = C \setminus \{a, b, d, e, \dots\}$, $C_{add} = \{f\}$, updated $\widetilde{N}(\cdot)$ and $\widetilde{H}$. In Figure 5 (f), we update $O(\cdot)$ and $O^{-1}(\cdot)$.

4.4 Efficient ϵ-Approximate Simplified Height Map Checking

Checking whether $\widetilde{H}$ is an ϵ-approximation of H involves many unnecessary distance checks. We give one notation first.

4.4.1 *Notation.*

Adjacent added cells: Given an added cell $c_{add} \in C_{add}$, we define a set of *adjacent added cells* of c_{add}, denoted by $A(c_{add})$, to be a set of added cells in C_{add} which contain c_{add} and are adjacent to each other. In Figure 7, $A(c_{add} = a) = \{a, b\}$.

4.4.2 *Detail and example.*

We then discuss our efficient checking. In Definition 1, we change "any pairs of points s and t on H" (involving more points) to points of the following three types of cells related to the *neighbor* cells of the added cell c_{add} (involving fewer points), and then use Inequality 1 for each type to efficiently check whether $\widetilde{H}$ is an ϵ-approximation of H.

(1) **Remaining to Remaining cells (*R2R*)**: We change to "any pairs of points s and t of remaining cells that are neighbor cells of each added cell in $A(c_{add})$". Figure 7 shows these points in orange.

(2) **Remaining to Deleted cells (*R2D*)**: We change to "any point s of remaining cell that is a neighbor cell of each cell in $A(c_{add})$,

and any point t of deleted cell that belongs to each added cell in $A(c_{add})$". Figure 7 shows these points in orange (corresponding to s) and purple (corresponding to t).

(3) **<u>D</u>eleted to <u>D</u>eleted cells (*D2D*)**: We change to "any pairs of points s and t of deleted cells that belong to each added cell in $A(c_{add})$". Figure 7 shows these points in purple.

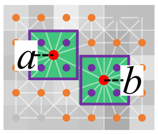


Figure 7: Distance checking

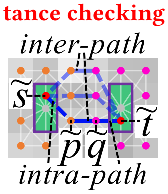


Figure 8: Intra- and inter-paths

Table 1: Height map datasets

Name	$\lvert n\rvert$
Original dataset	
GunnisonForest (*GF_h*) [11, 69]	0.5M
LaramieMount (*LM_h*) [14, 69]	0.5M
RobinsonMount (*RM_h*) [18, 69]	0.5M
BearHead (*BH_h*) [6, 61, 62, 69]	0.5M
EaglePeak (*EP_h*) [6, 61, 62, 69]	0.5M
Small-version dataset	
GF_h-small	1k
LM_h-small	1k
RM_h-small	1k
BH_h-small	1k
EP_h-small	1k
Multi-resolution dataset	
GF_h multi-resolution	5M, 10M, 15M, 20M, 25M
LM_h multi-resolution	5M, 10M, 15M, 20M, 25M
RM_h multi-resolution	5M, 10M, 15M, 20M, 25M
BH_h multi-resolution	5M, 10M, 15M, 20M, 25M
EP_h multi-resolution	5M, 10M, 15M, 20M, 25M
EP_h-small multi-resolution	10k, 20k, 30k, 40k, 50k

4.5 Shortest Path Query Phase

We illustrate the shortest path query phase using algorithm *HM-SP* on the simplified height map graph $\widetilde{G}$. Intuitively, we use Dijkstra's algorithm between the source and destination points on $\widetilde{G}$. But, when the source point s is on a deleted cell c_s (i.e., s does not exist in $\widetilde{G}$), a naive algorithm uses Dijkstra's algorithm multiple times using all points on neighbor cells of $O^{-1}(c_s)$ as sources for approximation. But, we propose an efficient algorithm using *an efficient implicit edge insertion technique* to use Dijkstra's algorithm only once. We give two types of notations first.

4.5.1 ***Notation.*** **Intra-path and inter-path**: An intra-path is a path from a point (on a deleted cell) to a point (on a remaining/added cell) and an inter-path is a path from a point (on a remaining/added cell) to a point (on a remaining/added cell). Specifically, given a point m on a deleted cell c_m and a point n on a cell $c_n \in \widetilde{N}(O^{-1}(c_m))$, we call the path between them passing on $\widetilde{H}$ as the *intra-path*, and denote it by $\Pi_1(\widetilde{m}, \widetilde{n}|\widetilde{H}) = \langle \widetilde{m}, \widetilde{n} \rangle$. Given a pair of points $\widetilde{p}$ and $\widetilde{q}$ on cells $\widetilde{c_p}$ and $\widetilde{c_q}$ in $\widetilde{C}$, we call the path between them passing on $\widetilde{H}$ as the *inter-path*, and denote it by $\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{H})$. In Figures 5 (g) and 8, $\Pi_1(\widetilde{g}, \widetilde{n}|\widetilde{H})$ and $\Pi_1(\widetilde{s}, \widetilde{p}|\widetilde{H})$ in blue dashed lines are two intra-paths, $\Pi_2(\widetilde{n}, k|\widetilde{H})$ and $\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{H})$ in blue solid lines are two inter-paths.

4.5.2 ***Detail and example.*** We then discuss the two steps.

(1) **Point estimation**: Given a pair of points s and t (on cells c_s and c_t) of H, we estimate $\widetilde{s}$ using s, such that $\widetilde{s}.x = s.x$, $\widetilde{s}.y = s.y$, $\widetilde{s}.z = O^{-1}(c_s).z$ (if c_s is a deleted cell), or $\widetilde{s} = s$ (if c_s is a remaining cell). We estimate $\widetilde{t}$ similarly.

(2) **Path querying**: There are three cases depending on whether c_s and c_t are deleted or remaining cells.

(i) *Both cells deleted*: Firstly, there are two special cases that we return $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H}) = \langle \widetilde{s}, \widetilde{t} \rangle$. One is that c_s and c_t belong to the different added cells c_u and c_v, where c_u and c_v are neighbor. The other one is that c_s and c_t belong to the same added cell. In Figure 5 (g), $\Pi(\widetilde{d}, \widetilde{i}|\widetilde{H}) = \langle \widetilde{d}, \widetilde{i} \rangle$ (i.e., the first case) and $\Pi(\widetilde{a}, \widetilde{d}|\widetilde{H}) = \langle \widetilde{a}, \widetilde{d} \rangle$ (i.e., the second case). Secondly, for common case, we return $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ by concatenating the intra-path $\Pi_1(\widetilde{s}, \widetilde{p}|\widetilde{H})$, the inter-path $\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{H})$, and the intra-path $\Pi_1(\widetilde{q}, \widetilde{t}|\widetilde{H})$, such that $|\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})| = \min_{\forall \widetilde{c_p} \in \widetilde{N}(O^{-1}(c_s)), \widetilde{c_q} \in \widetilde{N}(O^{-1}(c_t))} |\Pi_1(\widetilde{s}, \widetilde{p}|\widetilde{H})| + |\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{H})| + |\Pi_1(\widetilde{q}, \widetilde{t}|\widetilde{H})|$, where $\widetilde{p}$ and $\widetilde{q}$ are a pair of points on cells $\widetilde{c_p}$ and $\widetilde{c_q}$. In Figure 8, orange and pink points denote possible points on cells $\widetilde{N}(O^{-1}(c_s))$ and $\widetilde{N}(O^{-1}(c_t))$, $\widetilde{p}$ and $\widetilde{q}$ are points resulting in the minimum distance among these points, respectively. A naive algorithm uses Dijkstra's algorithm on $\widetilde{H}$ with each point on cell in $\widetilde{N}(O^{-1}(c_s))$ as a source to compute inter-paths. But, our efficient algorithm uses Dijkstra's algorithm only once. If the number of cells in $\widetilde{N}(O^{-1}(c_s))$ is less than that of in $\widetilde{N}(O^{-1}(c_t))$, we implicitly insert intra-paths between $\widetilde{c_s}$ and each cell in $\widetilde{N}(O^{-1}(c_s))$ as edges in $\widetilde{G}$ (we remove them after this calculation), and then we use Dijkstra's algorithm on $\widetilde{G}$ with $\widetilde{s}$ as a source, and terminate after visiting all points on cells in $\widetilde{N}(O^{-1}(c_t))$, to compute the intra-path connecting to $\widetilde{s}$ and the inter-path. We append them with the intra-path connecting to $\widetilde{t}$ and obtain $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$. If the number of cells in $\widetilde{N}(O^{-1}(c_s))$ is larger than that of in $\widetilde{N}(O^{-1}(c_t))$, we swap s and t. In Figure 5 (g), $\Pi(\widetilde{g}, \widetilde{i}|\widetilde{H}) = \langle \widetilde{g}, \widetilde{n}, k, \widetilde{i} \rangle$.

(ii) *One cell deleted and one cell remaining*: If $c_s \in C_{rema}$, the inter-path connecting to s does not exist, we use Dijkstra's algorithm on $\widetilde{G}$ with s as a source, and terminate after visiting all points on cells in $\widetilde{N}(O^{-1}(c_t))$. We append them with the intra-path connecting to $\widetilde{t}$ and obtain $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$. If $t \in C_{rema}$, we swap s and t. In Figure 5 (g), $\Pi(\widetilde{n}, \widetilde{i}|\widetilde{H}) = \langle \widetilde{n}, k, \widetilde{i} \rangle$.

(iii) *Both cells remaining*: Both inter-paths do not exist, we use Dijkstra's algorithm on $\widetilde{G}$ between s and t to obtain $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$. In Figure 5 (g), $\Pi(\widetilde{o}, \widetilde{p}|\widetilde{H}) = \langle \widetilde{o}, k, \widetilde{p} \rangle$.

4.6 Proximity Query Algorithms

Given H and $\widetilde{H}$, a query point i on cell c_i, a set of n' interested points on cells on H or $\widetilde{H}$, two parameters k (k value in *kNN* query) and r (range value in range query), we can answer *kNN* and range queries using algorithm *HM-SP*. A naive algorithm uses it for n' times between i and all interested points, and then performs a linear scan on the paths to compute *kNN* and range query results.

But, we propose an efficient algorithm using *an efficient parallel computation technique* to use it (i.e., Dijkstra's algorithm) only once. (1) For algorithm *HM-SP* on H, we use Dijkstra's algorithm once with i as a source and all interested points as destinations, and then directly return *kNN* and range query results without any linear scan, since these paths are already sorted in order during the execution of Dijkstra's algorithm. (2) For algorithm *HM-SP* on $\widetilde{H}$, we also use Dijkstra's algorithm once. Except for two special cases in Section 4.5.2 cases (2-i) that directly return the path $\Pi(\widetilde{i}, \widetilde{j}|\widetilde{H}) = \langle \widetilde{i}, \widetilde{j} \rangle$, where j is the interested point of an interested cell c_j, there are two cases. (i) If c_i is a deleted cell, we change "s" to "i", "terminate after Dijkstra's algorithm visits all points on cells in $\widetilde{N}(O^{-1}(c_t))$" to "terminate after Dijkstra's algorithm visits all points on cells in S, where S is a set of cells, such that for each interested cell c_j, we store c_j in S if c_j is a remaining cell, or we store cells in $\widetilde{N}(O^{-1}(c_j))$ into S if c_j is a deleted cell", and "append them with the intra-path connecting to $\widetilde{t}$" to "append them with the intra-path connecting to each $\widetilde{j}$ if c_j is a deleted cell" in Section 4.5.2 case (2-i). (ii) If c_i is a remaining cell, we apply the same three changes in Section 4.5.2 case (2-ii). Finally, we perform a linear scan on the paths to compute *kNN* and range query results.

4.7 Add-on Data Structure

Given a $(1+\epsilon)$-approximate simplified graph of a complete graph, study [50] constructs a $(1+\epsilon')$-approximate data structure (i.e., a graph $G_{\widetilde{H}}$) on the simplified graph, to return the $(1+\epsilon')(1+\epsilon)$ approximate shortest paths between any pairs vertices in $O(1)$ time. $G_{\widetilde{H}}$ can be used in the simplified height map graph of $\widetilde{H}$ in algorithm *HM-Simplify*, and we can use *HM-SP* on $G_{\widetilde{H}}$ to return paths in $O(1)$ time. We denote our adapted algorithms (after using $G_{\widetilde{H}}$) to be algorithms *HM-Simplify Data Structure* (*HM-Simplify-DS*) and *HM-SP Data Structure* (*HM-SP-DS*).

4.8 Theoretical Analysis

4.8.1 ***Algorithms HM-Simplify and HM-SP.*** We analyze them in Theorems 4.1 and 4.2.

Theorem 4.1. *The simplification time, number of cells in $\widetilde{H}$ and output size of algorithm HM-Simplify are $O(n\sqrt[3]{n}\log n)$, $O(\frac{n}{\mu})$ and $O(\frac{n}{\mu})$, respectively, where $\mu \in [2, \log n]$ is a constant depending on H and ϵ, and $\mu \in [5, 88]$ in our experiments. Given H, it returns $\widetilde{H}$ such that $(1-\epsilon)|\Pi(s,t|H)| \leq |\Pi(\widetilde{s},\widetilde{t}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s,t|H)|$ for any pairs of points s and t on H.*

Proof Sketch. The simplification time is due to the usage of Dijkstra's algorithm in $O(n\log n)$ time for $O(1)$ cells in *R2R*, *R2D* and *D2D* checking, with total $O(\sqrt[3]{n})$ cell merging iterations. The number of cells in $\widetilde{H}$ and output size are due to the total n cells on H and $O(\mu)$ deleted cells belonging to each added cell in average. The error guarantee of $\widetilde{H}$ is due to the *R2R*, *R2D* and *D2D* checking. The detailed proof appears in the appendix. □

Theorem 4.2. *The shortest path query time and memory usage of algorithm HM-SP are $O(n\log n)$ and $O(n)$ on H, and are $O(\frac{n}{\mu}\log\frac{n}{\mu})$ and $O(\frac{n}{\mu})$ on $\widetilde{H}$, respectively. It returns the exact shortest path passing on H, and returns an approximate shortest path passing on $\widetilde{H}$ such that $(1-\epsilon)|\Pi(s,t|H)| \leq |\Pi(\widetilde{s},\widetilde{t}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s,t|H)|$.*

Proof. Since there are $O(n)$ and $O(\frac{n}{\mu})$ cells in H and $\widetilde{H}$, respectively, algorithm *HM-SP* is Dijkstra's algorithm which returns the exact result on H and $\widetilde{H}$, and $\widetilde{H}$ is an ϵ-approximation of H, we finish the proof. □

4.8.2 ***Proximity query algorithms.*** We show query time and error guarantee of *kNN* and range queries using algorithm *HM-SP* on H and $\widetilde{H}$ in Theorem 4.3. Given a query point i, let p_f and p'_f be the furthest point to i computed using the ground-truth or optimal distance and a studied algorithm (i.e., computed by algorithm *HM-SP* on H and $\widetilde{H}$ in our case), respectively. Let the error ratio of *kNN* and range queries be $(\frac{|\Pi(i,p'_f|Z)|}{|\Pi(i,p_f|Z)|}-1)$, where $Z \in \{H, P, T\}$ means the 3D surface (height map, point cloud or *TIN*) used for calculating the ground-truth or optimal distance (i.e., $Z = H$ in our case).

Theorem 4.3. *The kNN and range query time of using algorithm HM-SP are both $O(n\log n)$ on H and $O(\frac{n}{\mu}\log\frac{n}{\mu})$ on $\widetilde{H}$, respectively. It returns the exact result on H and has an error ratio $\frac{2\epsilon}{1-\epsilon}$ on $\widetilde{H}$ for both kNN and range queries, respectively.*

Proof Sketch. The query time for algorithm *HM-SP* is due to the usage of it once. The error arises from its error. □

4.8.3 ***Algorithms HM-Simplify-DS and HM-SP-DS.*** We analyze them in Theorems 4.4 and 4.5.

Theorem 4.4. *The simplification time, number of edges in $G_{\widetilde{H}}$ and output size of algorithm HM-Simplify-DS are $O(n\sqrt[3]{n}\log n + \frac{n^2}{\mu^2}\log^2\frac{n}{\mu})$, $O(\frac{n}{\mu}\log\frac{n}{\mu})$ and $O(\frac{n}{\mu}\log\frac{n}{\mu})$, respectively. Given a height map H, it returns $G_{\widetilde{H}}$ such that $|\Pi(\widetilde{s},\widetilde{t}|G_{\widetilde{H}})| \leq (1+\epsilon')(1+\epsilon)|\Pi(s,t|H)|$ for any pairs of points s and t on H, where $\Pi(\widetilde{s},\widetilde{t}|G_{\widetilde{H}})$ is the approximate shortest path between $\widetilde{s}$ and $\widetilde{t}$ passing on $G_{\widetilde{H}}$.*

Theorem 4.5. *The shortest path query time, kNN and range query time and memory usage of algorithm HM-SP-DS are $O(1)$, $O(n')$ and $O(\frac{n}{\mu}\log\frac{n}{\mu})$, respectively. It returns an approximate shortest path passing on $G_{\widetilde{H}}$ such that $|\Pi(\widetilde{s},\widetilde{t}|G_{\widetilde{H}})| \leq (1+\epsilon')(1+\epsilon)|\Pi(s,t|H)|$, and has an error ratio $\epsilon'\cdot\epsilon+\epsilon'+\epsilon$ for both kNN and range queries.*

Proof Sketch. The detailed proofs of Theorems 4.4 and 4.5 appear in the appendix. □

5 EMPIRICAL STUDIES

5.1 Experimental Setup

We performed experiments using a Linux machine with 2.2 GHz CPU and 512GB memory. Algorithms were implemented in C++. The experiment setup follows studies [42, 43, 61, 62, 68–70].

5.1.1 ***Datasets.*** (1) Height map datasets: We conducted experiments using 34 (= 5+5+24) real height map datasets listed in Table 1, where the subscript h indicates a height map. (i) *5 Original datasets*: GF_h [11, 69], LM_h [14, 69] and RM_h [18, 69] are originally represented as height maps obtained from Google Earth [9]. They are used in study [69]. BH_h [6, 61, 62, 69] and EP_h [6, 61, 62, 69] are originally represented as points clouds, we created height maps with cell's 2D coordinate and elevation values equal to the z-coordinate values of these points. They are used in studies [61, 62, 69]. These five datasets have a 20km × 20km region with a 28m × 28m resolution [43, 62, 68, 69]. (ii) *5 Small-version datasets*: They are generated using the same region as the original datasets, with a 633m × 633m resolution, following the dataset generation steps [62, 68, 69]. (iii) *24 Multi-resolution datasets*: They are generated similarly with varying numbers of cells. (2 & 3) *Point cloud* and *TIN* datasets: We convert the height map datasets to 34 point cloud datasets by cell mapping [25, 45, 60, 72], and then to 34 *TIN* datasets by point triangulation [35, 59, 69]. We use p and t as subscripts, respectively.

5.1.2 ***Algorithms.*** (1) To solve our problem on Height Maps, we adapted existing algorithms on point clouds or *TIN*s, by converting the given height maps to point clouds [25, 45, 60, 72] or *TIN*s [25, 35, 45, 59, 60, 69, 72] so that the existing algorithms could be performed, and append "*-Adapt(HM)*" in algorithm names. We have 4 simplification algorithms: (i) the best-known adapted *TIN* simplification algorithm *TIN-SSimplify-Adapt(HM)* [40, 43], (ii) adapted *TIN* shortest network distance simplification algorithm *TIN-NSimplify-Adapt(HM)* [43], (iii) the best-known adapted point cloud simplification algorithm *PC-Simplify-Adapt(HM)* [23, 69] and (iv) our height map simplification algorithm *HM-Simplify*. We have 5 proximity query algorithms: (i) the best-known adapted exact *TIN* shortest surface path query algorithm *TIN-ESSP-Adapt(HM)* [27, 63, 70], (ii) the best-known adapted approximate *TIN* shortest surface path query algorithm *TIN-ASSP-Adapt(HM)* [42, 68], (iii) the best-known adapted approximate *TIN* shortest network path query algorithm *TIN-SNP-Adapt(HM)* [43], (iv) the best-known adapted exact point cloud shortest path query algorithm *PC-SP-Adapt(HM)* [69] and (v) our exact height map shortest path query algorithm *HM-SP*. The exact algorithms refer to their particular 3D surfaces only. For 4 proximity query algorithms *TIN-ESSP-Adapt(HM)*, *TIN-SNP-Adapt(HM)*, *PC-SP-Adapt(HM)* and *HM-SP*, we use $\epsilon = 0$ (resp. $\epsilon > 0$) to denote that we apply them on the original (resp. simplified) surfaces. Since *TIN-ESSP-Adapt(HM)* with $\epsilon > 0$ already means calculating the exact shortest surface path passing on a simplified *TIN*, there is no need to use *TIN-ASSP-Adapt(HM)* on the simplified *TIN* again, i.e., no need to distinguish $\epsilon = 0$ or $\epsilon > 0$ for it. So, we only consider it with $\epsilon > 0$ on the original height map for simplicity. We compare all algorithms in Tables 2 and 3.

Table 2: Comparison of simplification algorithms

Algorithm	Simplification time		Output size	
TIN-SSimplify-Adapt(HM) [40, 43]	$O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$	Large	$O(n)$	Large
TIN-NSimplify-Adapt(HM) [43]	$O(n^2\log n)$	Medium	$O(n)$	Large
PC-Simplify-Adapt(HM) [23, 69]	$O(n^2\log n)$	Medium	$O(n)$	Large
HM-Simplify **(ours)**	$O(n\sqrt[3]{n}\log n)$	**Small**	$O(\frac{n}{\mu})$	**Small**

Table 3: Comparison of proximity query algorithms

Algorithm	Shortest path query time		Error
On the original 3D surfaces			
TIN-ESSP-Adapt(HM) [27, 63, 70]	$O(n^2)$	Large	Small
TIN-ASSP-Adapt(HM) [42, 68]	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	Small
TIN-SNP-Adapt(HM) [43]	$O(n\log n)$	Medium	Medium
PC-SP-Adapt(HM) [69]	$O(n\log n)$	Medium	No error
HM-SP **(ours)**	$O(n\log n)$	**Medium**	**No error**
On the simplified 3D surfaces			
TIN-ESSP-Adapt(HM) [27, 63, 70]	$O(n^2)$	Large	Small
TIN-SNP-Adapt(HM) [43]	$O(n\log n)$	Medium	Medium
PC-SP-Adapt(HM) [69]	$O(n\log n)$	Medium	Small
HM-SP **(ours)**	$O(\frac{n}{\mu}\log\frac{n}{\mu})$	**Small**	**Small**

(2) To solve the existing problem on *Point Clouds* [69], we adapted algorithms on height maps or *TIN*s, by converting the given point clouds to height maps [24, 57] or *TIN*s [35, 59, 69], and append "*-Adapt(PC)*" in algorithm names. Similarly, we have 9 algorithms: (i) *TIN-SSimplify-Adapt(PC)* [40, 43], (ii) *TIN-NSimplify-Adapt(PC)* [43], (iii) *PC-Simplify* [23, 69], (iv) *HM-Simplify-Adapt(PC)*, (v) *TIN-ESSP-Adapt(PC)* [27, 63, 70], (vi) *TIN-ASSP-Adapt(PC)* [42, 68], (vii) *TIN-SNP-Adapt(PC)* [43], (viii) *PC-SP* [69] and (ix) *HM-SP-Adapt(PC)*.

(3) To solve the existing problem on *TIN*s [40, 43], we adapted algorithms on height maps or point clouds, by converting the given *TIN*s to height maps [22, 38] or point clouds [69, 71], and append "*-Adapt(TIN)*" in algorithm names. Similarly, we have 9 algorithms: (i) *TIN-SSimplify* [40, 43], (ii) *TIN-NSimplify* [43], (iii) *PC-Simplify-Adapt(TIN)* [23, 69], (iv) *HM-Simplify-Adapt(TIN)*, (v) *TIN-ESSP* [27, 63, 70], (vi) *TIN-ASSP* [42, 68], (vii) *TIN-SNP* [43], (viii) *PC-SP-Adapt(TIN)* [69] and (ix) *HM-SP-Adapt(TIN)*.

For points (2) and (3), they are *additional* adaptations since we want to see the performance of our algorithms for other problems. The above adaptations involve data conversion. If no data conversion is involved, (1) we can adapt *HM-Simplify* and *HM-SP* to the point cloud, and adapted versions have the same performance of them on the height map since the height map graph and the point cloud graph are the same, and (2) there is no reason to adapt *HM-Simplify* and *HM-SP* to the *TIN* since expensive *TIN* retriangulation is involved in simplification, and the *TIN*'s structure is more complex, which both significantly harm the performance (i.e., adapted versions have the similar performance of *TIN-NSimplify* and *TIN-SNP* on the *TIN*).

5.1.3 *Proximity Queries.*

We conducted 3 queries. (1) Shortest path query: we generated 100 queries by randomly selecting two points on the height map, point cloud or *TIN* as source and destination. We report the average, maximum and minimum results. The experimental result figures' points indicate the average results, and vertical bars represent the maximum and minimum values. (2 & 3) *kNN* and range queries: we randomly selected 1000 points on the height map, point cloud or *TIN* as query objects to perform the proximity query algorithm in Section 4.6.

5.1.4 *Factors and Metrics.*

We studied 5 factors: (1) ϵ, (2) n (dataset size, i.e., the number of cells of a height map, points of a point cloud, or vertices of a *TIN*), (3) d (the maximum pairwise distances among query objects), (4) k (k value in *kNN* query) and (5) r (range value in range query). When not varying $d \in$ [4km, 20km], $k \in [200, 1000]$ and $r \in$ [2km, 10km], we fix d at 10km, k at 500 and r at 5km according to studies [31, 56]. For simplification algorithms, we employed 3 metrics: (1) *preprocessing time* (the data conversion time (if any) plus the simplification time, where the former is 10^6 to 10^9 times smaller than the latter), (2) *number of cells, points or vertices* in the simplified height map, point cloud or *TIN* and (3) *output size*. For proximity query algorithms, we employed 7 metrics: (1) *query time* (the data conversion time (if any) plus the shortest path query time, where the former is 10^4 to 10^6 times smaller than the latter), (2 & 3) *kNN or range query time* (the data conversion time (if any) plus *kNN* or range query time), (4) *memory* (the storage complexity during algorithm execution), (5) *distance error ratio* (the error ratio of the distance calculated by a studied algorithm compared with the ground-truth or optimal distance, see as follows), (6 & 7) *kNN or range query error ratio* (see Section 4.8.2).

There are two sets of experiments in terms of distance error ratio calculation. Before we give the details, we introduce the following. The relative error of the *TIN*'s exact shortest surface distance [34, 41] and the height map's exact shortest distance (computed on the point cloud converted from the given height map) [45, 60] compared with the real shortest distance in the real world (measured in an on-site survey on a real 3D surface by human) are 0.0454 and 0.0613 in average, with variance 0.0015 and 0.0026, and standard deviation 0.0387 and 0.0511, respectively. Both distances are *approximation* of the real shortest distance without a bound guarantee. Based on this, we introduce the two sets of experiments. (1) We regard the *TIN*'s exact shortest surface distance (computed by *TIN-ESSP* with $\epsilon = 0$) as the ground-truth distance when using height maps, point clouds and *TIN*s as input consistently across experiments. Since compared with the real shortest distance, the average error of this distance, i.e., 0.0454, is smaller than that of the height map's exact shortest distance, i.e., 0.0613 (although this distance is not always smaller than the height map's exact shortest distance in Section 3.1, e.g., Euclidean distance is usually smaller than *TIN*'s exact shortest surface distance, but its error is larger), and a *TIN* is a more detailed representation of the underlying 3D surface. (2) For the rigorous formulation of our problem (based on height map only), we regard the height map's exact shortest distance (computed by *HM-SP* with $\epsilon = 0$) as the optimal distance under this particular 3D surface.

5.2 Experimental Results

5.2.1 ***Height maps with ground-truth distance.*** We studied proximity queries on height maps using the ground-truth distance for distance error ratio calculation. We compared all algorithms in Tables 2 and 3 on small-version datasets, and compared all algorithms except *TIN-SSimplify-Adapt(HM)*, *TIN-NSimplify-Adapt(HM)* and *PC-Simplify-Adapt(HM)* on original datasets (due to their large preprocessing time), and except *TIN-ESSP-Adapt(HM)* and *TIN-SNP-Adapt(HM)* on the simplified *TIN*, and *PC-SP-Adapt(HM)* on the simplified point cloud (due to their dependency on the previous three algorithms).

(1) **Baseline comparisons**:

(i) **Effect of** ϵ: In Figures 9 (a) to (g), we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_h*-small* dataset while fixing n at 1k for baseline comparisons. The preprocessing time of *HM-Simplify* is much smaller than three baselines' due to the efficient height map shortest path query and efficient ϵ-approximate simplified height map checking. The number of cells of the simplified height map and output size of *HM-Simplify* are also much smaller than three baselines' due to the novel cell merging technique. The shortest path query time and the *kNN* query time ($O(\frac{nn'}{\mu}\log\frac{n}{\mu})$ in Theorem 4.3) of *HM-SP* on the simplified height map are also small since its simplified height map has a small output size. Although increasing ϵ will slightly increase the experimental distance error ratio of *HM-SP* on the simplified height map, its value is 0.0340, i.e., close to 0. So, increasing ϵ has no impact on the experimental *kNN* and range query error ratios, their values are 0 (since $|\Pi(i, p'_f|T)| = |\Pi(i, p_f|T)|$ in Section 4.8.2), and their results are omitted. Compared with the real shortest distance, since the relative error of the ground-truth distance is 0.0454, the relative error of the shortest distance returned by *HM-SP* on the simplified height map is at most $0.0809 = \max(0.0809(= (1+0.0340)\times(1+0.0454)-1), 0.0779(= 1-(1-0.0340)\times(1-0.0454)))$.

(ii) **Effect of** n **(scalability test)**: In Figures 10 (a) to (e), we tested 5 values of n in {5M, 10M, 15M, 20M, 25M} on LM_h dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-Simplify* (in terms of output size, i.e., 6.8MB) and *HM-SP* on the simplified height map (in terms of range query time, i.e., 310s ≈ 5.1 min, and memory usage, i.e., 310MB) are scalable on extremely large height map with 25M cells. Although the theoretical output size of *HM-Simplify* is only μ times smaller than the size of an original height map, it returns a simplified height map with an experimental size of 6.8MB from an original one with size 600MB and 25M cells, and performing range query on them with 500 objects takes 400s ≈ 6.7 min and 35,200s ≈ 9.8 hours, respectively. When n is smaller, i.e., datasets with looser density or fragmentation (since multi-resolution datasets have the same region), algorithms can run faster.

(iii) **Effect of** d: In Figure 11, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on RM_h dataset while fixing ϵ at 0.25 and n at 0.5M for baseline comparisons. A smaller d reduces *kNN* and range query time, since our proximity query algorithm uses Dijkstra's algorithm once, we can terminate it earlier after visiting all query objects. As d increases, there is no upper bound on the increase in *kNN* query time (since we append the paths computed by Dijkstra's algorithm and the intra-paths as results, we cannot determine the distance correlations among these paths until we perform a linear scan, i.e., we terminate Dijkstra's algorithm based solely on d), but there is an upper bound on the increase in range query time (since we can also terminate Dijkstra's algorithm earlier if the searching distance exceeds r).

(2) **Ablation study for proximity query algorithms (effect of** k **and** r**)**: We considered two variations of *HM-SP* (on the simplified height map), i.e., (i) *HM-SP Large Query Time* (*HM-SP-LQT1*): *HM-SP* using the naive shortest path query algorithm in Section 4.5, but the efficient proximity query algorithm in Section 4.6, and (ii) *HM-SP-LQT2*: *HM-SP* using the efficient shortest path query algorithm, but the naive proximity query algorithm. In Figure 12, we tested 5 values of k in {200, 400, 600, 800, 1000} and 5 values of r in {2km, 4km, 6km, 8km, 10km} both on RM_h dataset while fixing ϵ at 0.25 and n at 0.5M for ablation study. On the simplified height map, *HM-SP* outperforms both *HM-SP-LQT1* and *HM-SP-LQT2*, since we use the efficient algorithm for querying. Due to the two reasons in the previous paragraph, k does not affect *kNN* query time, but a smaller r reduces range query time.

(3) **Ablation study for simplification algorithms**: We considered three variations of *HM-Simplify*, i.e., (i) *HM-Simplify Large output Size* (*HM-Simplify-LS*): *HM-Simplify* using the naive merging technique that only merges two cells in Section 4.2, (ii) *HM-Simplify Large Simplification Time* (*HM-Simplify-LST*): *HM-Simplify* using the naive checking technique that checks whether Inequality 1 is satisfied for all points in Section 4.2 and (iii) *HM-Simplify-DS* (with $\epsilon' = 0.25$). Let *HM-SP-LS*, *HM-SP-LST* and *HM-SP-DS* be the corresponding proximity query algorithms on the simplified height map. In Figure 13, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_h*-small* dataset while fixing n at 0.5M for ablation study. *HM-Simplify* performs the best, showing the effectiveness of our merging and checking techniques, and our light structure compared with the heavy data structure in study [50]. Since *HM-Simplify-DS* has a large simplification time but *HM-SP-DS* on the simplified height map has a small shortest path query time, they are useful when we prioritize the latter time over the former time.

5.2.2 ***Point clouds with ground-truth distance.*** We studied proximity queries on point clouds using the ground-truth distance for distance error ratio calculation. In Figure 14, we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on EP_p*-small* dataset while fixing n at 1k for baseline comparison. *HM-Simplify-Adapt(PC)* and

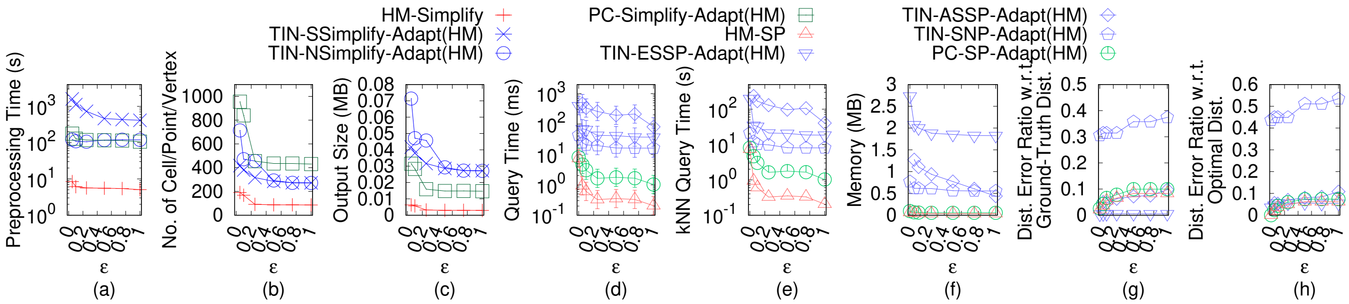


Figure 9: Effect of ϵ on GF_h-*small* height map dataset

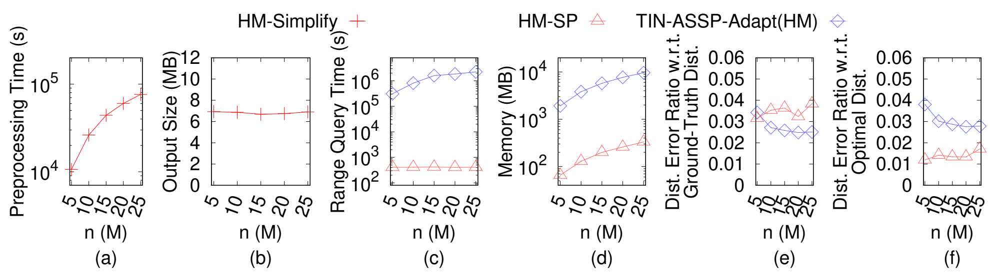


Figure 10: Effect of n on LM_h height map dataset

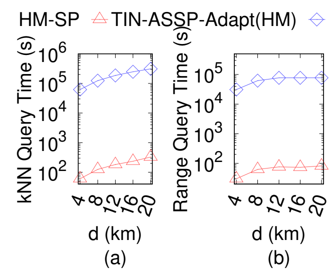


Figure 11: Effect of d on RM_h height map dataset

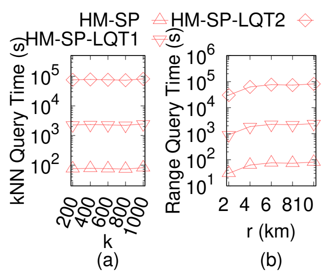


Figure 12: Ablation study for proximity query algorithms (effect of k and r on RM_h height map dataset)

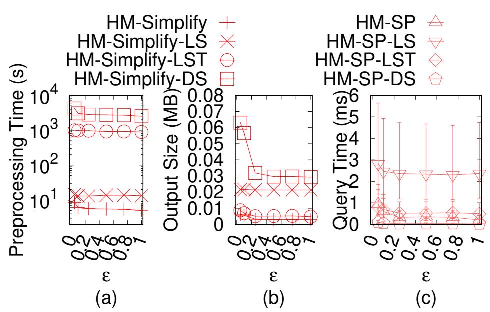


Figure 13: Ablation study for simplification algorithms on BH_h-*small* height map dataset

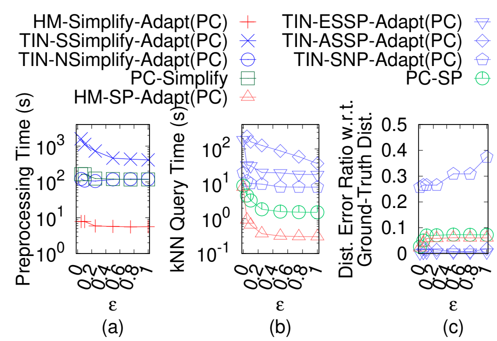


Figure 14: Effect of ϵ on EP_p-*small* point cloud dataset

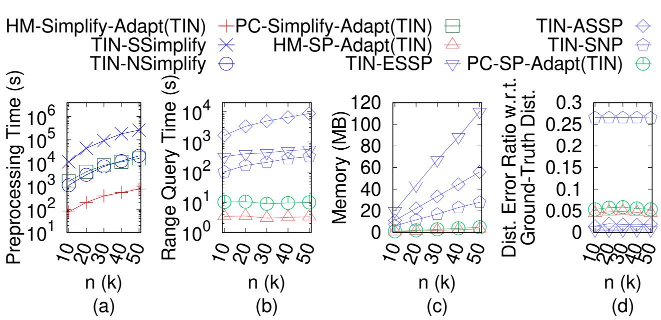


Figure 15: Effect of n on EP_t-*small* TIN dataset

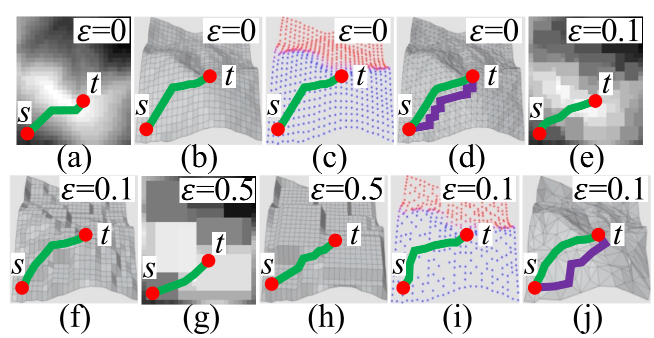


Figure 16: Paths passing on original/simplified height maps (in bird's view), point clouds and *TINs*

HM-SP-Adapt(PC) on the simplified height map still outperforms other baselines.

5.2.3 ***TINs with ground-truth distance.*** We studied proximity queries on *TIN*s using the ground-truth distance for distance error ratio calculation. In Figure 15, we tested 5 values of n in {10k, 20k, 30k, 40k, 50k} on EP_t*-small* dataset while fixing ϵ at 0.1 for baseline comparisons. *HM-Simplify-Adapt(TIN)* still outperforms other baselines. The distance error ratio of *HM-SP-Adapt(TIN)* on the simplified height map is 0.0401, but the distance error ratio of *TIN-SNP* on the simplified *TIN* is 0.2732.

5.2.4 ***Height maps with optimal distance.*** We studied proximity queries on height maps using the optimal distance for distance error ratio calculation. In Figures 9 (h) and 10 (f), the experimental distance error ratio of *HM-SP* on the simplified height map is 0.0186. Compared with the real shortest distance, since the relative error of the optimal distance is 0.0613, the relative error of the shortest distance returned by *HM-SP* on the simplified height map is at most $0.0810 = \max(0.0810(= (1 + 0.0186) \times (1 + 0.0613) - 1), 0.0788(= 1 - (1 - 0.0186) \times (1 - 0.0613))$. The experimental *kNN* and range query error ratios are 0 (since $|\Pi(i, p'_f|H)| = |\Pi(i, p_f|H)|$ in Section 4.8.2, although the theoretical ones are $\frac{2\epsilon}{1-\epsilon}$ in Theorem 4.3).

5.2.5 ***Case study.*** We performed a snowfall evacuation case study [5] at Gates of the Arctic [54] to evacuate tourists to nearby hotels. In Figure 1 (a), due to each hotel's capacity constraints, we find shortest paths from viewpoint a to k-nearest hotels b, c, d, where c and d are k-nearest options when $k = 2$. An individual will be buried in snow in 2.4 hours[3], and the evacuation can be finished in 2.2 hours[4]. Thus, we need to compute shortest paths within 12 min (= 2.4 − 2.2 hours). Our experiments show that for a height map with 50k cells, 10k possible tourist positions and 50 hotels, the simplification time for our algorithm *HM-Simplify*, our adapted algorithm *HM-Simplify-DS*, the best-known adapted point cloud simplification algorithm *PC-Simplify-Adapt(HM)* and the best-known adapted *TIN* simplification algorithm *TIN-SSimplify-Adapt(HM)* are 250s ≈ 4.6 min, 125,000s ≈ 1.5 days, 5,250s ≈ 1.5 hours and 103,000s ≈ 1.2 days, and computing 10 nearest hotels for each tourist position on the simplified 3D surfaces of these algorithms takes 50s, 5s, 250s ≈ 4.2 min and 67,000s ≈ 18.6 hours, respectively. Thus, height map simplification is necessary since 4.6 min + 1.6 min ≤ 12 min. Recall that since only the height map dataset is available for this region, we capture the height map dataset after snowfall, and we have efficient height map simplification and shortest path query algorithms, there is no reason to convert the height map to the point cloud or *TIN*, and perform other slow adapted point cloud or *TIN* algorithms for simplification and shortest path query. In addition, although algorithms with larger (resp. smaller) simplification time but smaller (resp. larger) shortest path query time are better (resp. worse), since we capture the height map dataset after snowfall, the simplification time is considered after snowfall. So, we design *HM-Simplify* to efficiently reduce the simplification time, and significantly reduce its output size so that the shortest path query time of *HM-SP* on the simplified height map is small. But, *HM-Simplify-DS* is not suitable due to the large simplification time.

[3] 2.4 hours = $\frac{10\text{centimeters}\times 24\text{hours}}{1\text{meter}}$, since the maximum snowfall rate (defined as the maximum accumulation of snow depth over a specified time [28, 55]) at Gates of the Arctic is 1 meter per 24 hours [5], and when the snow depth exceeds 10 centimeters, it is difficult to walk and easy to bury in the snow [36].

[4] 2.2 hours = $\frac{11.2\text{km}}{5.1\text{km/h}}$, since the average distance between the viewpoints and hotels at Gates of the Arctic is 11.2km [10], and human's average walking speed is 5.1 km/h [17].

5.2.6 ***Paths visualization.*** In Figure 16, we visualize different paths to verify distance relationships in Section 3.1. (1) Given a height map, the paths in Figures 16 (a) (showing the height map) and (b) (showing the same height map in bird's eye view) computed by our algorithm *HM-SP* on the original height map and the path in Figure 16 (c) computed by the best-known adapted point cloud shortest path query algorithm *PC-SP-Adapt(HM)* on the original point cloud are identical (since $|\Pi(s, t|H)| = |\Pi(s, t|P)|$). The paths in Figures 16 (a) and (b) are similar to the green path in Figure 16 (d) computed by the best-known adapted exact *TIN* shortest surface path query algorithm *TIN-ESSP-Adapt(HM)* on the original *TIN* (since $|\Pi(s, t|H)| \leq \alpha \cdot |\Pi(s, t|T)|$), but computing the former path is much quicker. The distance error ratios of the paths in Figures 16 (a) and (b) are smaller than that of the purple (network) path in Figure 16 (d) computed by the best-known approximate *TIN* shortest network path query algorithm *TIN-ESSP* on the original *TIN* (since $|\Pi(s, t|H)| \leq |\Pi_N(s, t|T)|$). The paths in Figures 16 (a) and (b) are similar to the paths in Figures 16 (e), (f), (g) and (h) computed by our algorithm *HM-SP* on the simplified height maps, but computing the latter four paths are quicker due to the simplified height maps. The path in Figures 16 (e) and (f) are similar to the green path in Figure 16 (i) on a simplified point cloud (generated by the best-known adapted point cloud simplification algorithm *PC-Simplify-Adapt(HM)*) and the green (surface) path in Figure 16 (j) on a simplified *TIN* (generated by the best-known adapted *TIN* simplification algorithm *TIN-SSimplify-Adapt(HM)*). (2 & 3) Given a point cloud or a *TIN*, the path results are the same, since only data conversion is involved in the beginning of the algorithm.

5.2.7 ***Summary.*** On a height map with 50k cells and 10k objects, *HM-Simplify*'s simplification time and output size are 250s ≈ 4.6 min and 0.07MB, which are up to 21 times and 5 times (resp. 412 times and 7 times) better than the best-known adapted point cloud (resp. *TIN*) simplification algorithm *PC-Simplify-Adapt(HM)* (resp. *TIN-SSimplify-Adapt(HM)*). Performing *kNN* query on our simplified height map takes 50s, which is up to 5 times and 1,340 times smaller than on the simplified point cloud and on the simplified *TIN*, respectively. All algorithms perform better on LM_h, RM_h and EP_h datasets, since their 3D surfaces are flatter. Due to the page limit, more experimental figures appear in the appendix.

6 CONCLUSION

We propose an efficient height map simplification algorithm *HM-Simplify*, that outperforms the best-known algorithm concerning the simplification time and output size. We also propose an efficient shortest path algorithm *HM-SP* on the original/simplified height map, and design algorithms for answering *kNN* and range queries on the original/simplified height map. For future work, we can propose new pruning techniques to further reduce the simplification time and output size of *HM-Simplify*.

REFERENCES

[1] 2017. *1.5 Billion the world's most expensive imaging satellite NISAR.* https://syntheticapertureradar.com/nasa-isro-sar-project-nisar/

[2] 2025. *125 Years of Topographic Mapping.* https://www.esri.com/news/arcnews/fall09articles/125-years.html

[3] 2025. *50 Years of Landsat Science.* https://landsat.gsfc.nasa.gov

[4] 2025. *Avalanche.* https://en.wikipedia.org/wiki/Avalanche

[5] 2025. *Climate and average weather year round in Gates of the Arctic National Park.* https://weatherspark.com/y/150336/Average-Weather-in-Gates-of-the-Arctic-National-Park-Alaska-United-States-Year-Round

[6] 2025. *Data Geocomm.* http://data.geocomm.com/

[7] 2025. *The Earth.* https://www.nationsonline.org/oneworld/earth.htm

[8] 2025. *Geocento commercial SAR satellite imagery resolutions 2022, by cost per scene.* https://www.statista.com/statistics/1293899/geocento-commercial-satellite-sar-imagery-resolution-cost-worldwide/

[9] 2025. *Google Earth.* https://earth.google.com/web

[10] 2025. *Google Map.* https://www.google.com/maps

[11] 2025. *Gunnison National Forest.* https://gunnisoncrestedbutte.com/visit/places-to-go/parks-and-outdoors/gunnison-national-forest/

[12] 2025. *The History of Point Cloud Development.* https://www.linkedin.com/pulse/history-point-cloud-development-bimprove/

[13] 2025. *How satellites work.* https://science.howstuffworks.com/satellite10.htm

[14] 2025. *Laramie Mountain.* https://www.britannica.com/place/Laramie-Mountains

[15] 2025. *Metaverse.* https://about.facebook.com/meta

[16] 2025. *Open Digital Elevation Model (OpenDEM).* https://www.opendem.info/

[17] 2025. *Preferred walking speed.* https://en.wikipedia.org/wiki/Preferred_walking_speed

[18] 2025. *Robinson Mountain.* https://www.mountaineers.org/activities/routes-places/robinson-mountain

[19] 2025. *Shuttle Radar Topography Mission (SRTM).* https://www.earthdata.nasa.gov/data/instruments/srtm

[20] 2025. *Transparent pricing for commercial Earth observation imagery.* https://skyfi.com/en/pricing

[21] 2025. *Zoom Earth.* https://zoom.earth/maps/

[22] Segun M Adedapo and Hamdi A Zurqani. 2024. Evaluating the performance of various interpolation techniques on digital elevation models in highly dense forest vegetation environment. *Ecological Informatics (EI)* 81 (2024).

[23] Pankaj K Agarwal and Pavan K Desikan. 1997. An efficient algorithm for terrain simplification. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 139–147.

[24] Mehul Arora, Louis Wiesmann, Xieyuanli Chen, and Cyrill Stachniss. 2021. Mapping the static parts of dynamic scenes from 3D LiDAR point clouds exploiting ground segmentation. In *IEEE European Conference on Mobile Robots (ECMR)*. 1–6.

[25] Mehul Arora, Louis Wiesmann, Xieyuanli Chen, and Cyrill Stachniss. 2023. Static map generation from 3D LiDAR point clouds exploiting ground segmentation. *Robotics and Autonomous Systems (RAS)* 159 (2023), 104287.

[26] Leizhen Cai. 1994. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics (DAM)* 48, 2 (1994), 187–194.

[27] Jindong Chen and Yijie Han. 1990. Shortest Paths on a Polyhedron. In *Symposium on Computational Geometry (SOCG)*. New York, NY, USA, 360–369.

[28] The Conversation. 2025. *How is snowfall measured? A meteorologist explains how volunteers tally up winter storms.* https://theconversation.com/how-is-snowfall-measured-a-meteorologist-explains-how-volunteers-tally-up-winter-storms-175628

[29] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms.* MIT press.

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

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

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

[33] Hristo N Djidjev and Christian Sommer. 2011. Approximate distance queries for weighted polyhedral surfaces. In *Proceedings of the European Symposium on Algorithms*. 579–590.

[34] Weili Fang, Weiya Chen, Peter ED Love, Hanbin Luo, Haiming Zhu, and Jiajing Liu. 2024. A status digital twin approach for physically monitoring over-and-under excavation in large tunnels. *Advanced Engineering Informatics (AEI)* 62 (2024), 102648.

[35] Haoan Feng, Yunting Song, and Leila De Floriani. 2024. Critical Features Tracking on Triangulated Irregular Networks by a Scale-Space Method. In *ACM International Conference on Advances in Geographic Information Systems (GIS)*. 54–66.

[36] Fresh Off The Grid. 2025. *Winter Hiking 101: Everything you need to know about hiking in snow.* https://www.freshofthegrid.com/winter-hiking-101-hiking-in-snow/

[37] Xian-Feng Han, Huixian Cheng, Hang Jiang, Dehong He, and Guoqiang Xiao. 2024. Pcb-randnet: Rethinking random sampling for lidar semantic segmentation in autonomous driving scene. In *IEEE International Conference on Robotics and Automation (ICRA)*. 4435–4441.

[38] Nadine Hobeika, Laurens van Rijssel, Maarit Prusti, Constantijn Dinklo, Denis Giannelli, Balázs Dukai, Arnaud Kok, Rob van Loon, René Nota, and Jantien Stoter. 2024. Automated noise modelling using a triangulated terrain model. *Geo-Spatial Information Science (GSIS)* 27, 6 (2024), 1893–1913.

[39] Hugues Hoppe. 1996. Progressive meshes. In *Conference on Computer Graphics and Interactive Techniques (CGIT)*. 99–108.

[40] Bo Huang, Victor Junqiu Wei, Raymond Chi-Wing Wong, and Bo Tang. 2023. EAR-Oracle: on efficient indexing for distance queries between arbitrary points on terrain surface. In *ACM International Conference on Management of Data (SIGMOD)*, Vol. 1. ACM New York, NY, USA, 1–26.

[41] Md Kamruzzaman, Tanzila Islam, and Smrity Rani Poddar. 2014. Accuracy of Handheld GPS compairing with Total Station in Land Use Survey: A Case Study in RUET Campus. *International Journal of Innovation and Applied Studies (IJIAS)* 7, 1 (2014), 343.

[42] Manohar Kaul, Raymond Chi-Wing Wong, and Christian S Jensen. 2015. New lower and upper bounds for shortest distance queries on terrains. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 9. 168–179.

[43] Manohar Kaul, Raymond Chi-Wing Wong, Bin Yang, and Christian S Jensen. 2013. Finding shortest paths on terrains by killing two birds with one stone. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 7. 73–84.

[44] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. In *Algorithmica*, Vol. 30. 527–562.

[45] Jinheng Li, Feng Shuang, Junjie Huang, Tao Wang, Sijia Hu, Junhao Hu, and Hanbo Zheng. 2023. Safe distance monitoring of live equipment based upon instance segmentation and pseudo-LiDAR. *IEEE Transactions on Power Delivery (TPD)* 38, 4 (2023), 2953–2964.

[46] Shujuan Li, Junsheng Zhou, Baorui Ma, Yu-Shen Liu, and Zhizhong Han. 2024. Learning continuous implicit field with local distance indicator for arbitrary-scale point cloud upsampling. In *AAAI Conference on Artificial Intelligence*, Vol. 38. 3181–3189.

[47] Lian Liu and Raymond Chi-Wing Wong. 2011. Finding shortest path on land surface. In *ACM International Conference on Management of Data (SIGMOD)*. 433–444.

[48] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. *SIAM J. Comput.* 16, 4 (1987), 647–668.

[49] Hoong Kee Ng, Hon Wai Leong, and Ngai Lam Ho. 2004. Efficient algorithm for path-based range query in spatial databases. In *IEEE International Database Engineering and Applications Symposium (IDEAS)*. 334–343.

[50] Eunjin Oh. 2020. Shortest-path queries in geometric networks. In *International Symposium on Algorithms and Computation (ISAAC)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 52–1.

[51] Jagan Sankaranarayanan and Hanan Samet. 2009. Distance oracles for spatial networks. In *IEEE International Conference on Data Engineering (ICDE)*. 652–663.

[52] Jagan Sankaranarayanan, Hanan Samet, and Houman Alborzi. 2009. Path oracles for spatial networks. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 2. 1210–1221.

[53] Olga Schukina, Maksud Abdukarimov, and Albina Valieva. 2024. Creating a 3D terrain model for the territory of the Hasti-Imom ensemble in Tashkent, Uzbekistan. In *E3S Web of Conferences*, Vol. 497. EDP Sciences, 02024.

[54] National Park Service. 2025. *Gates of the Arctic.* https://www.nps.gov/gaar/index.htm

[55] National Weather Service. 2025. *Measuring Snow.* https://www.weather.gov/dvn/snowmeasure

[56] Cyrus Shahabi, Lu-An Tang, and Songhua Xing. 2008. Indexing land surface for efficient knn query. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 1. 1020–1031.

[57] Yixin Sun, Yusen Luo, Qian Zhang, Lizhang Xu, Liying Wang, and Pengpeng Zhang. 2022. Estimation of crop height distribution for mature rice based on a moving surface and 3D point cloud elevation. *Agronomy* 12, 4 (2022), 836.

[58] Farhan Tauheed, Laurynas Biveinis, Thomas Heinis, Felix Schurmann, Henry Markram, and Anastasia Ailamaki. 2012. Accelerating range queries for brain simulations. In *IEEE International Conference on Data Engineering (ICDE)*. 941–952.

[59] Tao Wang, Lianbin Deng, Yuhong Li, and Hao Peng. 2021. Progressive TIN Densification with Connection Analysis for Urban Lidar Data. *Photogrammetric Engineering & Remote Sensing (PERS)* 87, 3 (2021), 205–213.

[60] Xixun Wang, Yoshiki Mizukami, Makoto Tada, and Fumitoshi Matsuno. 2021. Navigation of a mobile robot in a dynamic environment using a point cloud map. *Artificial Life and Robotics (ALR)* 26 (2021), 10–20.

[61] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, and David M. Mount. 2017. Distance oracle on terrain surface. In *ACM International Conference on Management of Data (SIGMOD)*. New York, NY, USA, 1211–1226.

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

[63] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, David M Mount, and Hanan Samet. 2024. On Efficient Shortest Path Computation on Terrain Surface: A Direction-Oriented Approach. *IEEE Transactions on Knowledge & Data*

Engineering (TKDE) 1 (2024), 1–14.

[64] Shi-Qing Xin and Guo-Jin Wang. 2009. Improving Chen and Han's algorithm on the discrete geodesic problem. *ACM Transactions on Graphics (TOG)* 28, 4 (2009), 1–8.

[65] Songhua Xing, Cyrus Shahabi, and Bei Pan. 2009. Continuous monitoring of nearest neighbors on land surface. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 2. 1114–1125.

[66] Da Yan, Zhou Zhao, and Wilfred Ng. 2012. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *ACM International Conference on Information and Knowledge Management (CIKM)*. 942–951.

[67] Yinzhao Yan and Raymond Chi-Wing Wong. 2021. Path Advisor: a multi-functional campus map tool for shortest path. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 14. 2683–2686.

[68] Yinzhao Yan and Raymond Chi-Wing Wong. 2024. Efficient shortest path queries on 3d weighted terrain surfaces for moving objects. In *IEEE International Conference on Mobile Data Management (MDM)*.

[69] Yinzhao Yan and Raymond Chi-Wing Wong. 2024. Proximity queries on point clouds using rapid construction path oracle. In *ACM International Conference on Management of Data (SIGMOD)*, Vol. 2. 1–26.

[70] Yinzhao Yan, Raymond Chi-Wing Wong, and Christian S Jensen. 2024. An Efficiently Updatable Path Oracle for Terrain Surfaces. *IEEE Transactions on Knowledge & Data Engineering (TKDE)* 1 (2024), 1–14.

[71] Hongchuan Yu, Jian J Zhang, and Zheng Jiao. 2014. Geodesics on point clouds. *Mathematical Problems in Engineering (MPE)* 2014 (2014).

[72] Yifei Zhang and Shiyuan Wang. 2023. A robot navigation system in complex terrain based on statistical features of point clouds. *IEEE Transactions on Intelligent Vehicles (TIV)* (2023).

A SUMMARY OF ALL NOTATION

Table 4 shows a summary of all notation.

B COMPARISON OF ALL ALGORITHMS

Tables 5 and 6 show comparisons of all simplification and proximity query algorithms. Recall that we have two variations of *HM-SP* (on the simplified height map) in terms of proximity queries, i.e., *HM-SP-LQT1* (on the simplified height map) and *HM-SP-LQT2* (on the simplified height map). Let *HM-Simplify-LQT1* and *HM-Simplify-LQT2* be the simplification algorithms, so that *HM-SP-LQT1* and *HM-SP-LQT2* are applied on the simplified height map of these two simplification algorithms.

C EMPIRICAL STUDIES

C.1 Experimental Results for Height Maps with Ground-truth Distance

We studied proximity queries on height maps using the ground-truth distance for distance error ratio calculation. We compared algorithms *TIN-SSimplify-Adapt(HM)*, *TIN-NSimplify-Adapt(HM)*, *PC-Simplify-Adapt(HM)*, *HM-Simplify*, *TIN-ESSP-Adapt(HM)* (on the original height map and the simplified *TIN*), *TIN-ASSP-Adapt(HM)*, *TIN-SNP-Adapt(HM)* (on the original height map and the simplified *TIN*), *PC-SP-Adapt(HM)* (on the original and simplified point cloud) and *HM-SP* (on the original and simplified height map) on small-version datasets, and compared all algorithms except *TIN-SSimplify-Adapt(HM)*, *TIN-NSimplify-Adapt(HM)* and *PC-Simplify-Adapt(HM)* on original datasets (due to their excessive simplification time), and except *TIN-ESSP-Adapt(HM)* and *TIN-SNP-Adapt(HM)* on the simplified *TIN*, and *PC-SP-Adapt(HM)* on the simplified point cloud (due to their dependency on the previous two algorithms).

C.1.1 ***Baseline comparisons***. **Effect of ϵ**: In Figure 17, Figure 19, Figure 21, Figure 23 and Figure 25, we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on *GF_h-small*, *LM_h-small*, *RM_h-small*, *BH_h-small* and *EP_h-small* dataset while fixing n at 1k for baseline comparisons. In Figure 28, Figure 31, Figure 34, Figure 37 and Figure 40, we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_h, LM_h, RM_h, BH_h and EP_h dataset while fixing n at 0.5M for baseline comparisons. The preprocessing time of *HM-Simplify* is much smaller than three baselines' due to the efficient height map shortest path query and efficient ϵ-approximate simplified height map checking. The number of cells of the simplified height map and output size of *HM-Simplify* are also much smaller than three baselines' due to the novel cell merging technique. The shortest path query time and the *kNN* query time of *HM-SP* on the simplified height map are also small since its simplified height map has a

Table 4: Summary of all notation

Notation	Meaning
H	The height map with a set of cells
C	The set of cells of H
$N(\cdot)$	The neighbor cells table of H
n	The number of cells of H
P	The point cloud converted from H
T	The *TIN* converted from H
θ	The minimum inner angle of any face in T
G	The height map graph of H and the point cloud graph of P
$G.V/G.E$	The set of vertices and edges of G
$\Pi(s,t\|H)$	The shortest path passing on H between s and t
$\|\Pi(s,t\|H)\|$	$\Pi(s,t\|H)$'s length
$\Pi(s,t\|P)$	The shortest path passing on P between s and t
$\Pi(s,t\|T)$	The shortest surface path passing on T between s and t
$\Pi_N(s,t\|T)$	The shortest network path passing on T between s and t
$\Pi_E(s,t\|T)$	The shortest path passing on the edges of T between s and t where these edges belongs to the faces that $\Pi(s,t\|T)$ passes
$\widetilde{H}$	The simplified height map
$\widetilde{C}$	The set of cells of $\widetilde{H}$
$\widetilde{N}(\cdot)$	The neighbor cells table of $\widetilde{H}$
C_{rema}	The set of remaining cells
C_{dd}	The set of added cells
$\widetilde{G}$	The simplified height graph of $\widetilde{H}$
$\Pi(\widetilde{s},\widetilde{t}\|\widetilde{H})$	The approximate shortest path passing on $\widetilde{H}$ between s and t
ϵ	The error parameter
l_{max}/l_{min}	The longest / shortest edge's length of T
$O(\cdot)$	The containing table
$O^{-1}(\cdot)$	The belonging table
$\widehat{C}$	The set of adjacent cells that we need to merge in each simplification iteration
c_{add}	The added cell formed by merging each cell $\widehat{C}$
$\widetilde{c}$	The estimated cell of c
$\Pi_1(p,q\|\widetilde{H})$	The intra-path passing on $\widetilde{H}$ between c and q
$\Pi_2(p,q\|\widetilde{H})$	The inter-path passing on $\widetilde{H}$ between c and q
$A(c_{add})$	The set of adjacent added cells of c_{add}
$G_{\widetilde{H}}$	The data structure from study [50] used in algorithm *HM-Simplify*

Table 5: Comparison of all simplification algorithms

Algorithm	Simplification time		Output size	
TIN-SSimplify-Adapt(HM) [40, 43]	$O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$	Large	$O(n)$	Large
TIN-NSimplify-Adapt(HM) [43]	$O(n^2\log n)$	Medium	$O(n)$	Large
PC-Simplify-Adapt(HM) [23, 69]	$O(n^2\log n)$	Medium	$O(n)$	Large
HM-Simplify-LQT1	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\mu})$	Small
HM-Simplify-LQT2	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\mu})$	Small
HM-Simplify-LS	$O(n\sqrt[3]{n}\log n)$	Small	$O(n)$	Large
HM-Simplify-LST	$O(n^2\sqrt[3]{n}\log n)$	Large	$O(\frac{n}{\mu})$	Small
HM-Simplify-DS [50]	$O(n\sqrt[3]{n}\log n)+\frac{n^2}{\mu^2}\log^2\frac{n}{\mu}$	Large	$O(\frac{n}{\mu}\log\frac{n}{\mu})$	Medium
HM-Simplify **(ours)**	$\boldsymbol{O(n\sqrt[3]{n}\log n)}$	**Small**	$\boldsymbol{O(\frac{n}{\mu})}$	**Small**

Table 6: Comparison of all proximity query algorithms

Algorithm	Shortest path query time		*k*NN and range query time		Error
On the original 3D surfaces					
TIN-ESSP-Adapt(HM) [27, 63, 70]	$O(n^2)$	Large	$O(n^2)$	Large	Small
TIN-ASSP-Adapt(HM) [42, 68]	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	Small
TIN-SNP-Adapt(HM) [43]	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Medium
PC-SP-Adapt(HM) [69]	$O(n\log n)$	Medium	$O(n\log n)$	Medium	No error
HM-SP **(ours)**	$\boldsymbol{O(n\log n)}$	**Medium**	$\boldsymbol{O(n\log n)}$	**Medium**	**No error**
On the simplified 3D surfaces					
TIN-ESSP-Adapt(HM) [27, 63, 70]	$O(n^2)$	Large	$O(n^2)$	Large	Small
TIN-SNP-Adapt(HM) [43]	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Medium
PC-SP-Adapt(HM) [69]	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Small
HM-SP-LQT1	$O(\frac{n^2}{\log n}\log\frac{n}{\mu})$	Medium	$O(\frac{n^2}{\log n}\log\frac{n}{\mu})$	Medium	Small
HM-SP-LQT2	$O(\frac{n}{\mu}\log\frac{n}{\mu})$	Small	$O(\frac{nn'}{\log n}\log\frac{n}{\mu})$	Medium	Small
HM-SP-LS	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Small
HM-SP-LST	$O(\frac{n}{\mu}\log\frac{n}{\mu})$	Small	$O(\frac{n}{\mu}\log\frac{n}{\mu})$	Small	Small
HM-SP-DS [50]	$O(1)$	Small	$O(n')$	Small	Large
HM-SP **(ours)**	$\boldsymbol{O(\frac{n}{\mu}\log\frac{n}{\mu})}$	**Small**	$\boldsymbol{O(\frac{n}{\mu}\log\frac{n}{\mu})}$	**Small**	**Small**

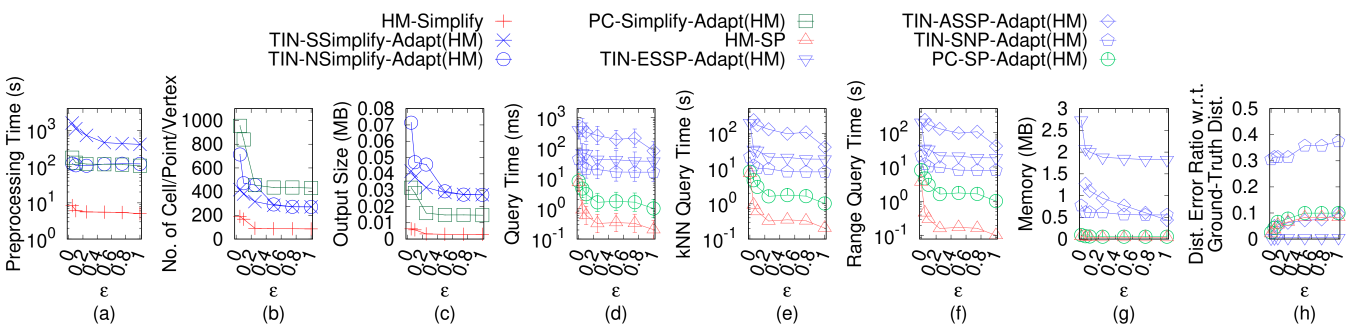


Figure 17: Effect of ϵ on *GF_h-small* height map dataset with ground-truth distance in distance error ratio calculation

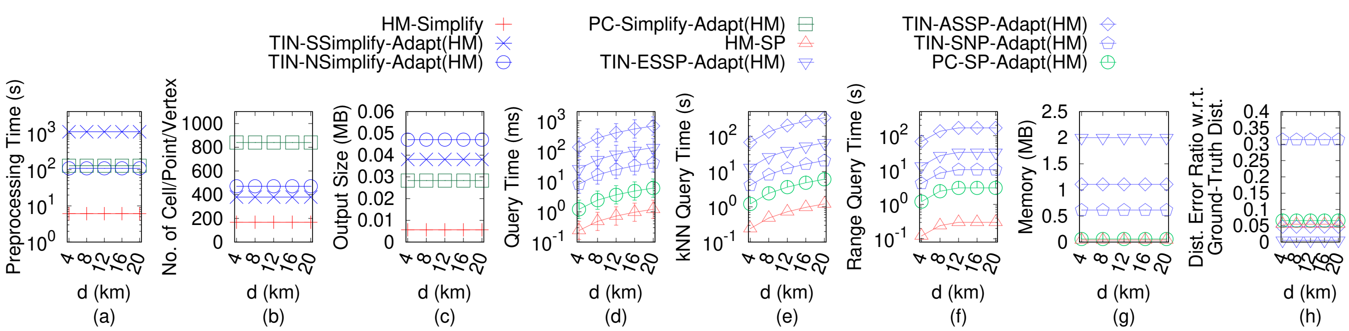


Figure 18: Effect of d on *GF_h-small* height map dataset with ground-truth distance in distance error ratio calculation

small output size. Although increasing ϵ will slightly increase the experimental distance error ratio of *HM-SP* on the simplified height

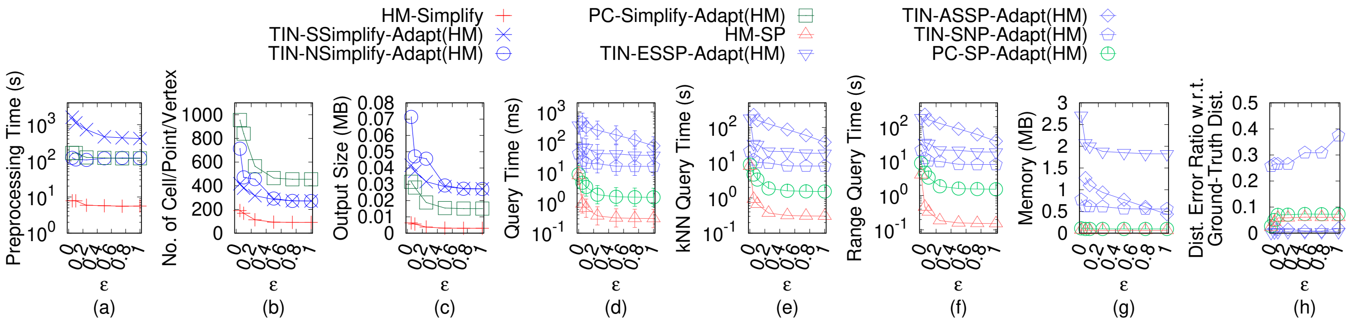


Figure 19: Effect of ϵ on LM_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

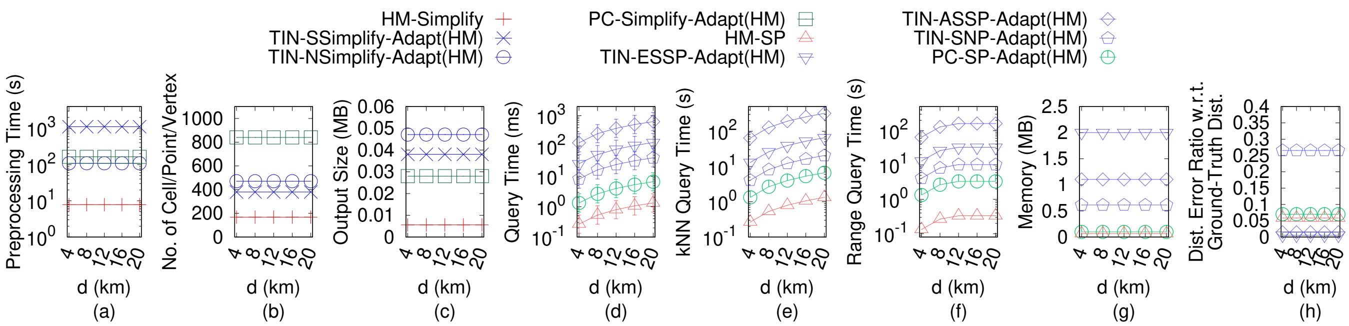


Figure 20: Effect of d on LM_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

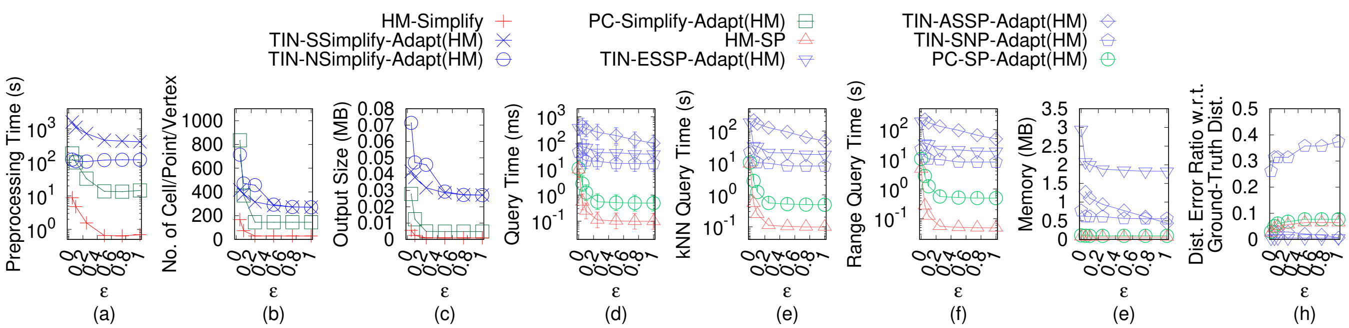


Figure 21: Effect of ϵ on RM_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

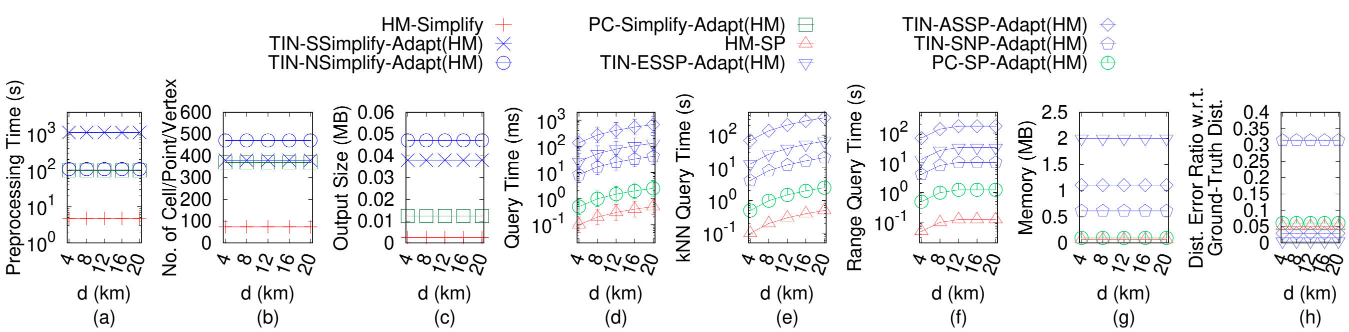


Figure 22: Effect of d on RM_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

map, its value is 0.0340, i.e., close to 0. So, increasing ϵ has no impact on the experimental *kNN* and range query error ratios, their values

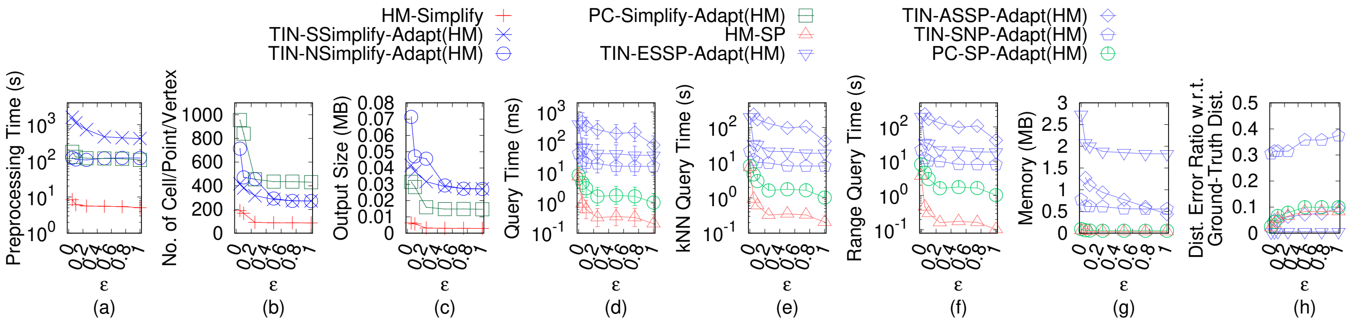


Figure 23: Effect of ϵ on BH_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

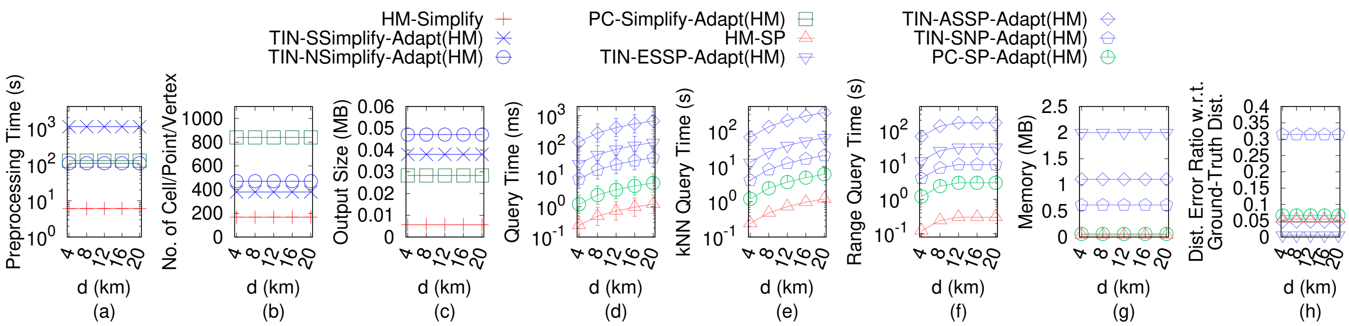


Figure 24: Effect of d on BH_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

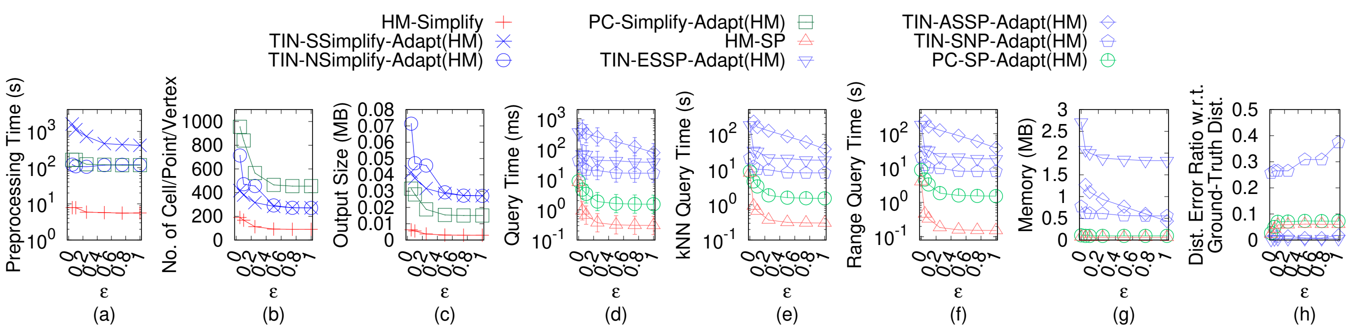


Figure 25: Effect of ϵ on EP_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

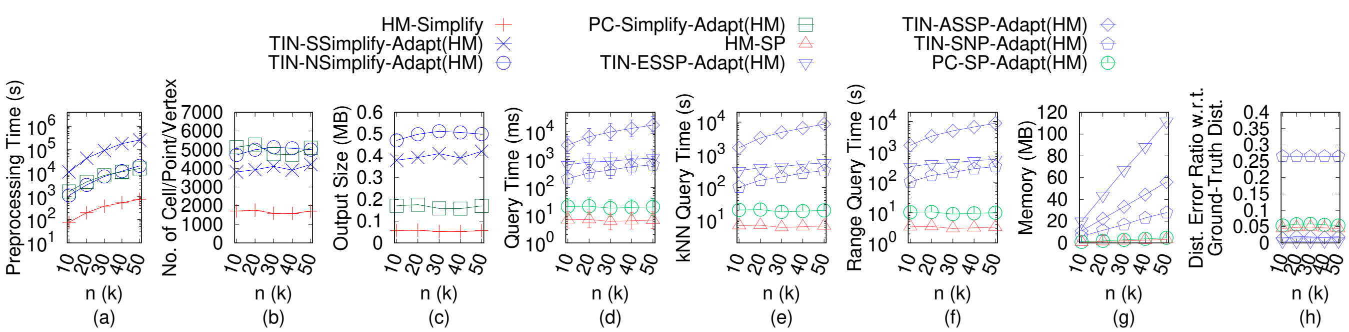


Figure 26: Effect of n on EP_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

are 0, and their results are omitted. Compared with the real shortest distance, since the relative error of the ground-truth distance is

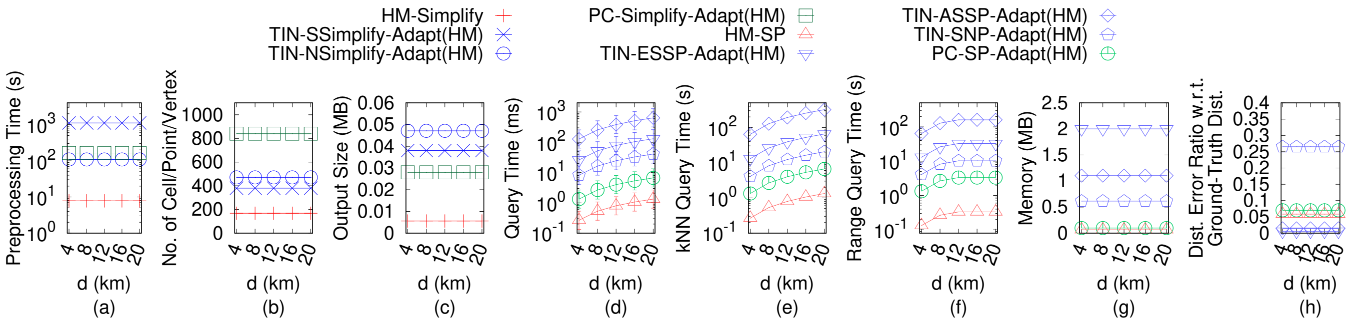


Figure 27: Effect of d on EP_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

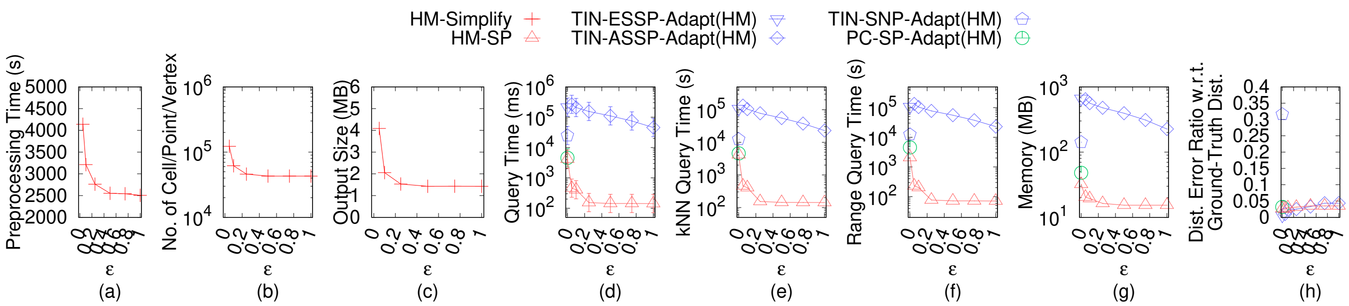


Figure 28: Effect of ϵ on GF_h height map dataset with ground-truth distance in distance error ratio calculation

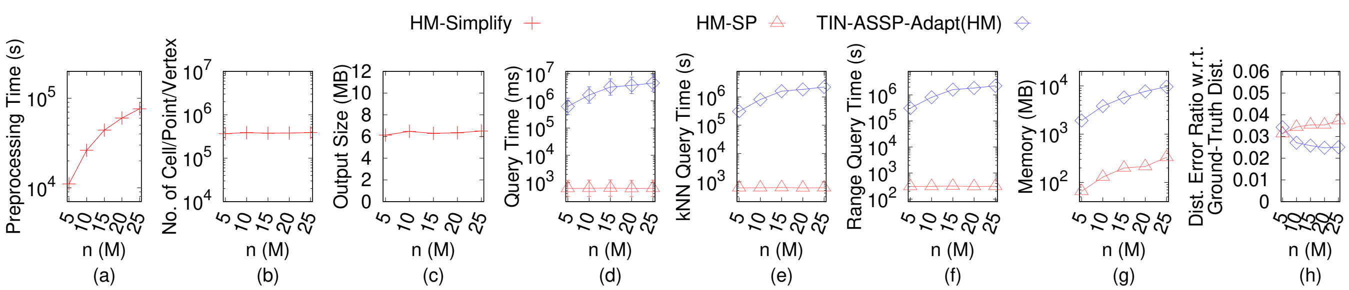


Figure 29: Effect of n on GF_h height map dataset with ground-truth distance in distance error ratio calculation

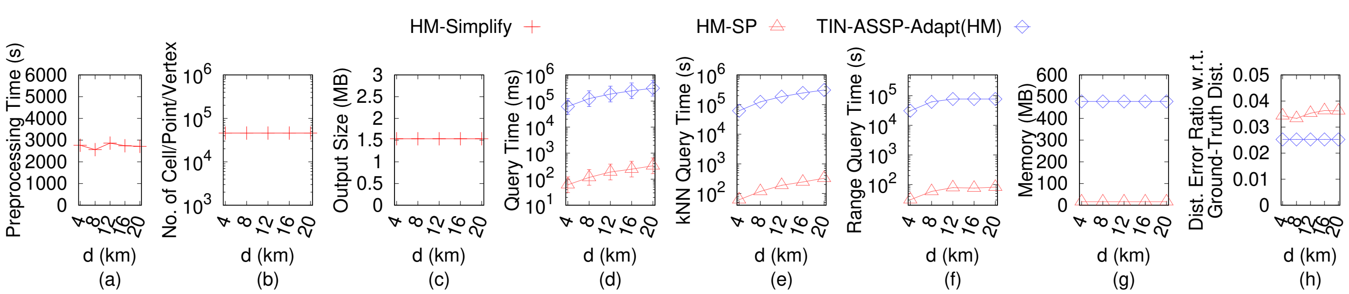


Figure 30: Effect of d on GF_h height map dataset with ground-truth distance in distance error ratio calculation

0.0454, the relative error of the shortest distance returned by *HM-SP* on the simplified height map is at most $0.0809 = \max(0.0809(= (1+0.0340)\times(1+0.0454)-1), 0.0779(= 1-(1-0.0340)\times(1-0.0454)))$.

Effect of n (scalability test): In Figure 26, we tested 5 values of n in {10k, 20k, 30k, 40k, 50k} on EP_h-*small* dataset while fixing ϵ at 0.1 for baseline comparisons. In Figure 29, Figure 32, Figure 35, Figure 38 and Figure 41, we tested 5 values of n in {5M, 10M, 15M, 20M, 25M} on GF_h, LM_h, RM_h, BH_h and EP_h dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-Simplify* (in terms of output size, i.e., 6.8MB) and *HM-SP* on the simplified height map (in terms

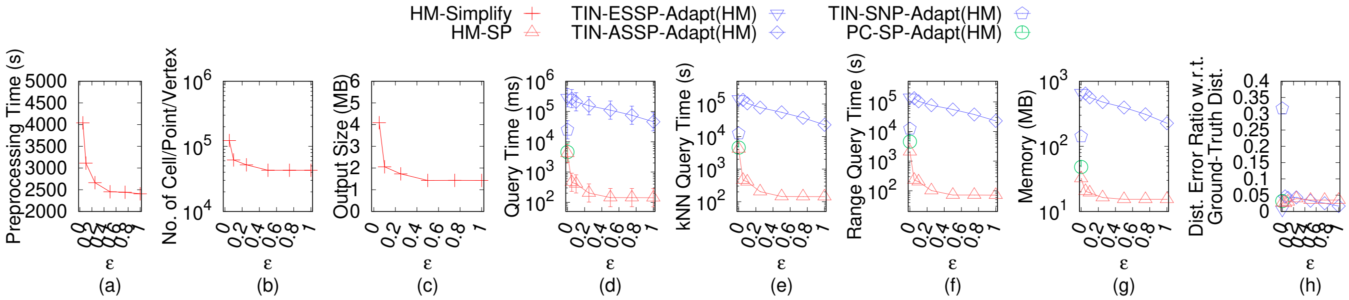


Figure 31: Effect of ϵ on LM_h height map dataset with ground-truth distance in distance error ratio calculation

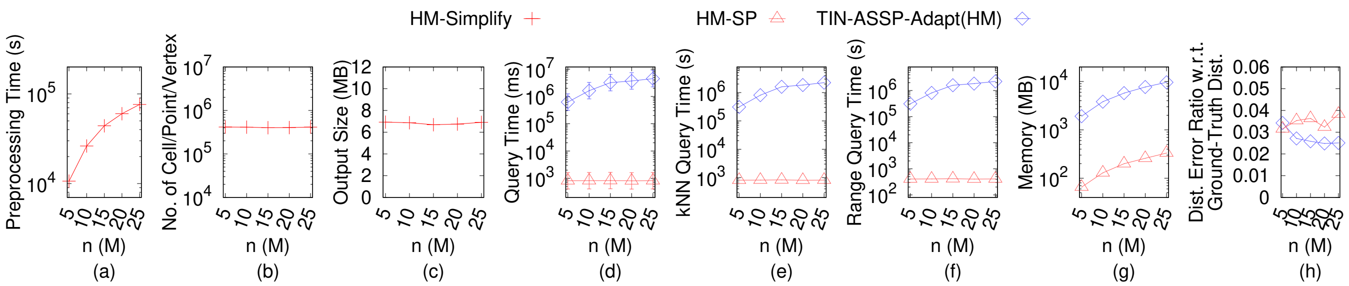


Figure 32: Effect of n on LM_h height map dataset with ground-truth distance in distance error ratio calculation

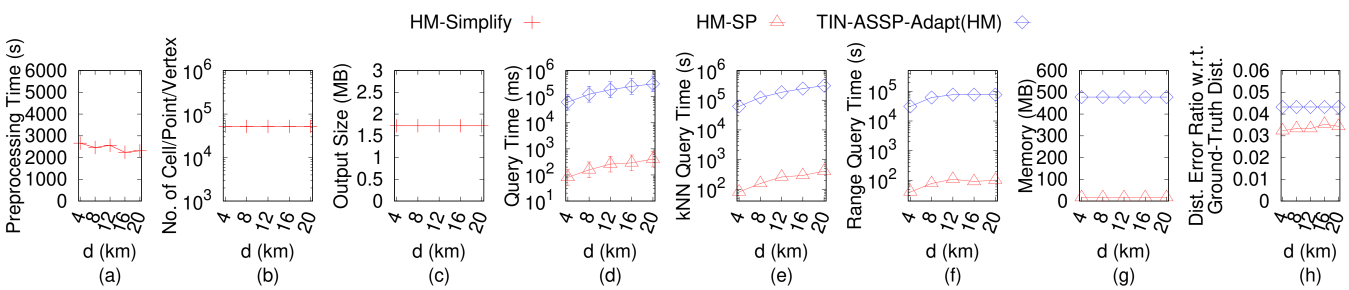


Figure 33: Effect of d on LM_h height map dataset with ground-truth distance in distance error ratio calculation

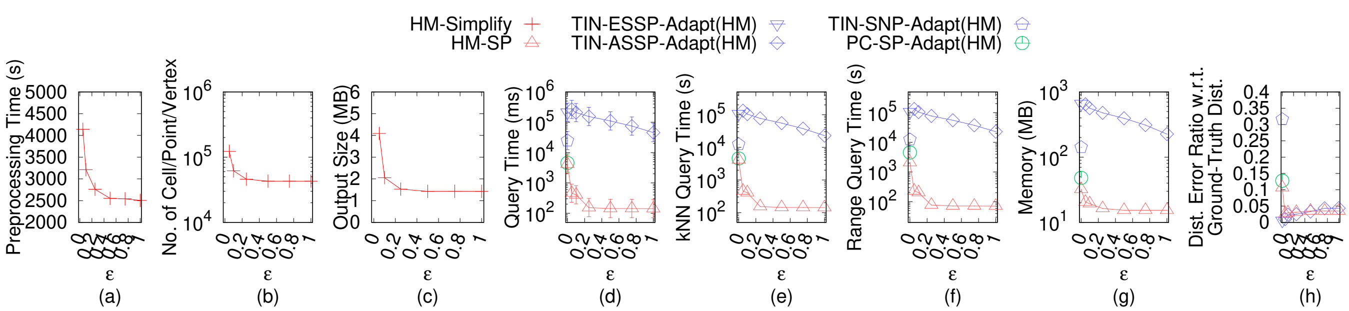


Figure 34: Effect of ϵ on RM_h height map dataset with ground-truth distance in distance error ratio calculation

of range query time, i.e., 310s ≈ 5.1 min, and memory usage, i.e., 310MB) are scalable on extremely large height map with 25M cells. Although the theoretical output size of *HM-Simplify* is only μ times smaller than the size of an original height map, it returns a simplified height map with an experimental size of 6.8MB from an original one with size 600MB and 25M cells, and performing range query on them with 500 objects takes 400s ≈ 6.7 min and 35,200s ≈ 9.8 hours, respectively. When n is smaller, i.e., datasets with looser density or fragmentation (since multi-resolution datasets have the same region), algorithms can run faster.

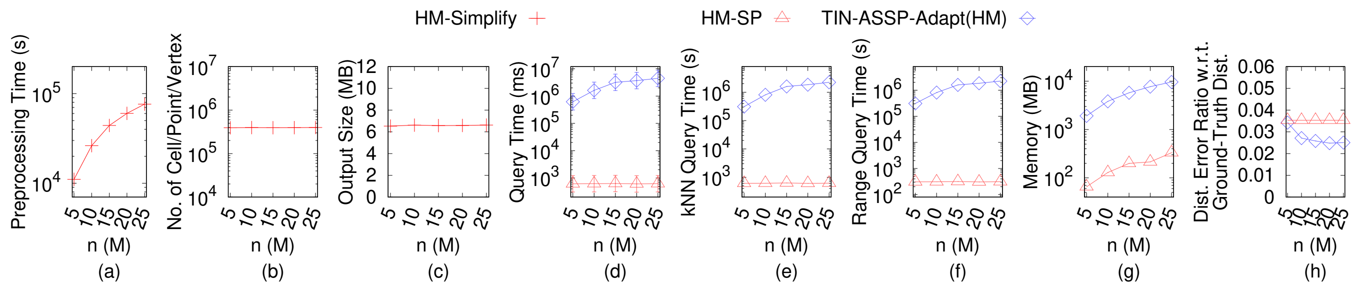


Figure 35: Effect of n on RM_h height map dataset with ground-truth distance in distance error ratio calculation

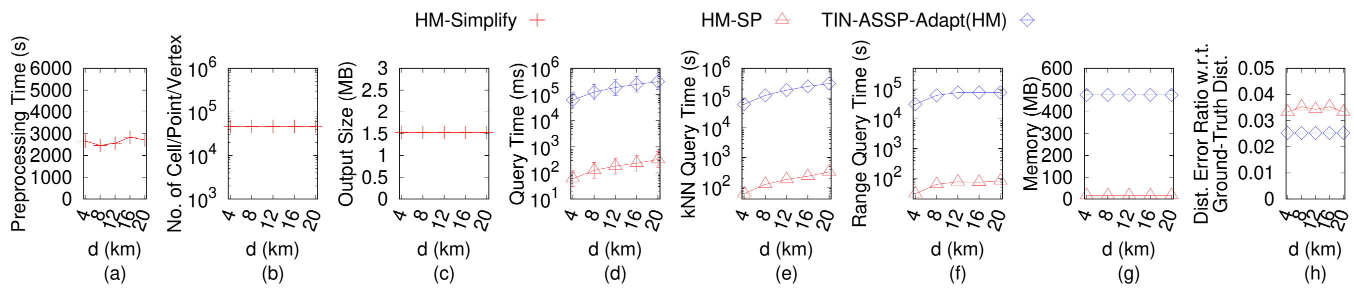


Figure 36: Effect of d on RM_h height map dataset with ground-truth distance in distance error ratio calculation

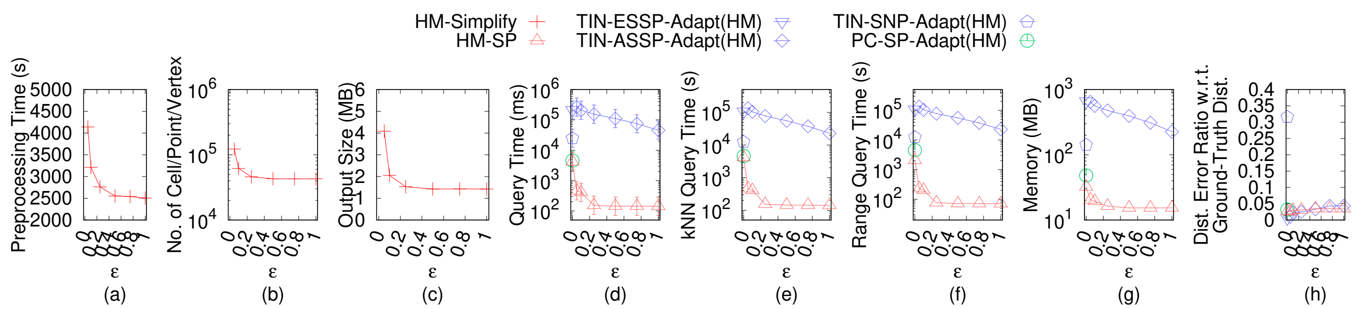


Figure 37: Effect of ϵ on BH_h height map dataset with ground-truth distance in distance error ratio calculation

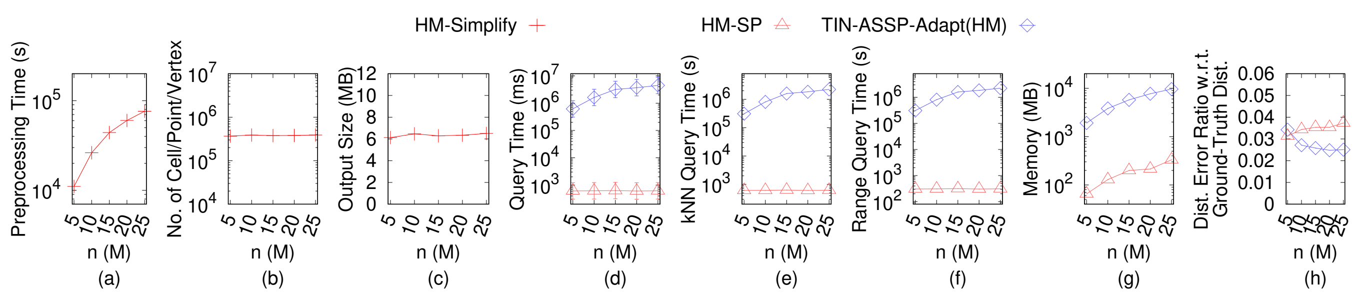


Figure 38: Effect of n on BH_h height map dataset with ground-truth distance in distance error ratio calculation

Effect of d: In Figure 18, Figure 20, Figure 22, Figure 24 and Figure 27, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on GF_h*-small*, LM_h*-small*, RM_h*-small*, BH_h*-small* and EP_h*-small* dataset while fixing ϵ at 0.1 and n at 1k for baseline comparisons. In Figure 30, Figure 33, Figure 36, Figure 39 and Figure 42, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on GF_h, LM_h, RM_h, BH_h and EP_h dataset while fixing ϵ at 0.25 and n at 0.5M for baseline comparisons. A smaller d reduces *kNN* and range query time, since our proximity query algorithm uses Dijkstra's algorithm once, we can terminate it earlier after visiting all query objects. As d increases, there is no upper bound on the increase in *kNN* query time (since we append the paths computed by Dijkstra's algorithm

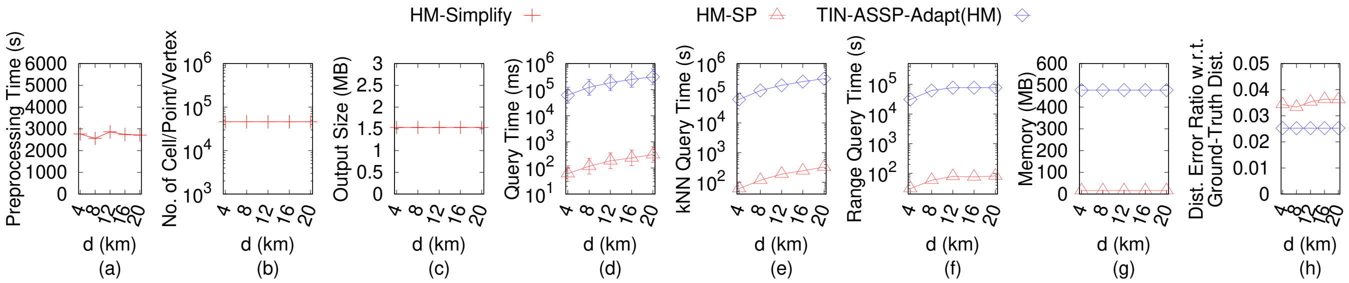


Figure 39: Effect of d on BH_h height map dataset with ground-truth distance in distance error ratio calculation

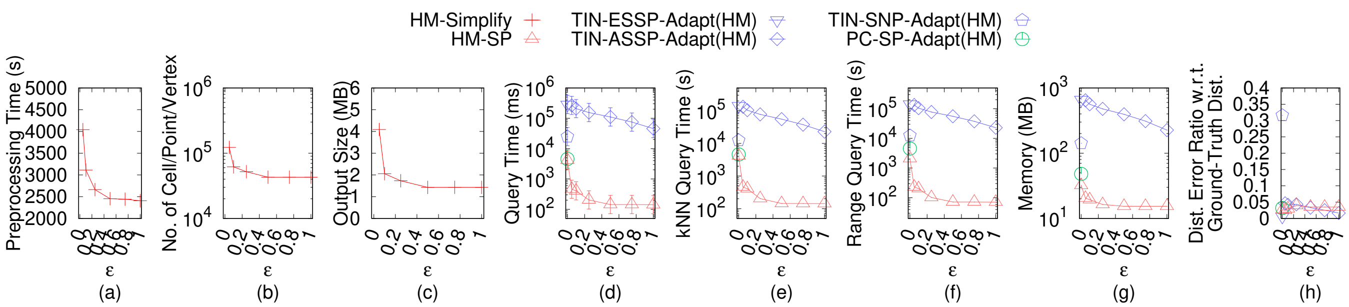


Figure 40: Effect of ϵ on EP_h height map dataset with ground-truth distance in distance error ratio calculation

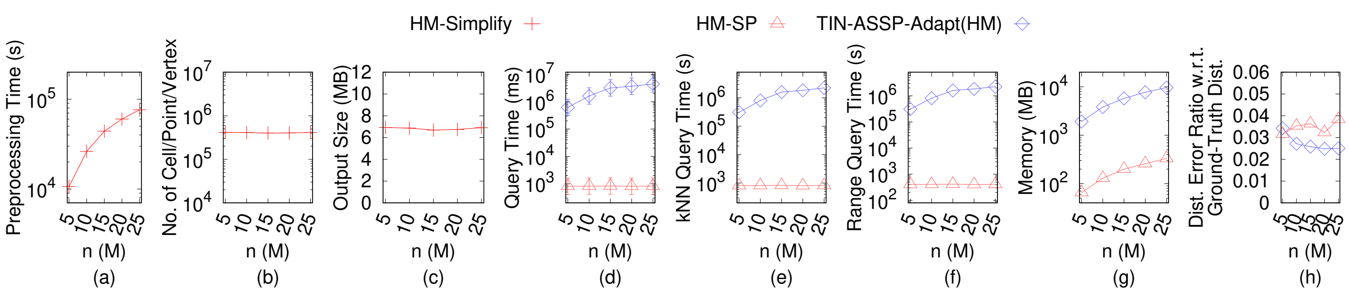


Figure 41: Effect of n on EP_h height map dataset with ground-truth distance in distance error ratio calculation

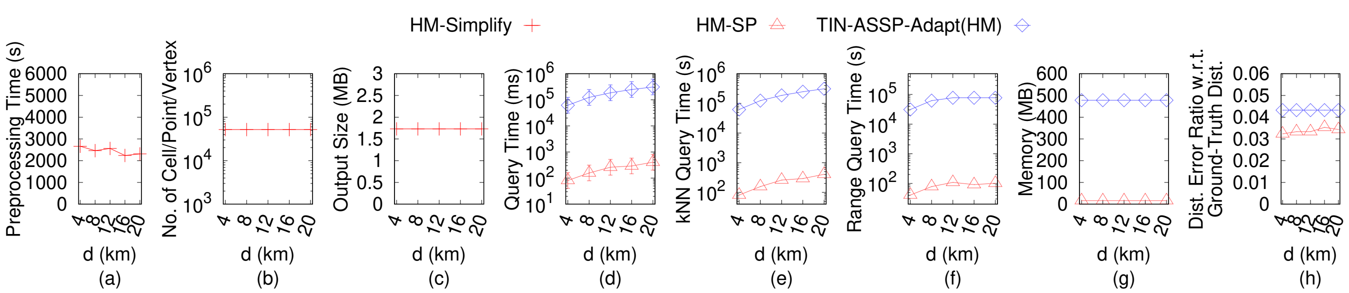


Figure 42: Effect of d on EP_h height map dataset with ground-truth distance in distance error ratio calculation

and the intra-paths as results, we cannot determine the distance correlations among these paths until we perform a linear scan, i.e., we terminate Dijkstra's algorithm based solely on d), but there is an upper bound on the increase in range query time (since we can also terminate Dijkstra's algorithm earlier if the searching distance exceeds r).

C.1.2 ***Ablation study for proximity query algorithms.*** **Effect of k and r**: In Figure 43, Figure 45, Figure 47, Figure 49 and Figure 51, we tested 5 values of k in $\{200, 400, 600, 800, 1000\}$ on GF_h, LM_h, RM_h, BH_h and EP_h dataset while fixing ϵ at 0.25 and n at 0.5M for ablation study. In Figure 44, Figure 46, Figure 48, Figure 50 and Figure 52, we tested 5 values of r in $\{2\text{km}, 4\text{km}, 6\text{km}, 8\text{km}, 10\text{km}\}$

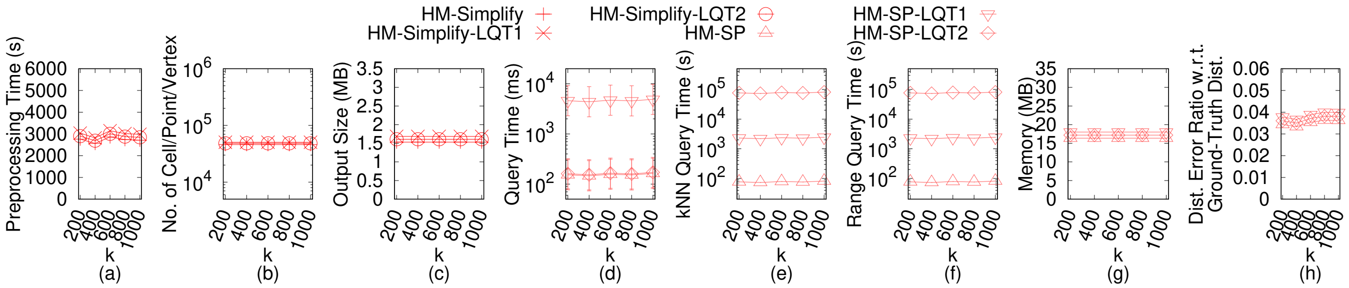


Figure 43: Ablation study for proximity query algorithms (effect of k on GF_h height map dataset) with ground-truth distance in distance error ratio calculation

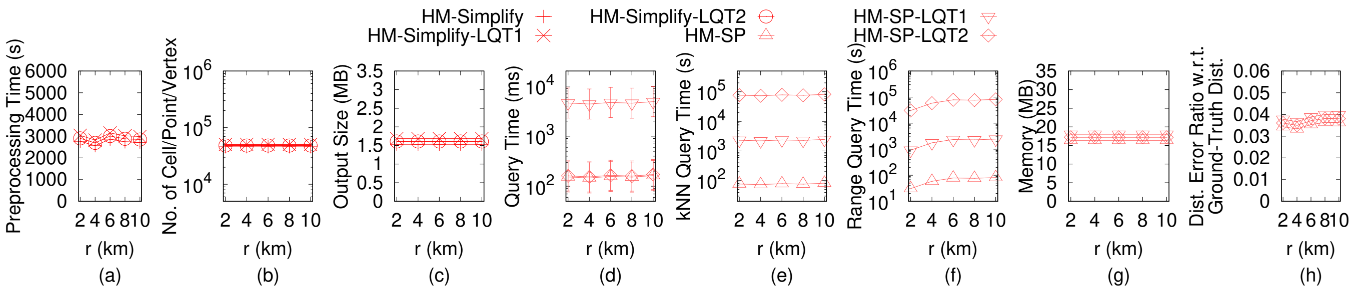


Figure 44: Ablation study for proximity query algorithms (effect of r on GF_h height map dataset) with ground-truth distance in distance error ratio calculation

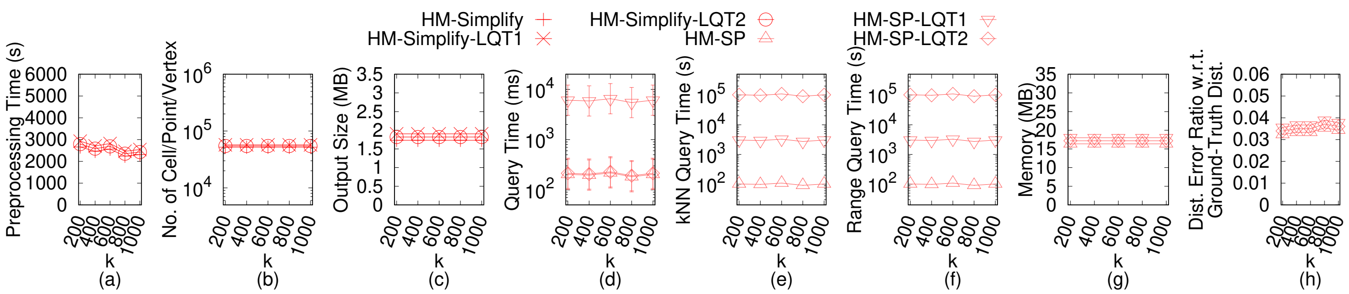


Figure 45: Ablation study for proximity query algorithms (effect of k on LM_h height map dataset) with ground-truth distance in distance error ratio calculation

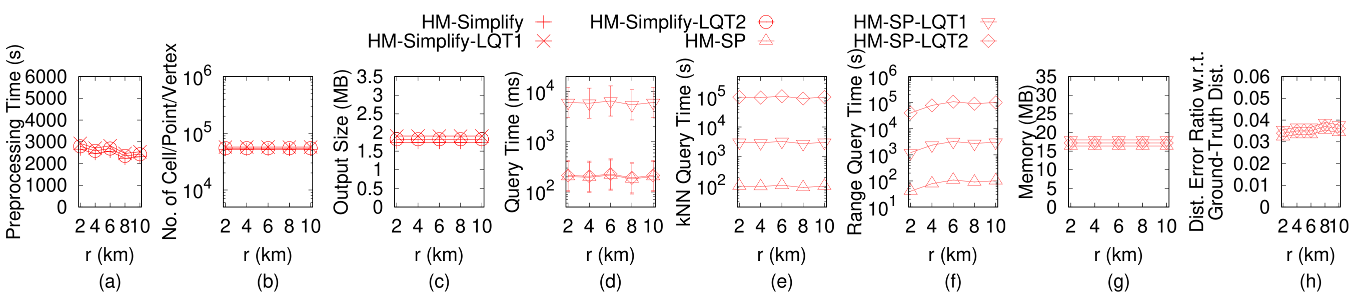


Figure 46: Ablation study for proximity query algorithms (effect of r on LM_h height map dataset) with ground-truth distance in distance error ratio calculation

on GF_h, LM_h, RM_h, BH_h and EP_h dataset while fixing ϵ at 0.25 and n at 0.5M for ablation study for proximity query algorithms. On the simplified height map, *HM-SP* outperforms both *HM-SP-LQT1* and *HM-SP-LQT2*, since we use the efficient algorithm for querying. k does not affect *kNN* query time, since we append the paths computed by Dijkstra's algorithm and the intra-paths as the path

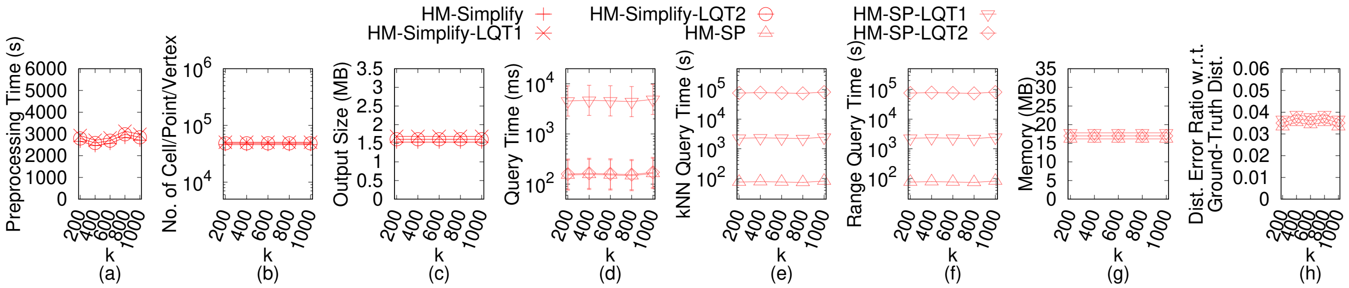


Figure 47: Ablation study for proximity query algorithms (effect of k on RM_h height map dataset) with ground-truth distance in distance error ratio calculation

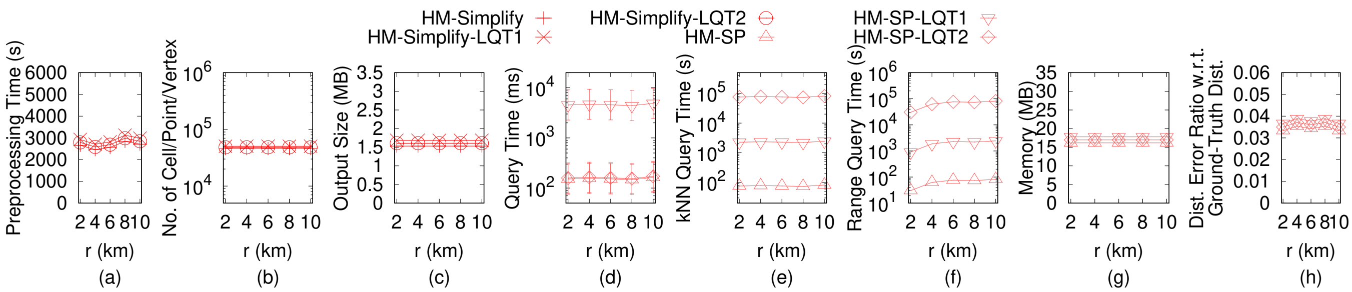


Figure 48: Ablation study for proximity query algorithms (effect of r on RM_h height map dataset) with ground-truth distance in distance error ratio calculation

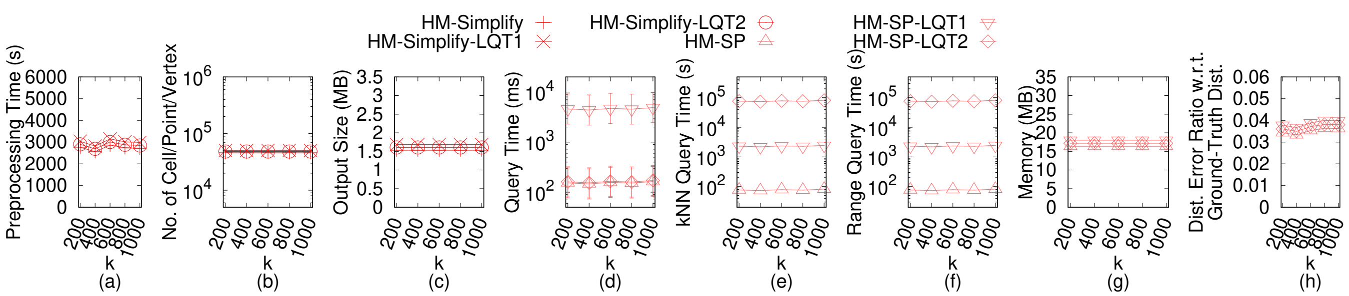


Figure 49: Ablation study for proximity query algorithms (effect of k on BH_h height map dataset) with ground-truth distance in distance error ratio calculation

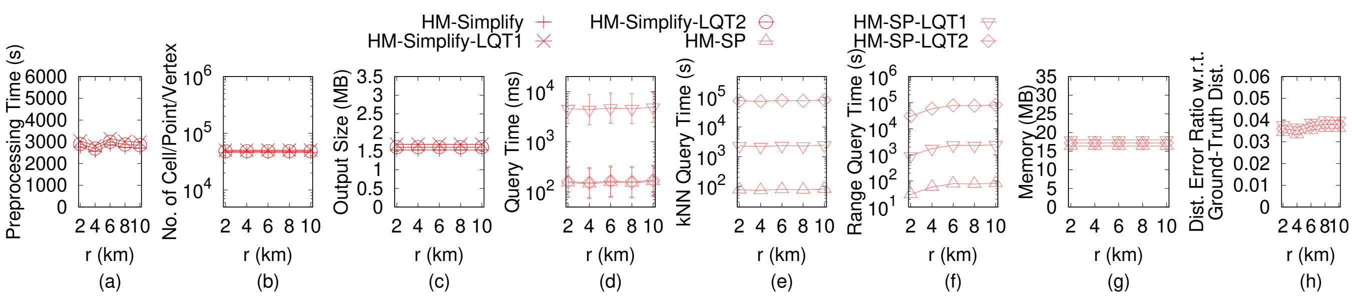


Figure 50: Ablation study for proximity query algorithms (effect of r on BH_h height map dataset) with ground-truth distance in distance error ratio calculation

results, and we do not know the distance correlations among these paths before we perform a linear scan on them. But, a smaller r reduces range query time, since we can terminate Dijkstra's algorithm earlier when the searching distance is larger than r.

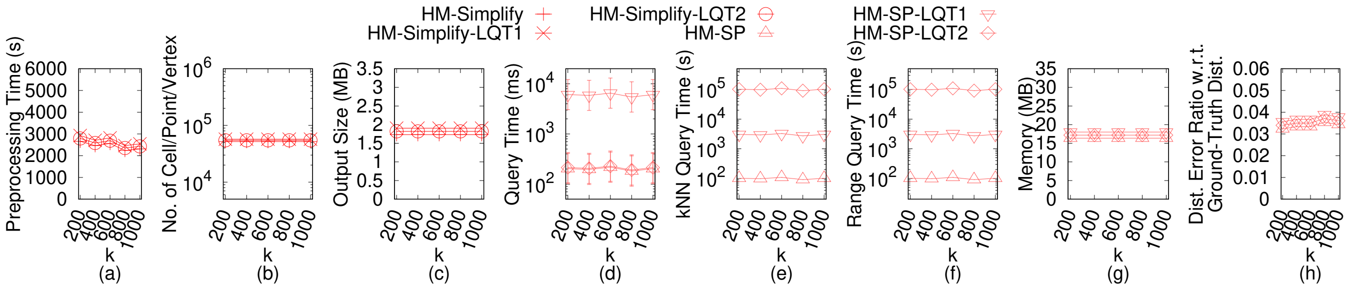


Figure 51: Ablation study for proximity query algorithms (effect of k on EP_h height map dataset) with ground-truth distance in distance error ratio calculation

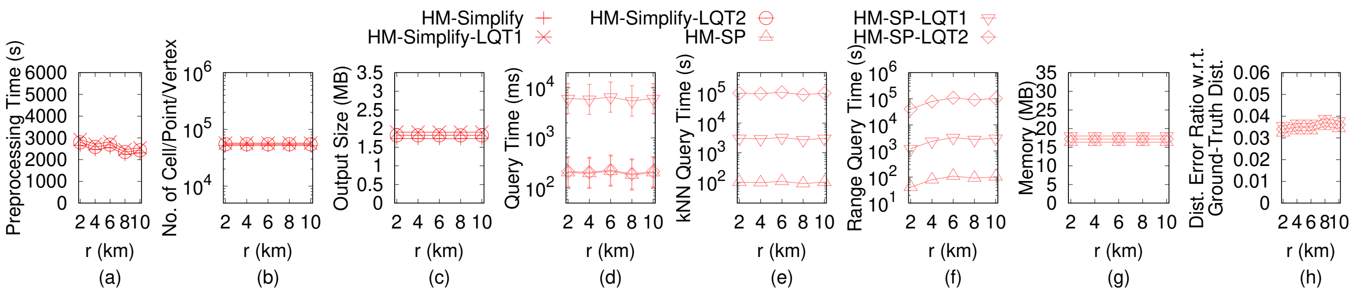


Figure 52: Ablation study for proximity query algorithms (effect of r on EP_h height map dataset) with ground-truth distance in distance error ratio calculation

C.1.3 ***Ablation study for simplification algorithms.*** In Figure 53, Figure 54, Figure 55, Figure 56 and Figure 57, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_h*-small*, LM_h*-small*, RM_h*-small*, BH_h*-small* and EP_h*-small* dataset while fixing n at 0.5M for ablation study. *HM-Simplify* performs the best, showing the effectiveness of our merging and checking techniques, and our light structure compared with the heavy data structure in study [50]. Since *HM-Simplify-DS* has a large simplification time but *HM-SP-DS* on the simplified height map has a small shortest path query time, they are useful when we prioritize the shortest path query time over simplification time.

C.2 Experimental Results for Point Clouds with Ground-truth Distance

We studied proximity queries on point clouds using the ground-truth distance for distance error ratio calculation. We compared algorithms *TIN-SSimplify-Adapt(PC)*, *TIN-NSimplify-Adapt(PC)*, *PC-Simplify*, *HM-Simplify-Adapt(PC)*, *TIN-ESSP-Adapt(PC)* (on the original point cloud and the simplified *TIN*), *TIN-ASSP-Adapt(PC)*, *TIN-SNP-Adapt(PC)* (on the original point cloud and the simplified *TIN*), *PC-SP* (on the original and simplified point cloud) and *HM-SP-Adapt(PC)* (on the original point cloud and the simplified height map) on small-version datasets, and compared all algorithms except *TIN-SSimplify-Adapt(PC)*, *TIN-NSimplify-Adapt(PC)* and *PC-Simplify* on original datasets (due to their excessive simplification time), and except *TIN-ESSP-Adapt(PC)* and *TIN-SNP-Adapt(PC)* on the simplified *TIN*, and *PC-SP* on the simplified point cloud (due to their dependency on the previous three algorithms).

Effect of ϵ: In Figure 58, Figure 60, Figure 62, Figure 64 and Figure 66, we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_p*-small*, LM_p*-small*, RM_p*-small*, BH_p*-small* and EP_p*-small* dataset while fixing n at 1k for baseline comparisons. In Figure 69, Figure 72, Figure 75, Figure 78 and Figure 81, we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_p, LM_p, RM_p, BH_p and EP_p dataset while fixing n at 0.5M for baseline comparisons. The preprocessing time, number of cells of the simplified height map and output size of *HM-Simplify-Adapt(PC)* are much smaller than three baselines'. The proximity queries time of *HM-SP-Adapt(PC)* on the simplified height map are also small since its simplified height map has a small output size.

Effect of n (scalability test): In Figure 67, we tested 5 values of n in {10k, 20k, 30k, 40k, 50k} on EP_p*-small* dataset while fixing ϵ at 0.1 for baseline comparisons. In Figure 70, Figure 73, Figure 76, Figure 79 and Figure 82, we tested 5 values of n in {5M, 10M, 15M, 20M, 25M} on GF_p, LM_p, RM_p, BH_p and EP_p dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-Simplify-Adapt(PC)* outperforms all the remaining simplification algorithms and *HM-SP-Adapt(PC)* on the simplified height map outperforms all the remaining proximity query algorithms.

Effect of d: In Figure 59, Figure 61, Figure 63, Figure 65 and Figure 68, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on GF_p*-small*, LM_p*-small*, RM_p*-small*, BH_p*-small* and EP_p*-small* dataset while fixing ϵ at 0.1 and n at 1k for baseline comparisons. In Figure 71, Figure 74, Figure 77, Figure 80 and Figure 83, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on GF_p, LM_p, RM_p, BH_p and EP_p dataset while fixing ϵ at 0.25 and n at 0.5M for baseline comparisons. A smaller d reduces *kNN* and range query

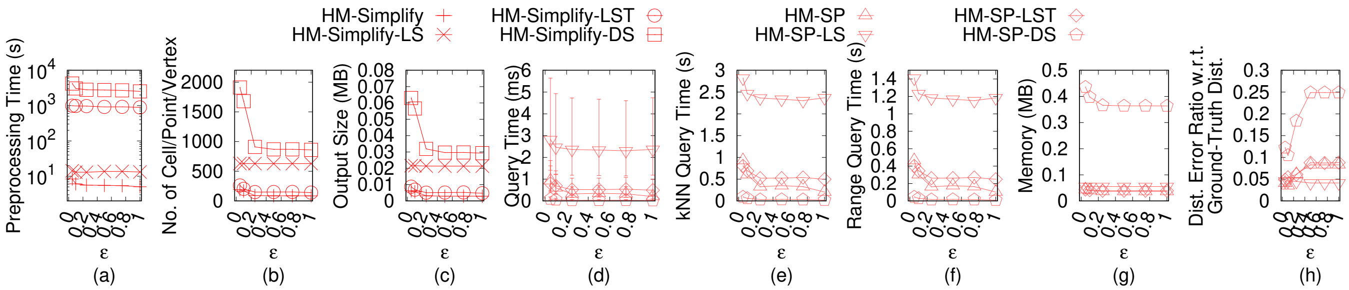


Figure 53: Ablation study for simplification algorithms on GF_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

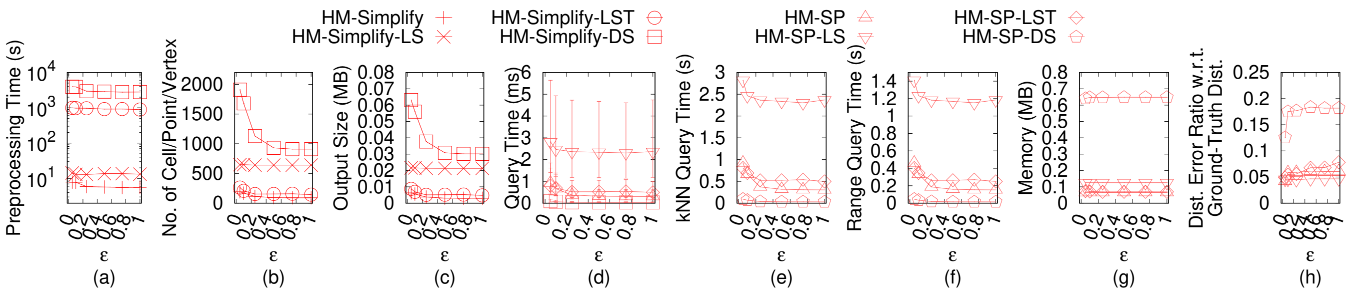


Figure 54: Ablation study for simplification algorithms on LM_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

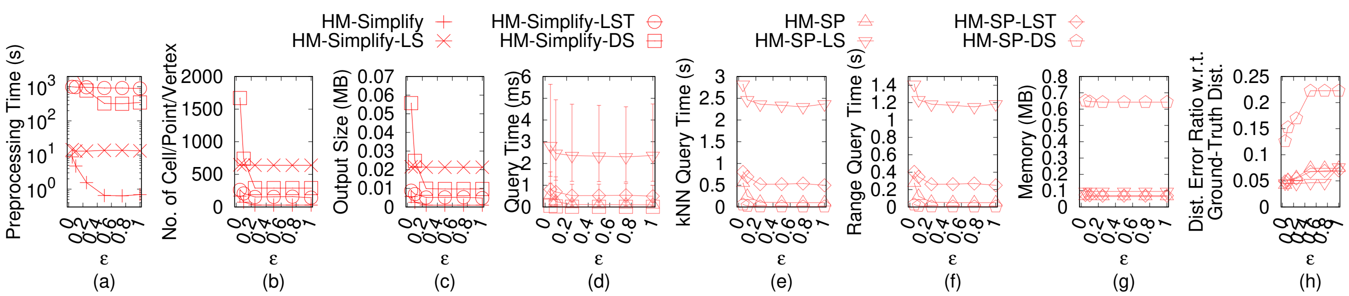


Figure 55: Ablation study for simplification algorithms on RM_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

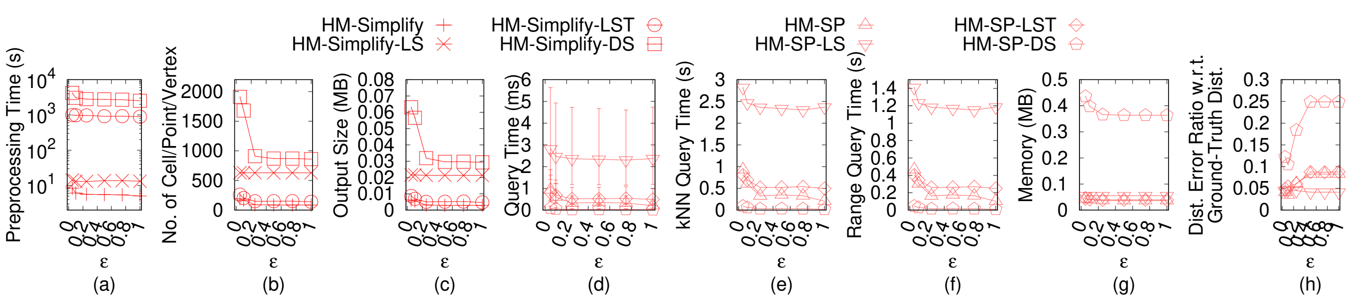


Figure 56: Ablation study for simplification algorithms on BH_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

time, since our proximity query algorithm uses Dijkstra's algorithm once, we can terminate it earlier after visiting all query objects. As d increases, there is no upper bound on the increase in *kNN* query time (since we append the paths computed by Dijkstra's algorithm

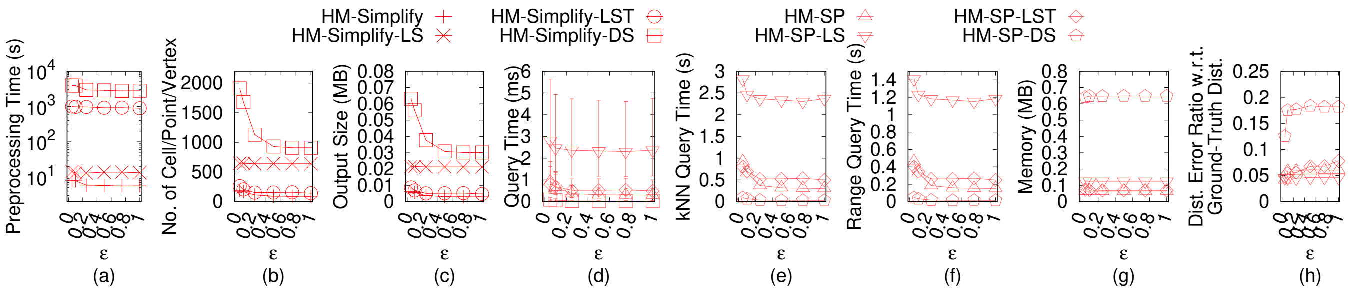


Figure 57: Ablation study for simplification algorithms on EP_h-*small* height map dataset with ground-truth distance in distance error ratio calculation

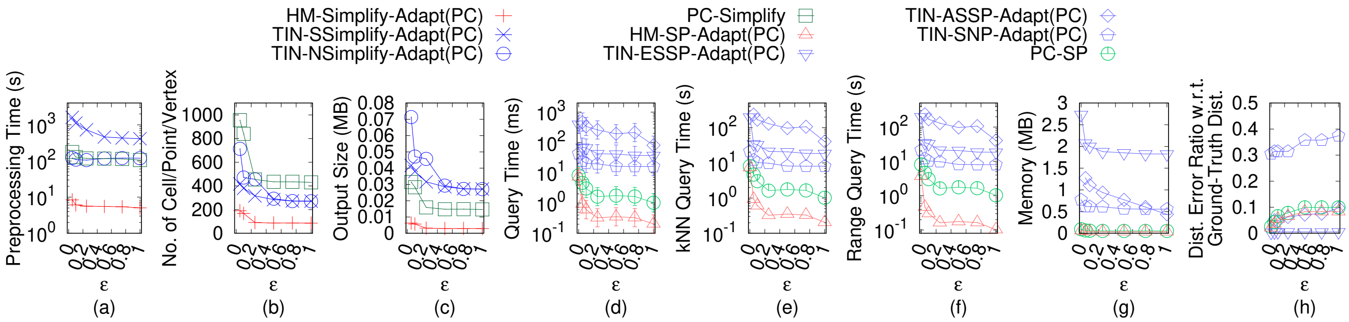


Figure 58: Effect of ϵ on GF_p-*small* point cloud dataset with ground-truth distance in distance error ratio calculation

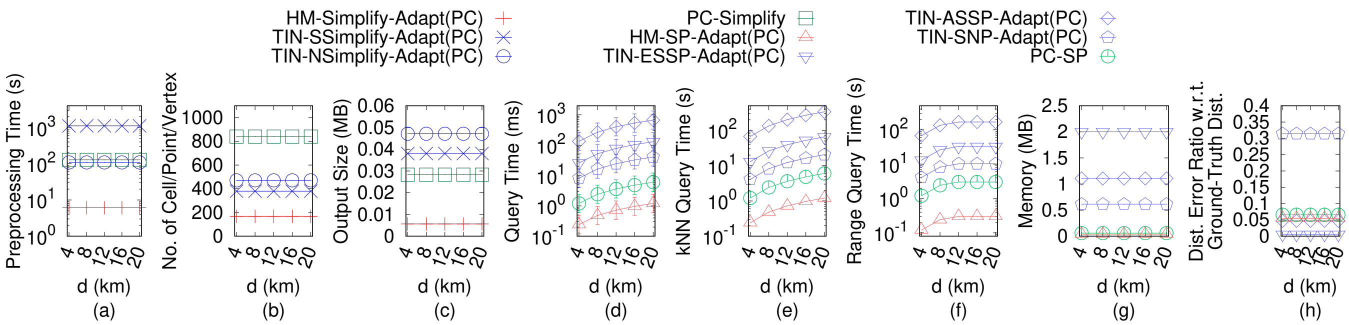


Figure 59: Effect of d on GF_p-*small* point cloud dataset with ground-truth distance in distance error ratio calculation

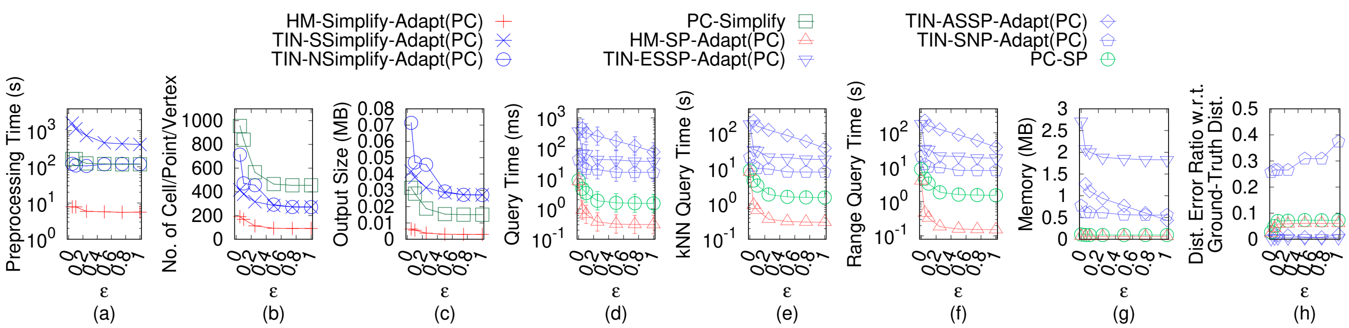


Figure 60: Effect of ϵ on LM_p-*small* point cloud dataset with ground-truth distance in distance error ratio calculation

and the intra-paths as results, we cannot determine the distance correlations among these paths until we perform a linear scan, i.e., we terminate Dijkstra's algorithm based solely on d), but there is an upper bound on the increase in range query time (since we can

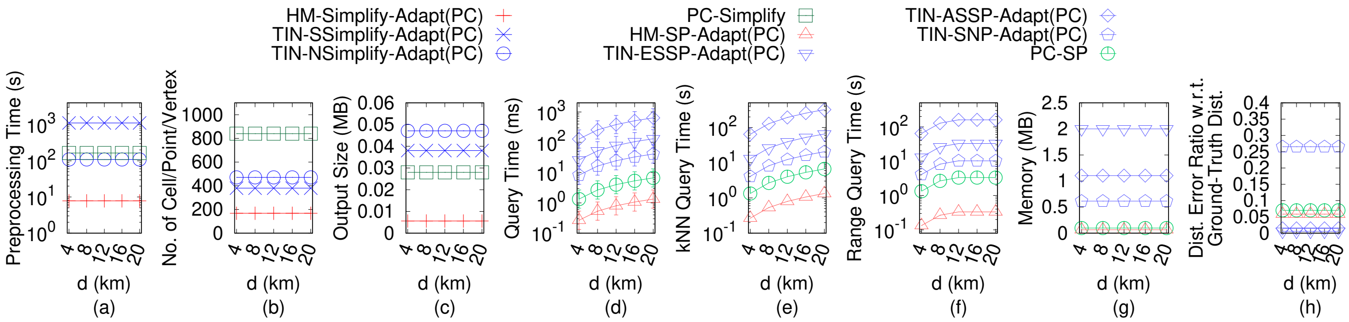


Figure 61: Effect of d on *LM_p-small* point cloud dataset with ground-truth distance in distance error ratio calculation

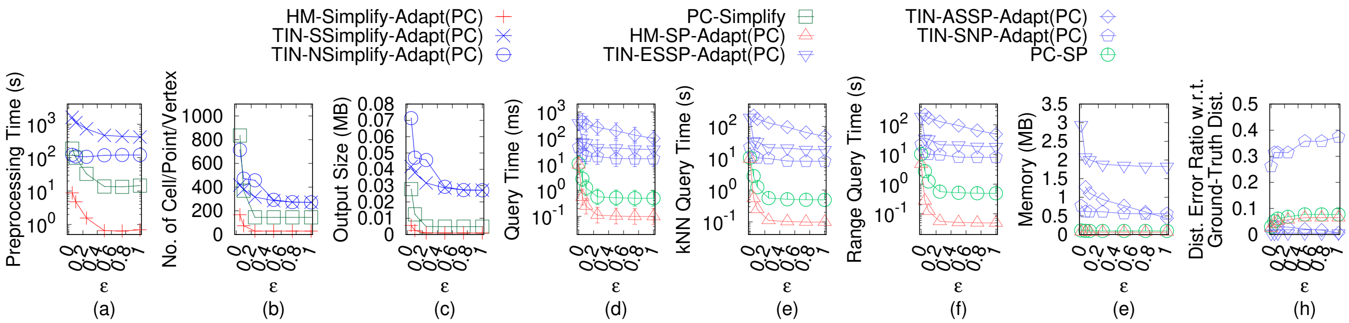


Figure 62: Effect of ϵ on *RM_p-small* point cloud dataset with ground-truth distance in distance error ratio calculation

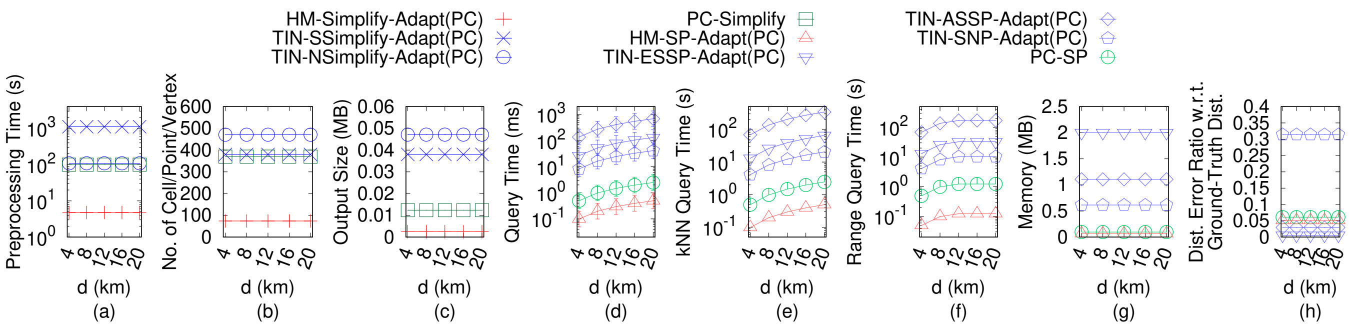


Figure 63: Effect of d on *RM_p-small* point cloud dataset with ground-truth distance in distance error ratio calculation

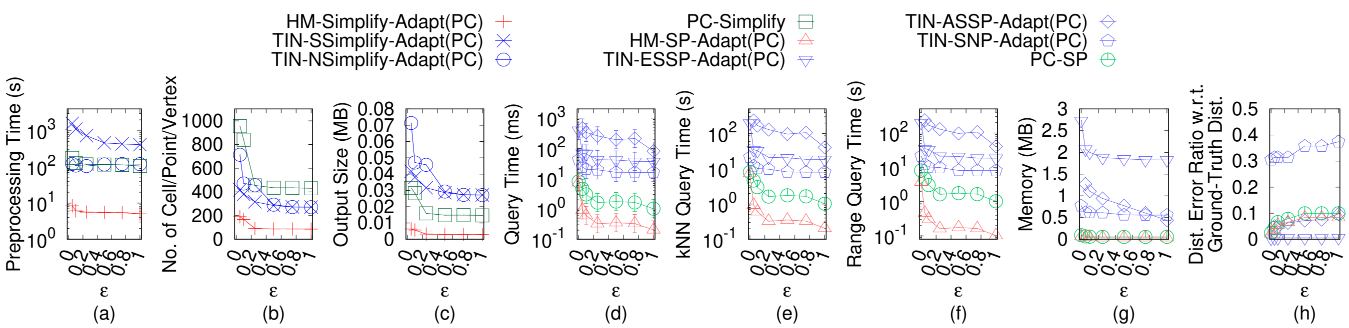


Figure 64: Effect of ϵ on *BH_p-small* point cloud dataset with ground-truth distance in distance error ratio calculation

also terminate Dijkstra's algorithm earlier if the searching distance exceeds r).

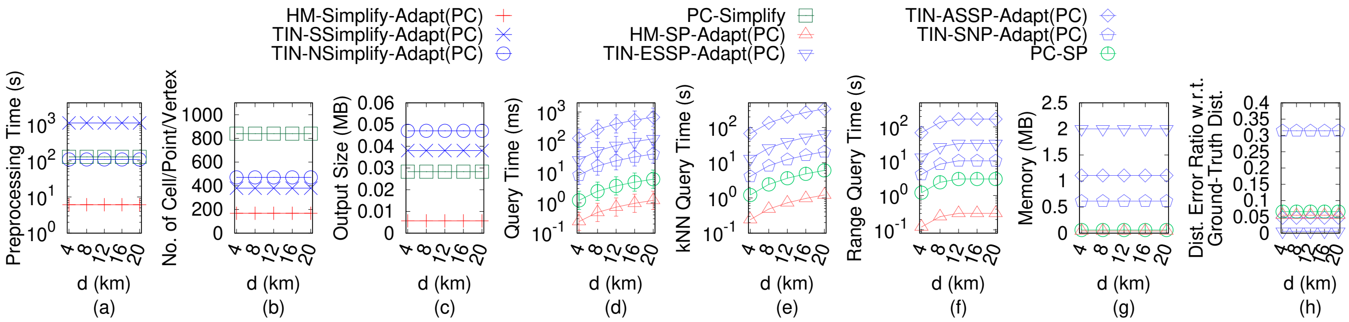


Figure 65: Effect of d on *BH_p-small* point cloud dataset with ground-truth distance in distance error ratio calculation

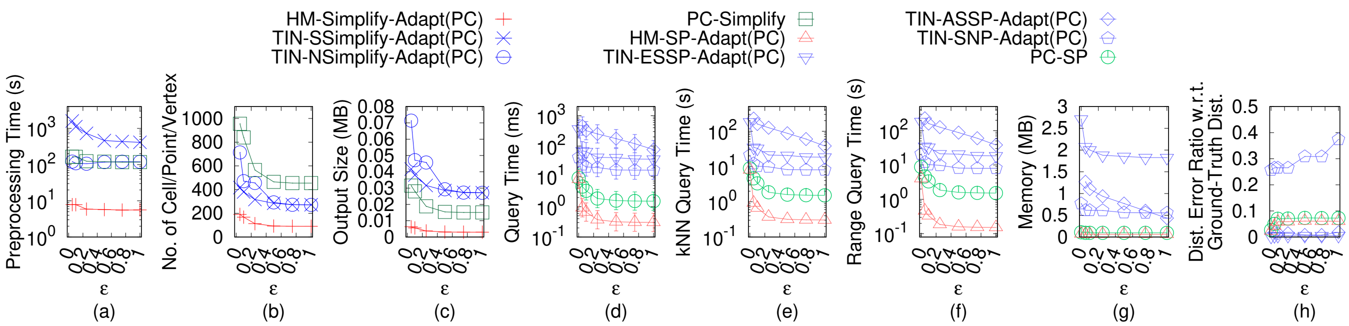


Figure 66: Effect of ϵ on *EP_p-small* point cloud dataset with ground-truth distance in distance error ratio calculation

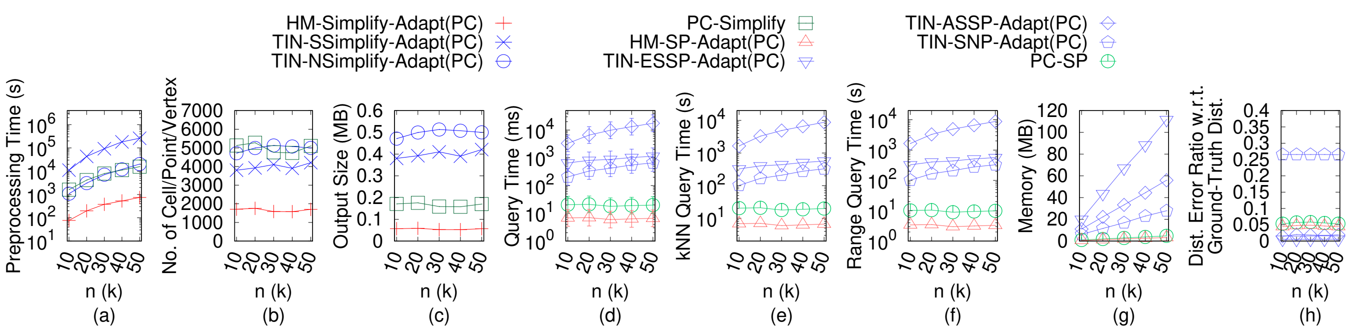


Figure 67: Effect of n on *EP_p-small* point cloud dataset with ground-truth distance in distance error ratio calculation

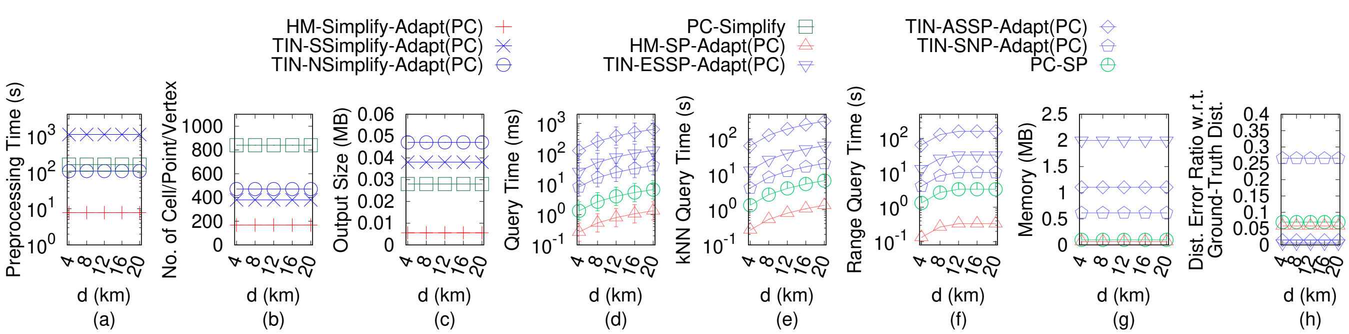


Figure 68: Effect of d on *EP_p-small* point cloud dataset with ground-truth distance in distance error ratio calculation

C.3 Experimental Results for *TIN*s with Ground-truth Distance

We studied proximity queries on *TIN*s using the ground-truth distance for distance error ratio calculation. We compared algorithms *TIN-SSimplify*, *TIN-NSimplify*, *PC-Simplify-Adapt(TIN)*, *HM-Simplify-Adapt(TIN)*, *TIN-ESSP* (on the original and simplified *TIN*), *TIN-ASSP*, *TIN-SNP* (on the original and simplified *TIN*), *PC-SP-Adapt(TIN)* (on the original *TIN* and the simplified point cloud) and

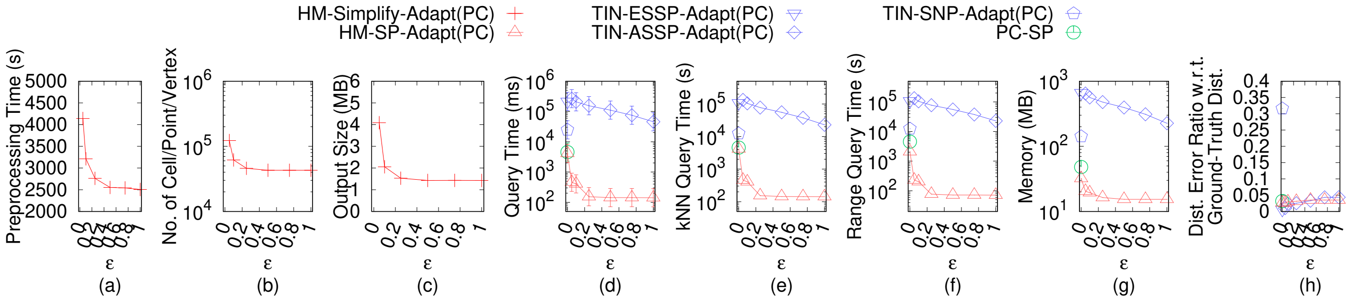


Figure 69: Effect of ϵ on GF_p point cloud dataset with ground-truth distance in distance error ratio calculation

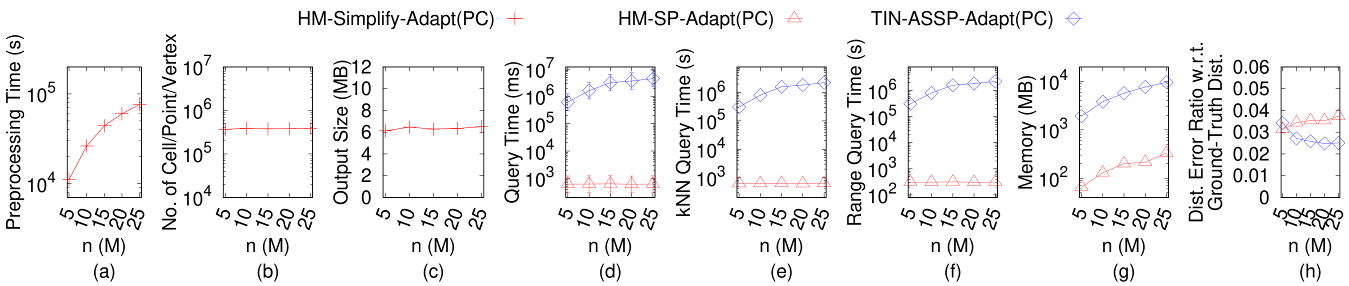


Figure 70: Effect of n on GF_p point cloud dataset with ground-truth distance in distance error ratio calculation

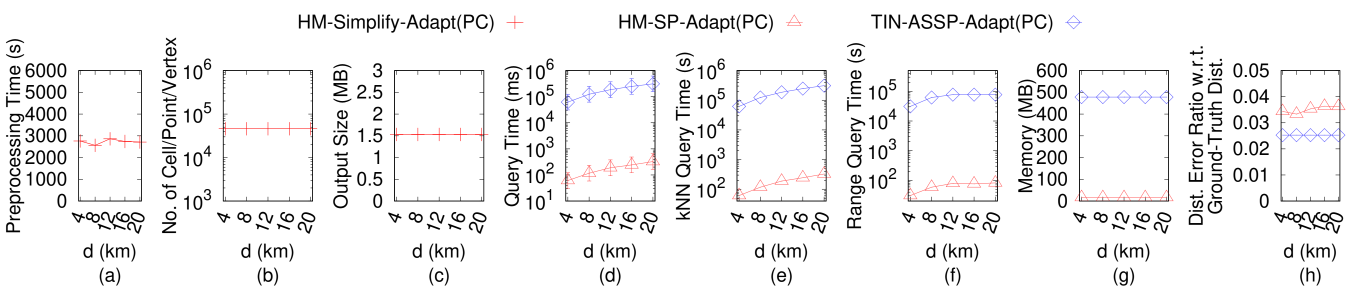


Figure 71: Effect of d on GF_p point cloud dataset with ground-truth distance in distance error ratio calculation

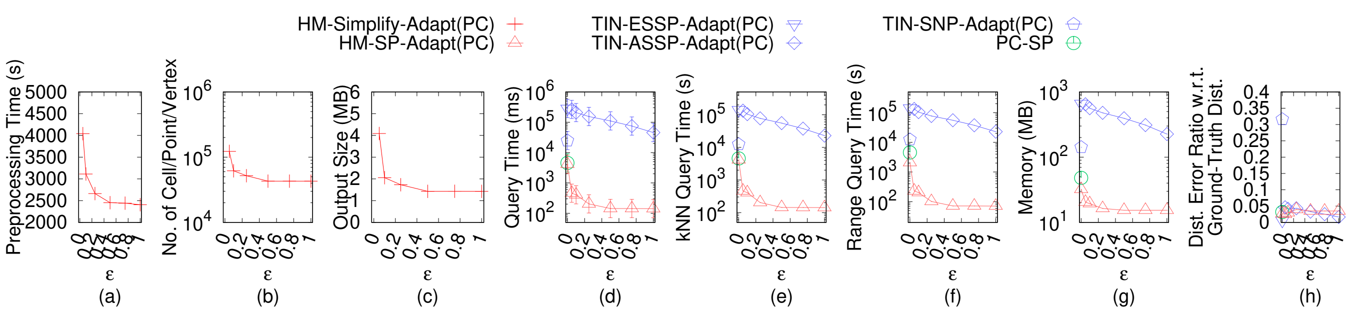


Figure 72: Effect of ϵ on LM_p point cloud dataset with ground-truth distance in distance error ratio calculation

HM-SP-Adapt(TIN) (on the original *TIN* and the simplified height map) on small-version datasets, and compared all algorithms except *TIN-SSimplify*, *TIN-NSimplify* and *PC-Simplify-Adapt(TIN)* on original datasets (due to their excessive simplification time), and except *TIN-ESSP* and *TIN-SNP* on the simplified *TIN*, and *PC-SP-Adapt(TIN)* on the simplified point cloud (due to their dependency on the previous three algorithms).

Effect of ϵ: In Figure 84, Figure 86, Figure 88, Figure 90 and Figure 92, we tested 7 values of ϵ in $\{0, 0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on

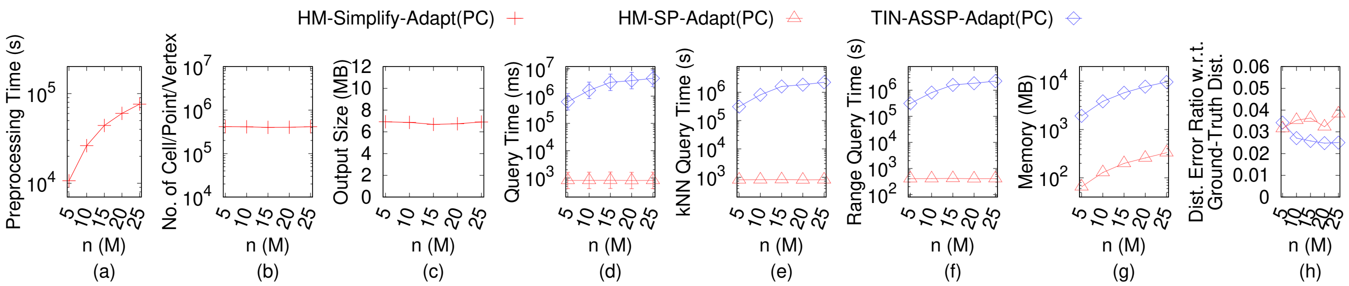


Figure 73: Effect of n on LM_p point cloud dataset with ground-truth distance in distance error ratio calculation

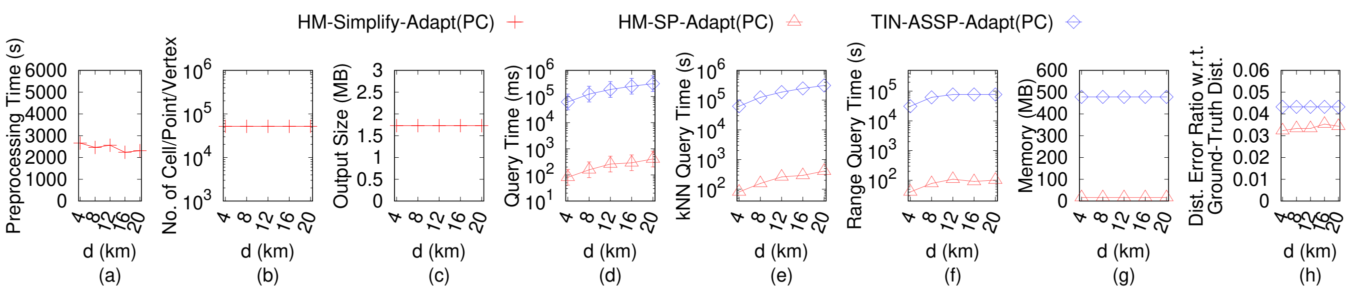


Figure 74: Effect of d on LM_h height map dataset with ground-truth distance in distance error ratio calculation

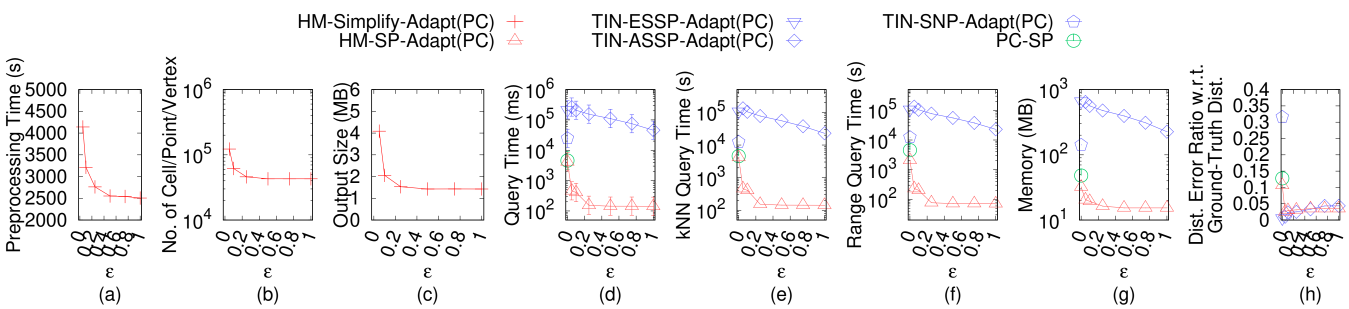


Figure 75: Effect of ϵ on RM_p point cloud dataset with ground-truth distance in distance error ratio calculation

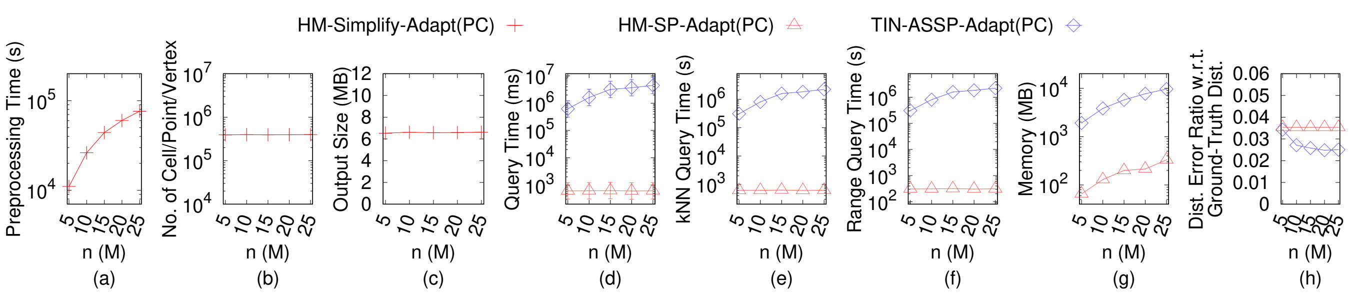


Figure 76: Effect of n on RM_p point cloud dataset with ground-truth distance in distance error ratio calculation

GF_t-small, *LM_t-small*, *RM_t-small*, *BH_t-small* and *EP_t-small* dataset while fixing n at 1k for baseline comparisons. In Figure 95, Figure 98, Figure 101, Figure 104 and Figure 107, we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_t, LM_t, RM_t, BH_t and EP_t dataset while fixing n at 0.5M for baseline comparisons. The preprocessing time, number of cells of the simplified height map and output size of *HM-Simplify-Adapt(TIN)* are much smaller than three baselines'. The proximity queries time of *HM-SP-Adapt(TIN)* on the simplified height map are also small since its simplified height map has a small output size.

Effect of n (scalability test): In Figure 93, we tested 5 values of n in {10k, 20k, 30k, 40k, 50k} on *EP_t-small* dataset while fixing ϵ

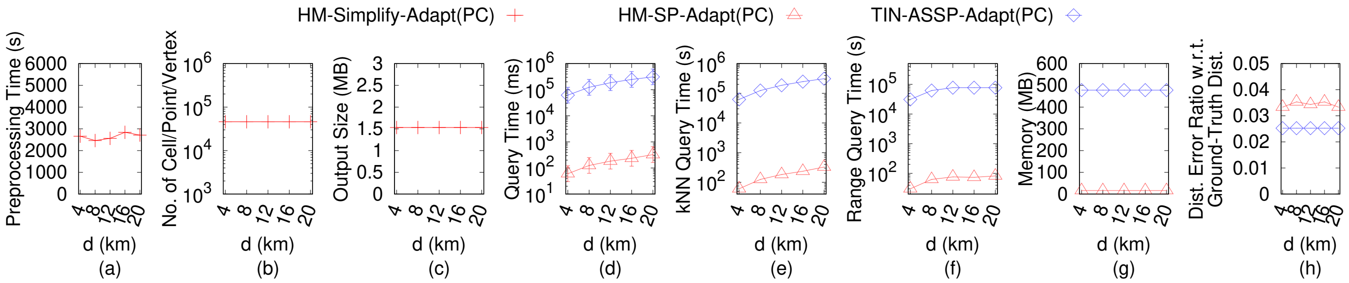


Figure 77: Effect of d on RM_h height map dataset with ground-truth distance in distance error ratio calculation

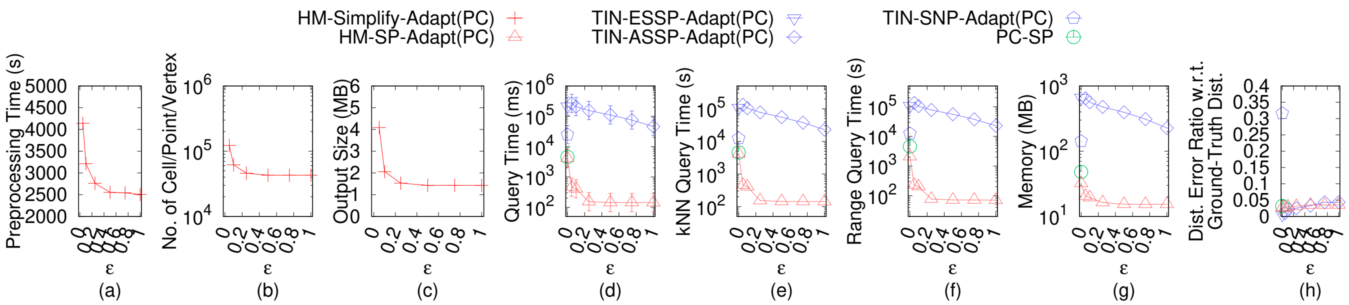


Figure 78: Effect of ϵ on BH_p point cloud dataset with ground-truth distance in distance error ratio calculation

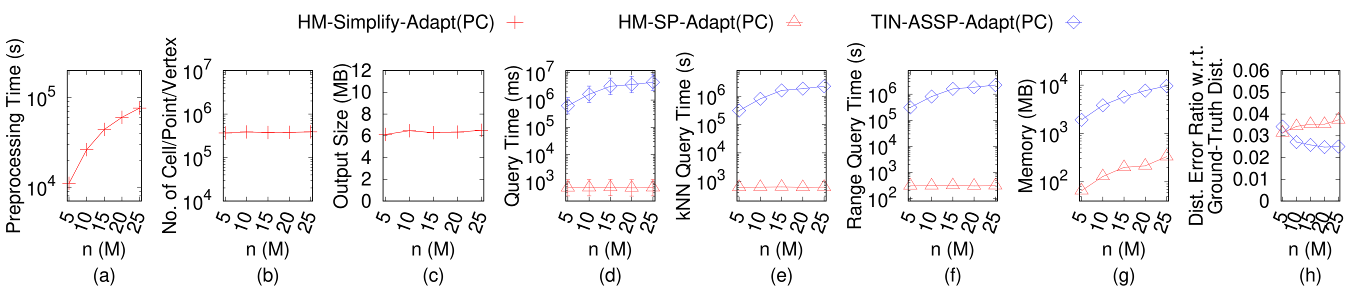


Figure 79: Effect of n on BH_p point cloud dataset with ground-truth distance in distance error ratio calculation

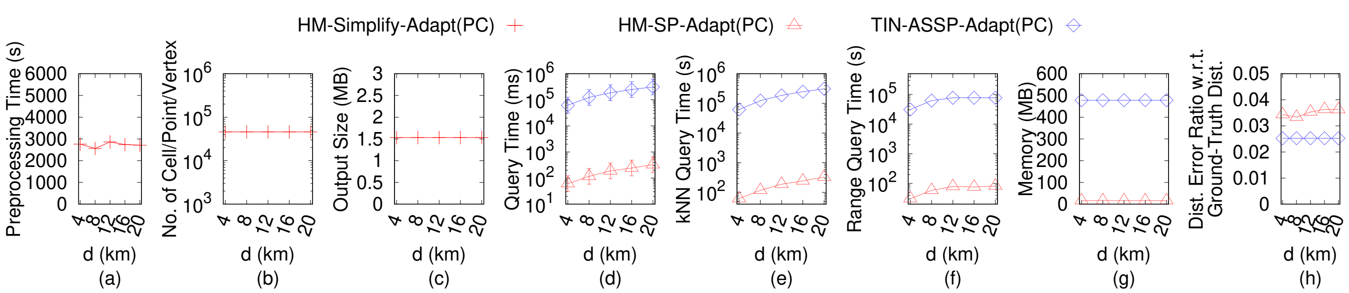


Figure 80: Effect of d on BH_h height map dataset with ground-truth distance in distance error ratio calculation

at 0.1 for baseline comparisons. In Figure 96, Figure 99, Figure 102, Figure 105 and Figure 108, we tested 5 values of n in {5M, 10M, 15M, 20M, 25M} on GF_t, LM_t, RM_t, BH_t and EP_t dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-Simplify-Adapt(TIN)* outperforms all the remaining simplification algorithms and *HM-SP-Adapt(TIN)* on the simplified height map outperforms all the remaining proximity query algorithms.

Effect of d: In Figure 85, Figure 87, Figure 89, Figure 91 and Figure 94, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on GF_t*-small*, LM_t*-small*, RM_t*-small*, BH_t*-small* and EP_t*-small* dataset

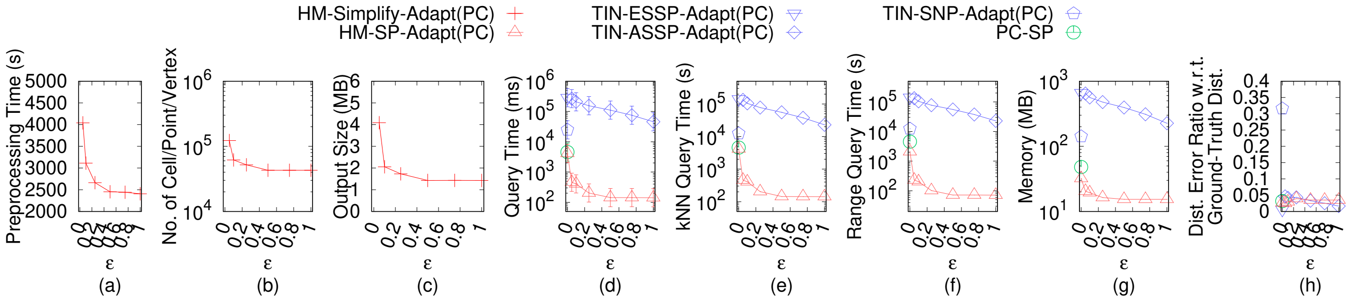


Figure 81: Effect of ϵ on EP_p point cloud dataset with ground-truth distance in distance error ratio calculation

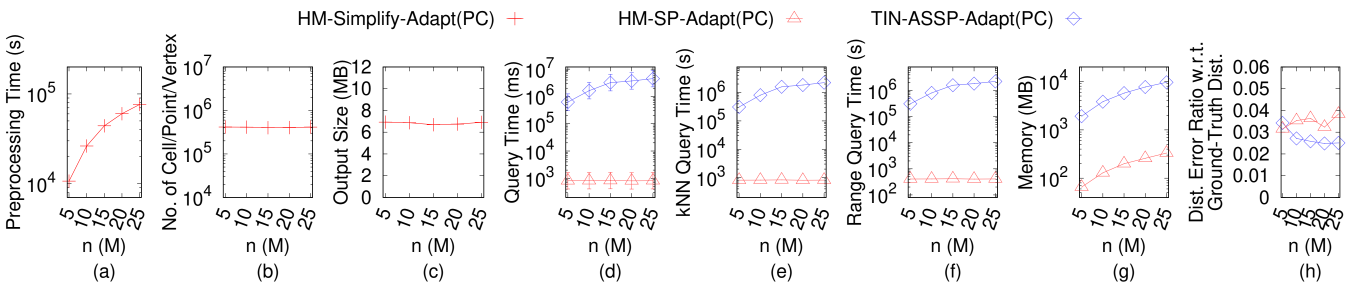


Figure 82: Effect of n on EP_p point cloud dataset with ground-truth distance in distance error ratio calculation

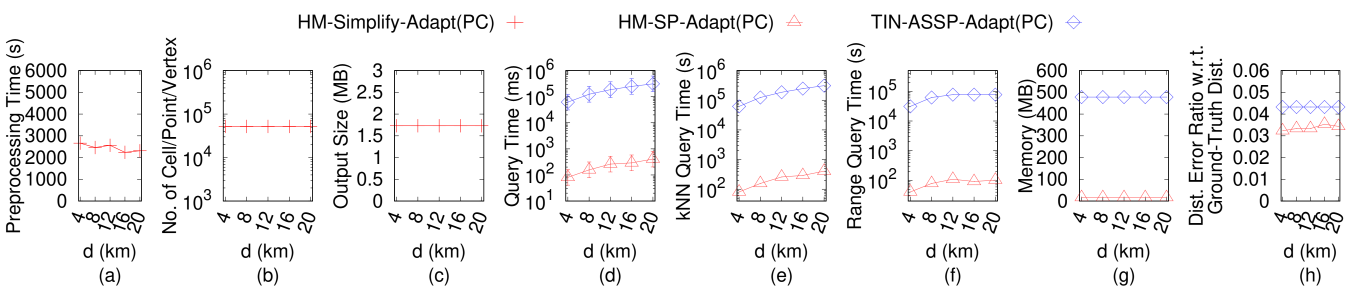


Figure 83: Effect of d on EP_h height map dataset with ground-truth distance in distance error ratio calculation

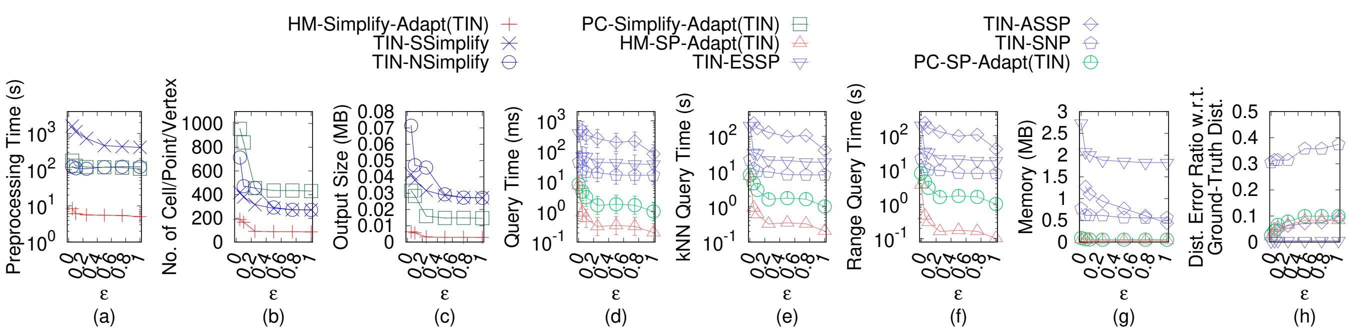


Figure 84: Effect of ϵ on GF_t-*small TIN* dataset with ground-truth distance in distance error ratio calculation

while fixing ϵ at 0.1 and n at 1k for baseline comparisons. In Figure 97, Figure 100, Figure 103, Figure 106 and Figure 109, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on GF_t, LM_t, RM_t, BH_t and EP_t dataset while fixing ϵ at 0.25 and n at 0.5M for baseline comparisons. A smaller d reduces *kNN* and range query time, since our proximity query algorithm uses Dijkstra's algorithm once, we can terminate it earlier after visiting all query objects. As d increases, there is no upper bound on the increase in *kNN* query time (since we append the paths computed by Dijkstra's algorithm and the intra-paths as results, we cannot determine the distance

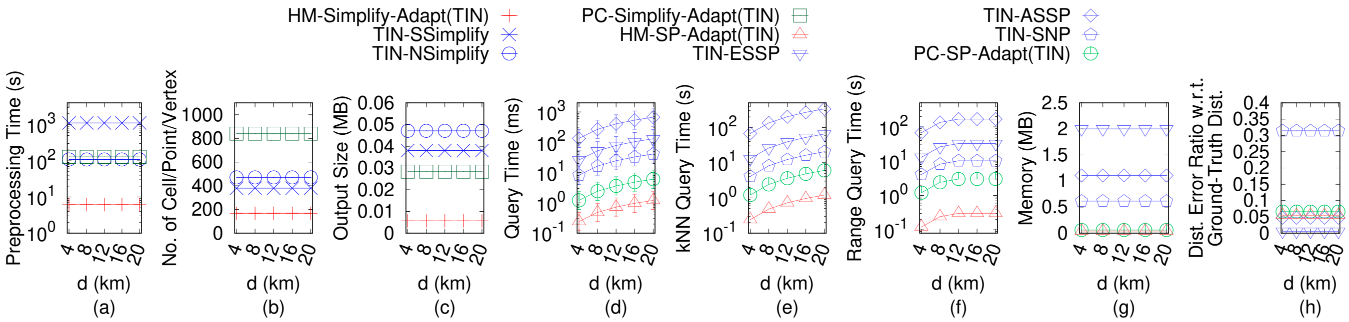


Figure 85: Effect of d on GF_t*-small TIN* dataset with ground-truth distance in distance error ratio calculation

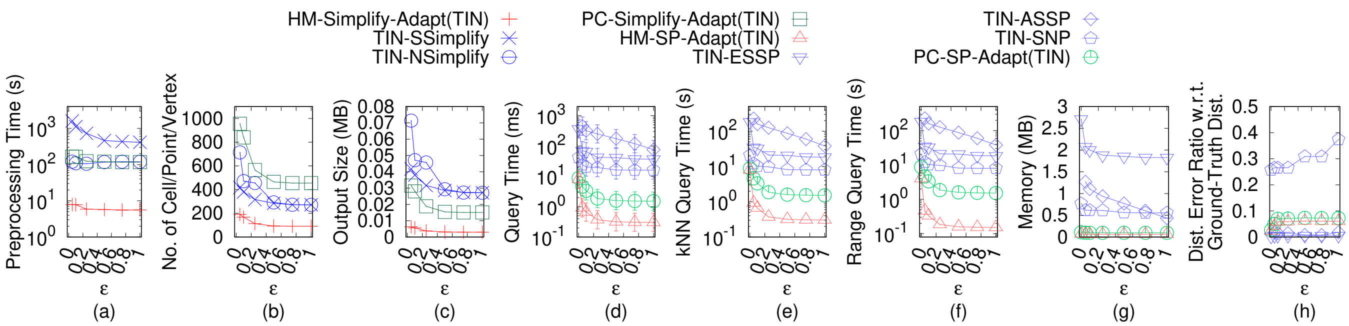


Figure 86: Effect of ϵ on LM_t*-small TIN* dataset with ground-truth distance in distance error ratio calculation

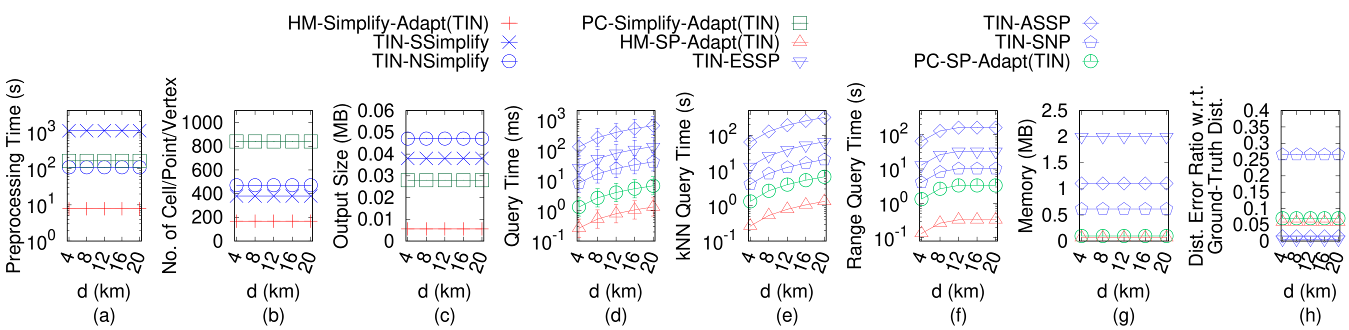


Figure 87: Effect of d on LM_t*-small TIN* dataset with ground-truth distance in distance error ratio calculation

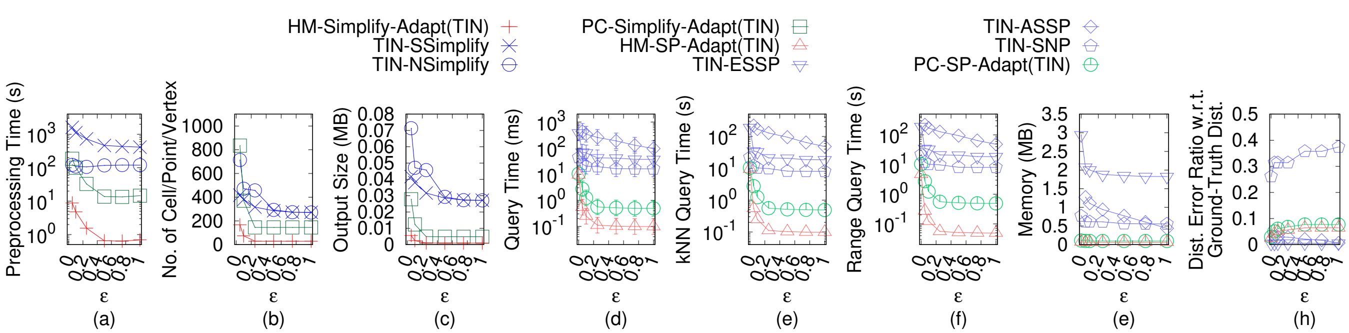


Figure 88: Effect of ϵ on RM_t*-small TIN* dataset with ground-truth distance in distance error ratio calculation

correlations among these paths until we perform a linear scan, i.e., we terminate Dijkstra's algorithm based solely on d), but there is

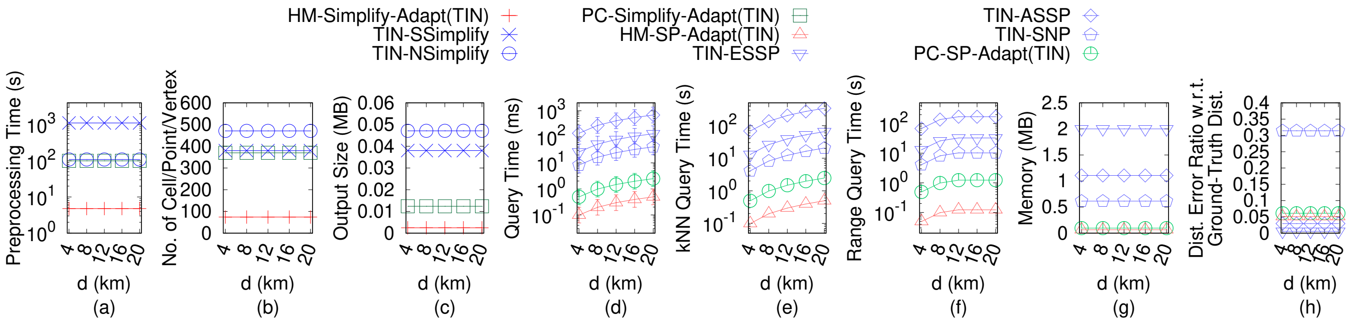


Figure 89: Effect of d on *RM_t-small TIN* dataset with ground-truth distance in distance error ratio calculation

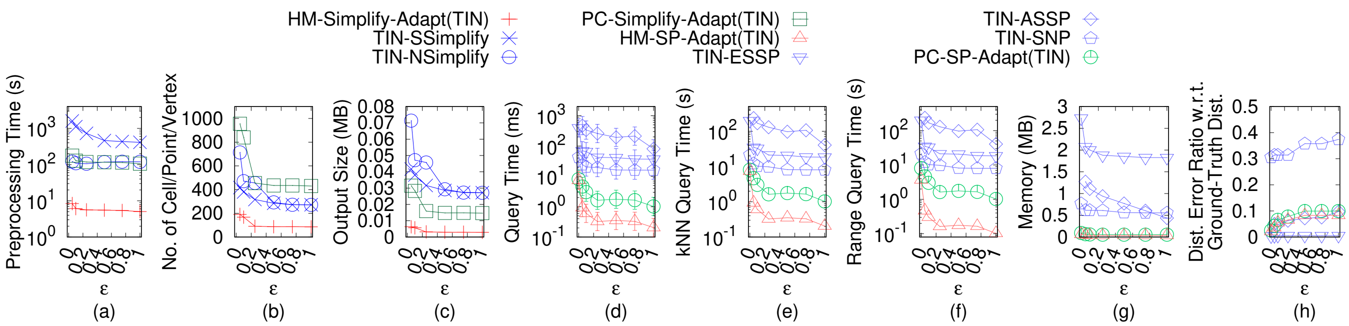


Figure 90: Effect of ϵ on *BH_t-small TIN* dataset with ground-truth distance in distance error ratio calculation

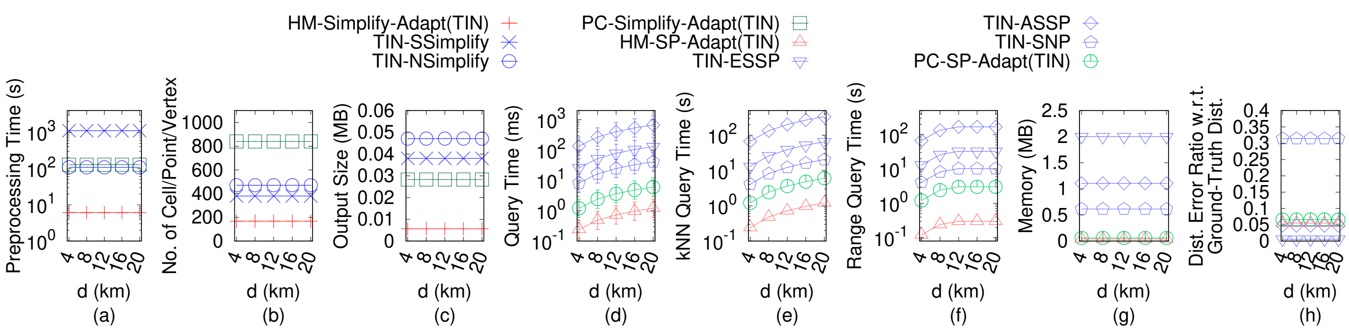


Figure 91: Effect of d on *BH_t-small TIN* dataset with ground-truth distance in distance error ratio calculation

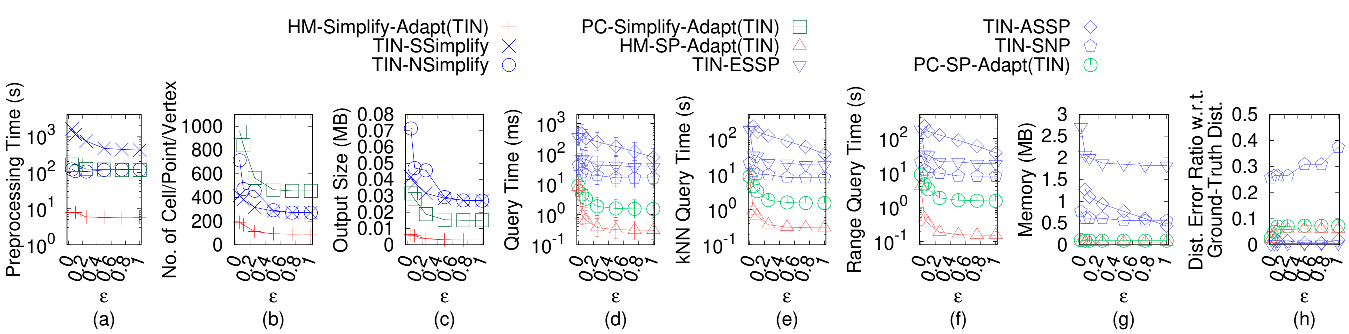


Figure 92: Effect of ϵ on *EP_t-small TIN* dataset with ground-truth distance in distance error ratio calculation

an upper bound on the increase in range query time (since we can also terminate Dijkstra's algorithm earlier if the searching distance exceeds r).

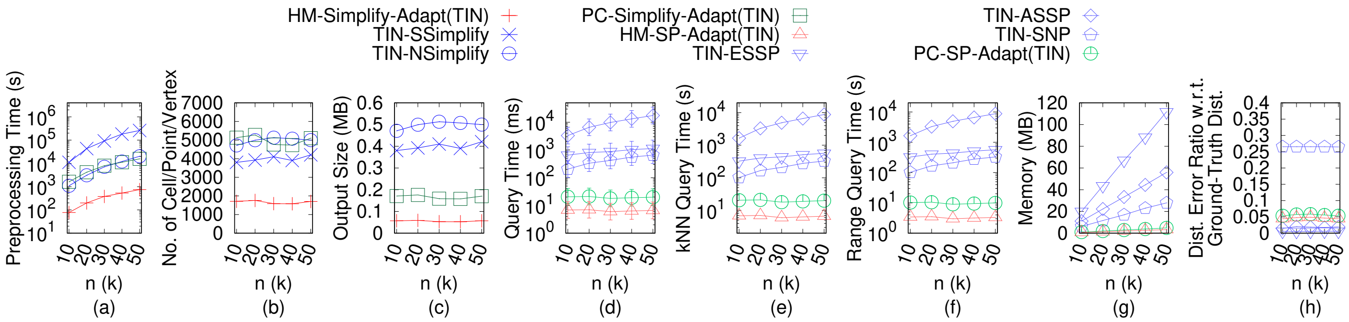


Figure 93: Effect of n on EP_t-small TIN dataset with ground-truth distance in distance error ratio calculation

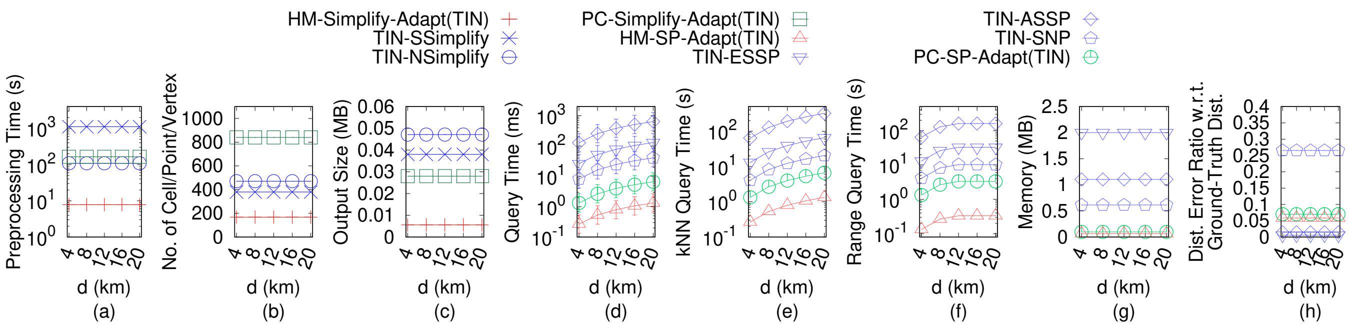


Figure 94: Effect of d on EP_t-small TIN dataset with ground-truth distance in distance error ratio calculation

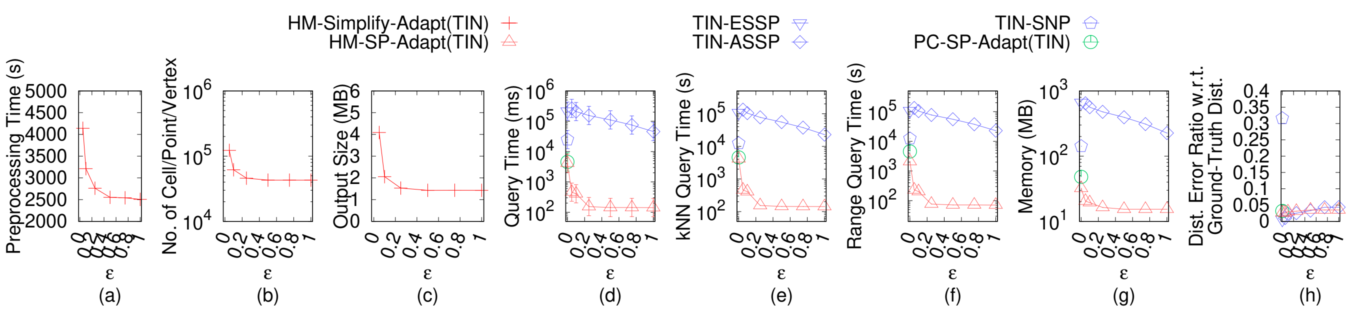


Figure 95: Effect of ϵ on GF_t TIN dataset with ground-truth distance in distance error ratio calculation

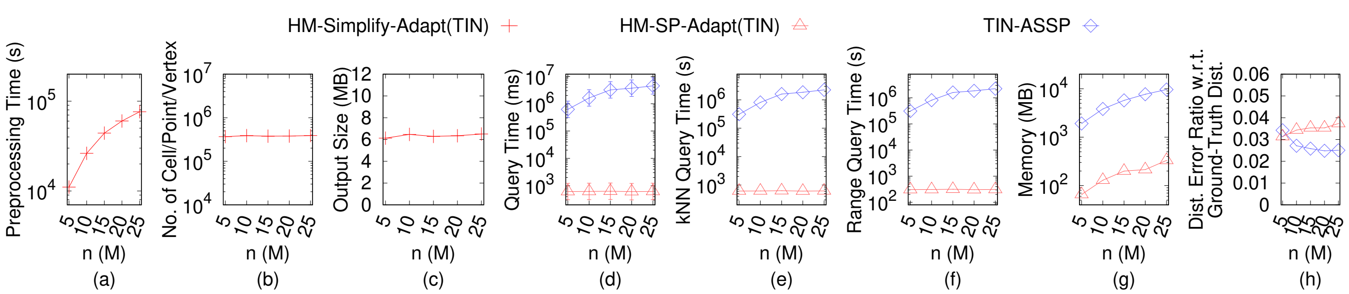


Figure 96: Effect of n on GF_t TIN dataset with ground-truth distance in distance error ratio calculation

C.4 Experimental Results for Height Maps with Optimal Distance

We studied proximity queries on height maps using the optimal distance for distance error ratio calculation. We compared algorithms *TIN-SSimplify-Adapt(HM)*, *TIN-NSimplify-Adapt(HM)*, *PC-Simplify-Adapt(HM)*, *HM-Simplify*, *TIN-ESSP-Adapt(HM)* (on the

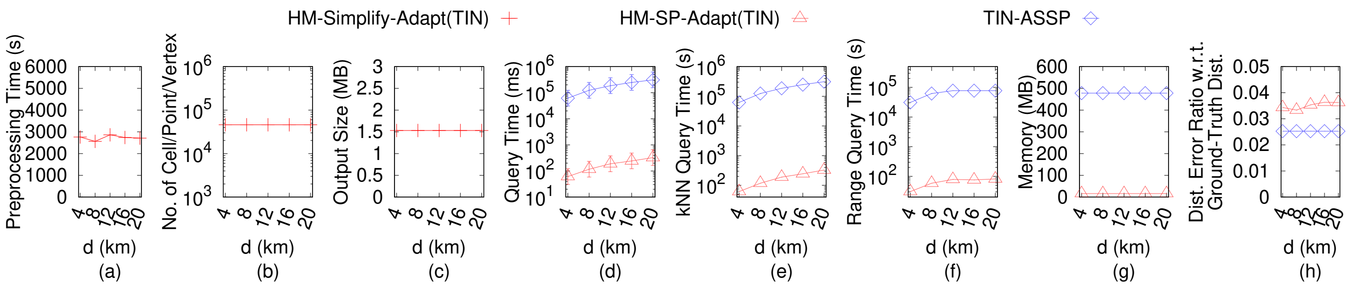


Figure 97: Effect of d on GF_t *TIN* dataset with ground-truth distance in distance error ratio calculation

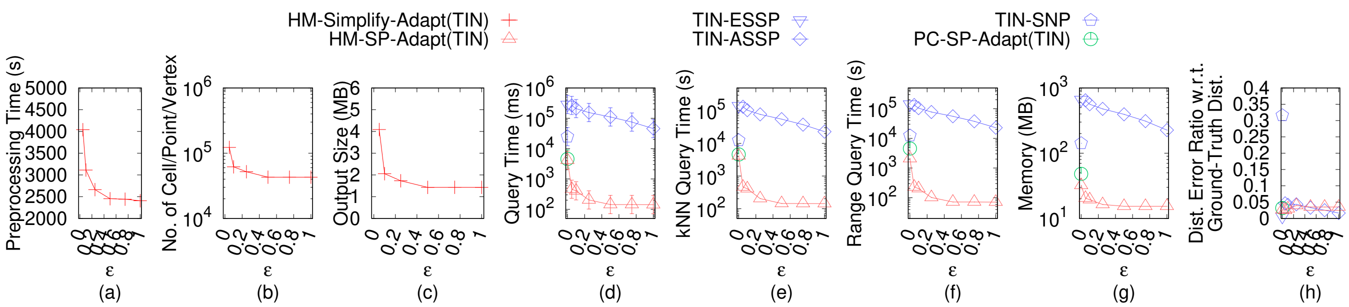


Figure 98: Effect of ϵ on LM_t *TIN* dataset with ground-truth distance in distance error ratio calculation

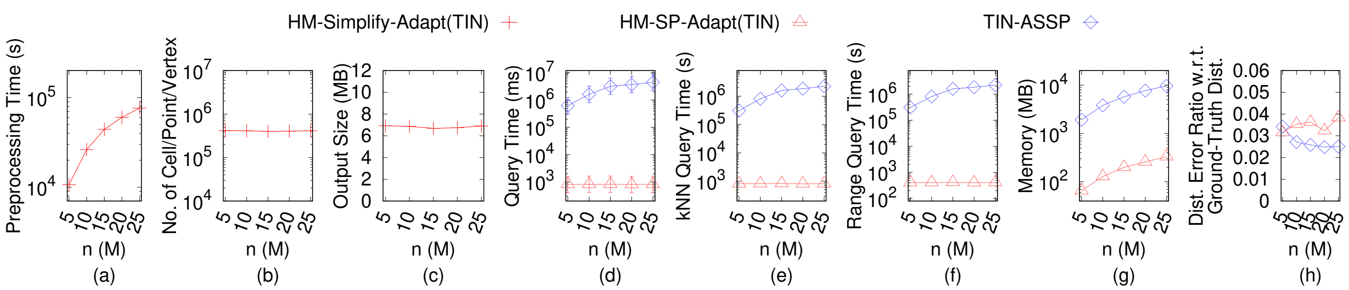


Figure 99: Effect of n on LM_t *TIN* dataset with ground-truth distance in distance error ratio calculation

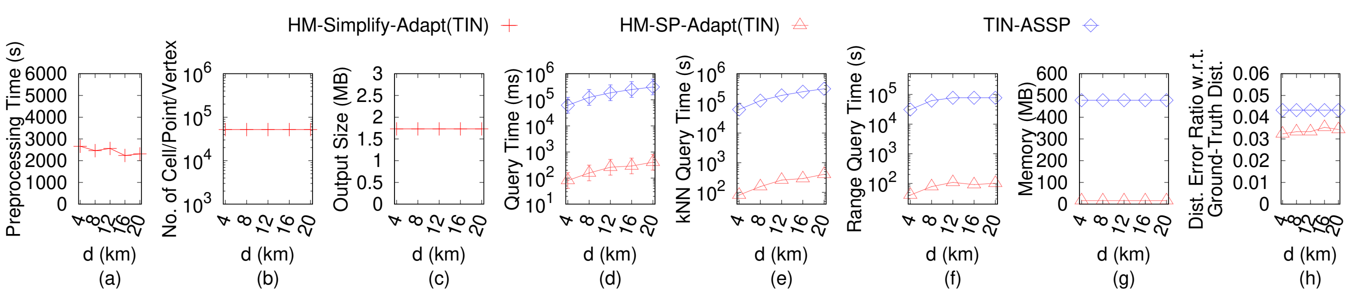


Figure 100: Effect of d on LM_t *TIN* dataset with ground-truth distance in distance error ratio calculation

original height map and the simplified *TIN*), *TIN-ASSP-Adapt(HM)*, *TIN-SNP-Adapt(HM)* (on the original height map and the simplified *TIN*), *PC-SP-Adapt(HM)* (on the original and simplified point cloud) and *HM-SP* (on the original and simplified height map) on small-version datasets, and compared all algorithms except *TIN-SSimplify-Adapt(HM)*, *TIN-NSimplify-Adapt(HM)* and *PC-Simplify-Adapt(HM)* on original datasets (due to their excessive simplification time), and except *TIN-ESSP-Adapt(HM)* and *TIN-SNP-Adapt(HM)* on the simplified *TIN*, and *PC-SP-Adapt(HM)* on the simplified point cloud (due to their dependency on the previous two algorithms).

C.4.1 ***Baseline comparisons*****.** **Effect of ϵ**: In Figure 110, Figure 112, Figure 114, Figure 116 and Figure 118, we tested 7 values

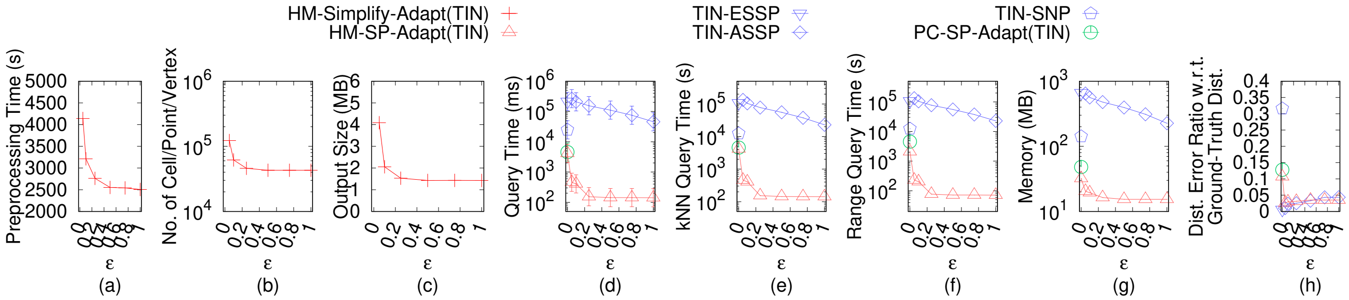


Figure 101: Effect of ϵ on RM_t *TIN* dataset with ground-truth distance in distance error ratio calculation

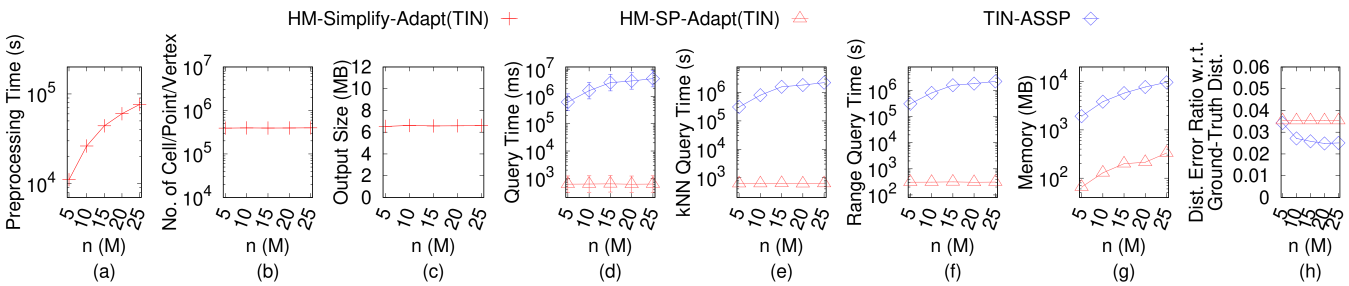


Figure 102: Effect of n on RM_t *TIN* dataset with ground-truth distance in distance error ratio calculation

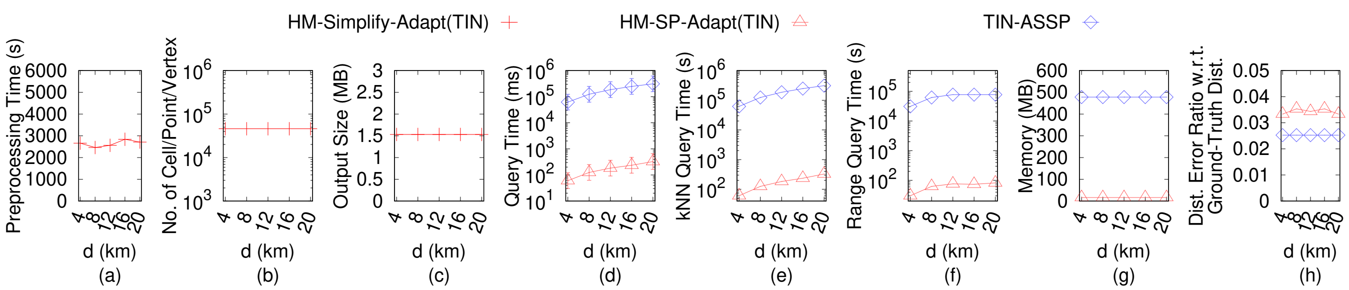


Figure 103: Effect of d on RM_t *TIN* dataset with ground-truth distance in distance error ratio calculation

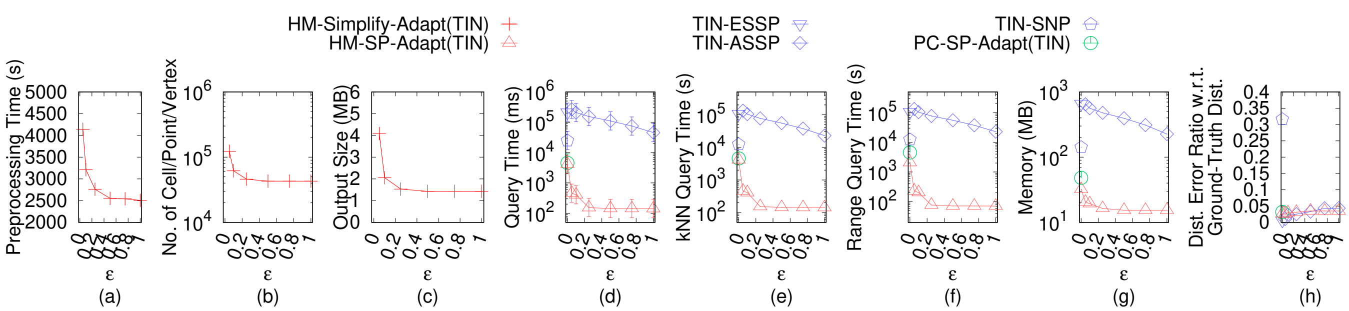


Figure 104: Effect of ϵ on BH_t *TIN* dataset with ground-truth distance in distance error ratio calculation

of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_h*-small*, LM_h*-small*, RM_h*-small*, BH_h*-small* and EP_h*-small* dataset while fixing n at 1k for baseline comparisons. In Figure 121, Figure 124, Figure 127, Figure 130 and Figure 133, we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_h, LM_h, RM_h, BH_h and EP_h dataset while fixing n at 0.5M for baseline comparisons. The preprocessing time of *HM-Simplify* is much smaller than three baselines' due to the efficient height map shortest path query and efficient ϵ-approximate simplified height map checking. The number of cells of the simplified height map and output size of *HM-Simplify* are also much

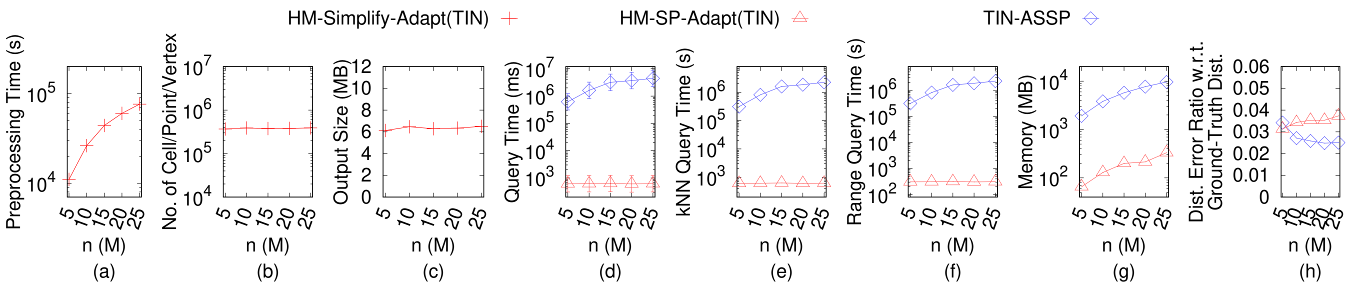


Figure 105: Effect of n on BH_t *TIN* dataset with ground-truth distance in distance error ratio calculation

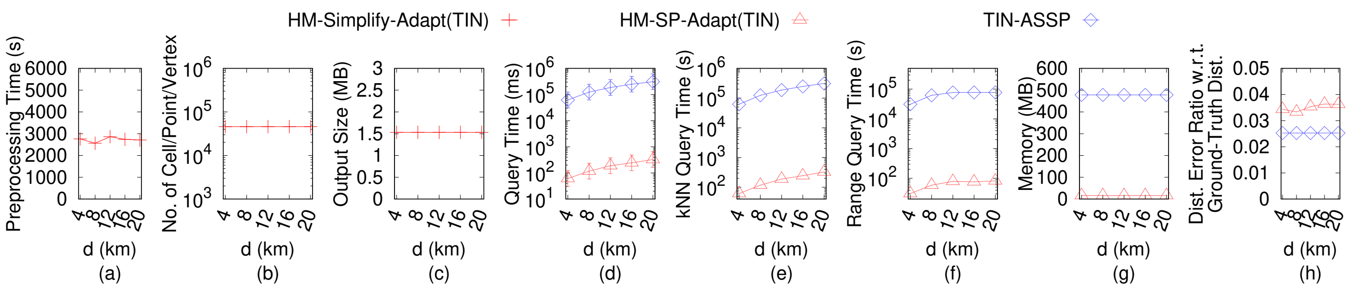


Figure 106: Effect of d on BH_t *TIN* dataset with ground-truth distance in distance error ratio calculation

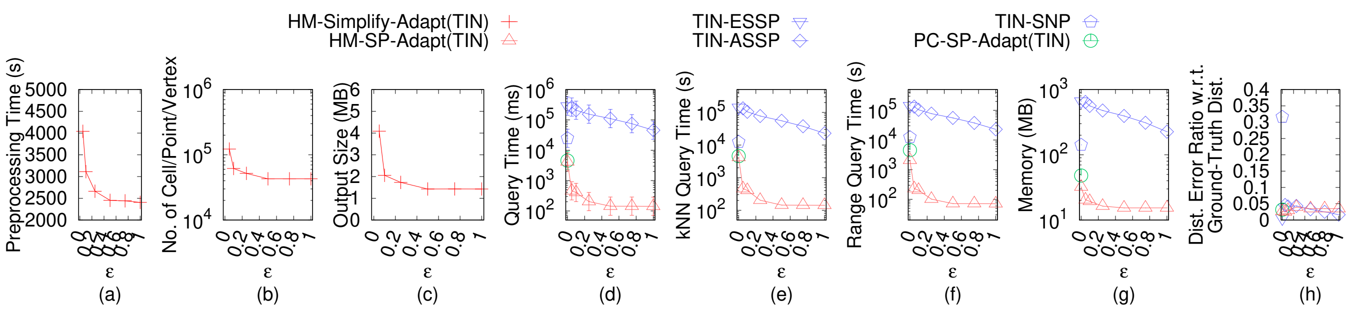


Figure 107: Effect of ϵ on EP_t *TIN* dataset with ground-truth distance in distance error ratio calculation

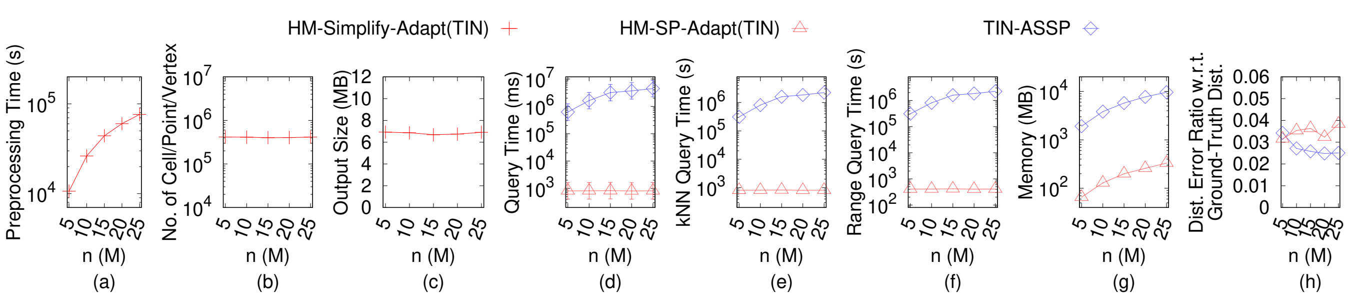


Figure 108: Effect of n on EP_t *TIN* dataset with ground-truth distance in distance error ratio calculation

smaller than three baselines' due to the novel cell merging technique. The shortest path query time and the *kNN* query time of *HM-SP* on the simplified height map are also small since its simplified height map has a small output size. Although increasing ϵ will slightly increase the experimental distance error ratio of *HM-SP* on the simplified height map, its value is 0.0186, i.e., close to 0. So, increasing ϵ has no impact on the experimental *kNN* and range query error ratios, their values are 0, and their results are omitted. Compared with the real shortest distance, since the relative error of the optimal distance is 0.0613, the relative error of the shortest distance returned by *HM-SP* on the simplified height map is at most

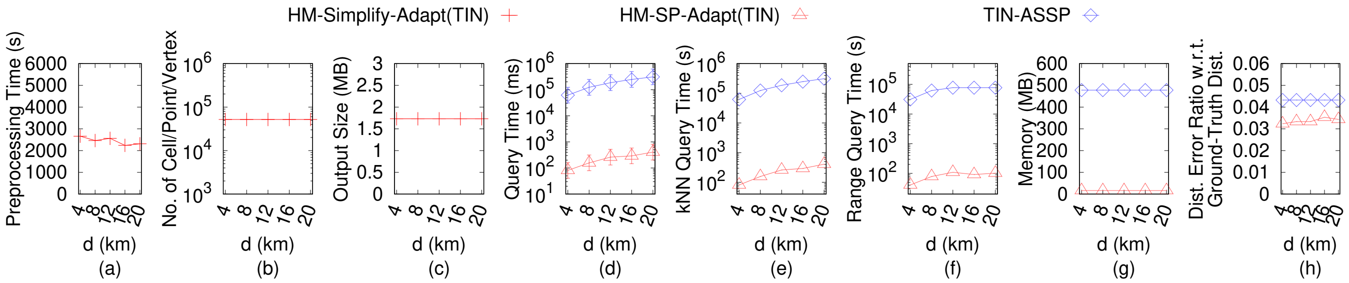


Figure 109: Effect of d on EP_t *TIN* dataset with ground-truth distance in distance error ratio calculation

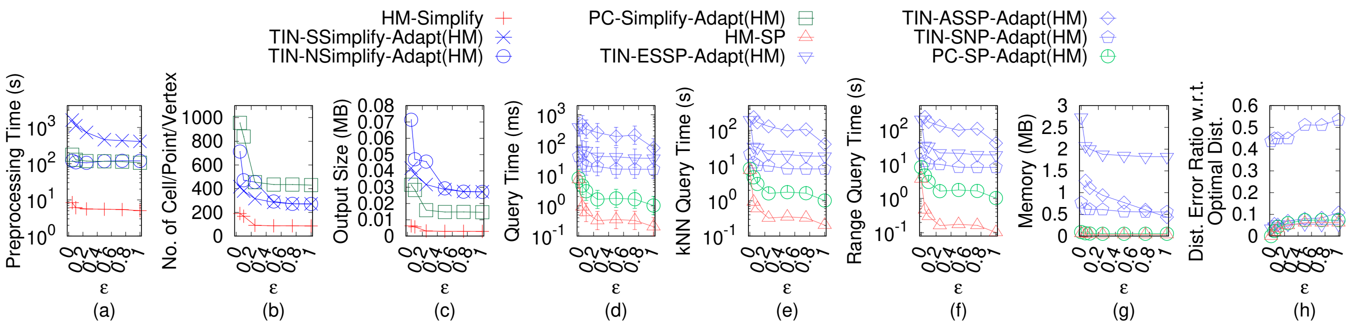


Figure 110: Effect of ϵ on GF_h*-small* height map dataset with optimal distance in distance error ratio calculation

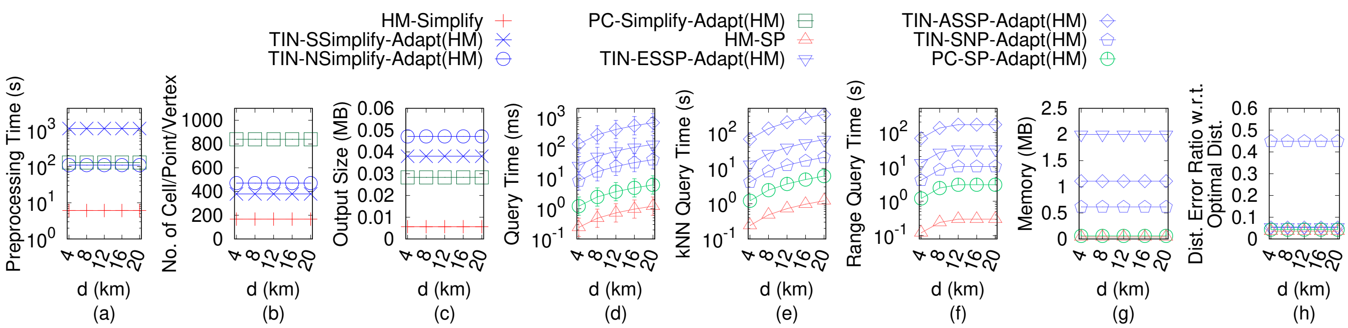


Figure 111: Effect of d on GF_h*-small* height map dataset with optimal distance in distance error ratio calculation

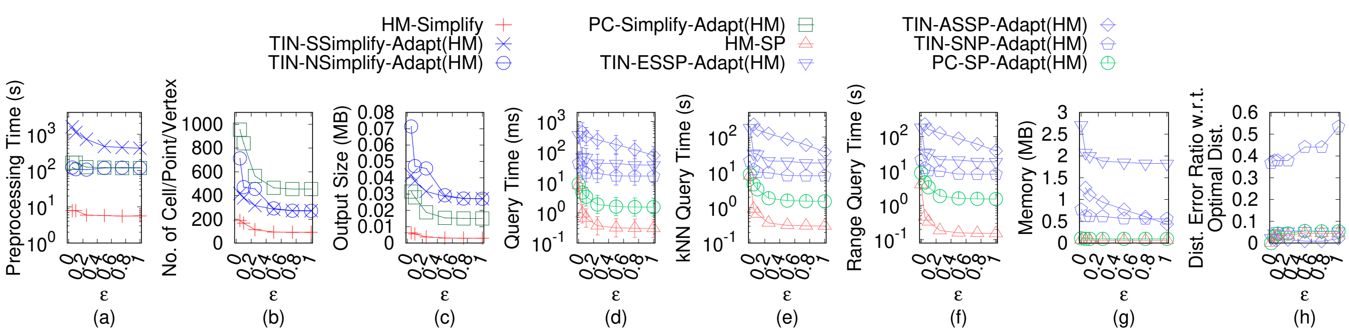


Figure 112: Effect of ϵ on LM_h*-small* height map dataset with optimal distance in distance error ratio calculation

$0.0810 = \max(0.0810(= (1 + 0.0186) \times (1 + 0.0613) - 1), 0.0788(= 1 - (1 - 0.0186) \times (1 - 0.0613))$.

Effect of n (scalability test): In Figure 119, we tested 5 values of n in {10k, 20k, 30k, 40k, 50k} on EP_h*-small* dataset while fixing ϵ at 0.1 for baseline comparisons. In Figure 122, Figure 125, Figure 128,

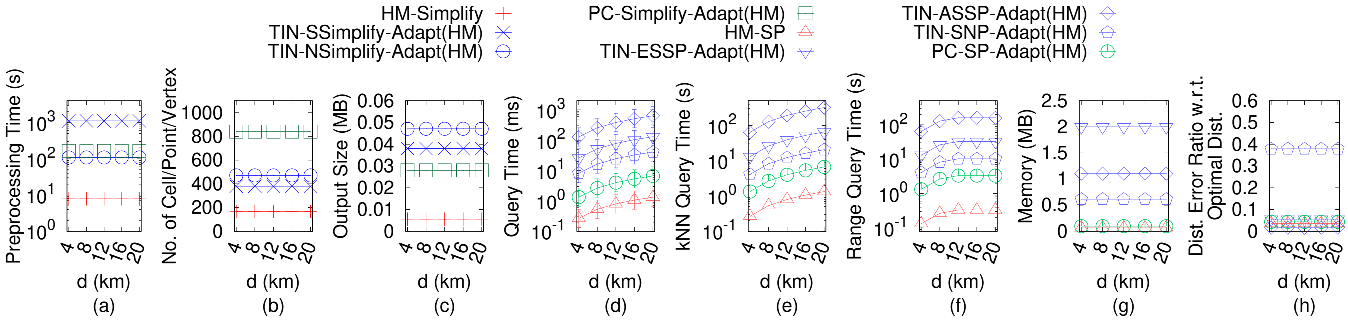


Figure 113: Effect of d on LM_h-small height map dataset with optimal distance in distance error ratio calculation

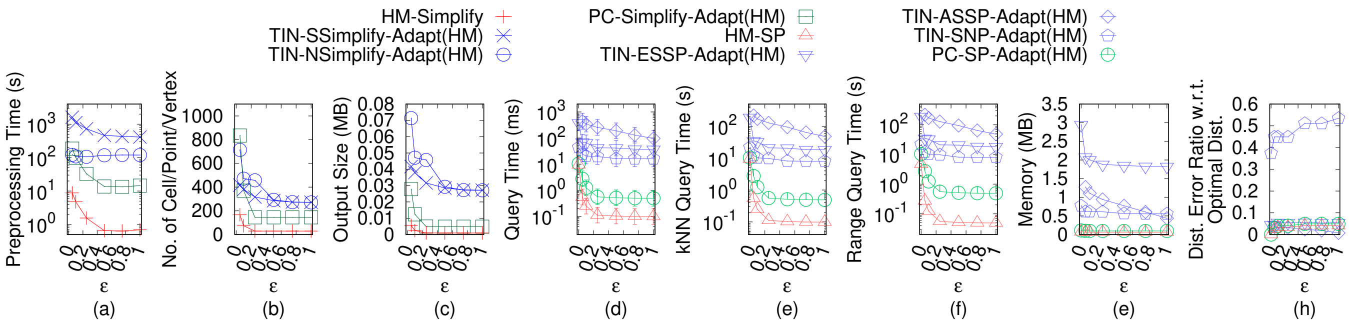


Figure 114: Effect of ϵ on RM_h-small height map dataset with optimal distance in distance error ratio calculation

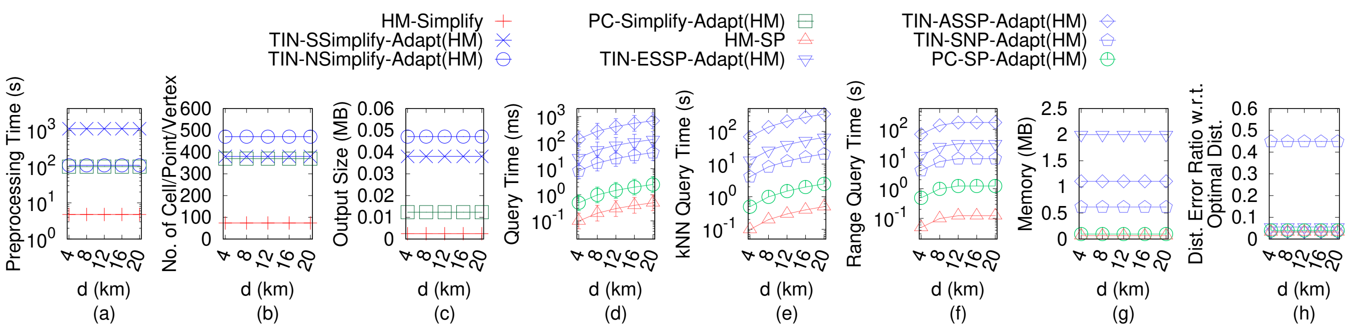


Figure 115: Effect of d on RM_h-small height map dataset with optimal distance in distance error ratio calculation

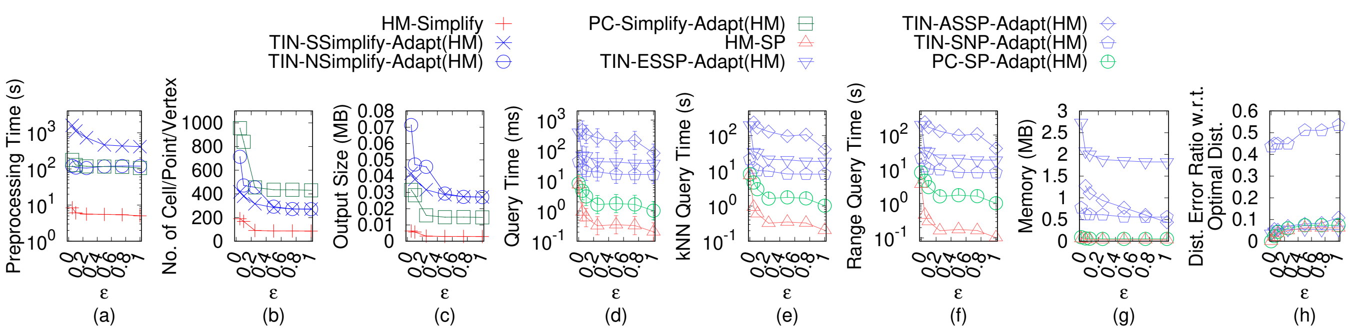


Figure 116: Effect of ϵ on BH_h-small height map dataset with optimal distance in distance error ratio calculation

Figure 131 and Figure 134, we tested 5 values of n in {5M, 10M, 15M, 20M, 25M} on GF_h, LM_h, RM_h, BH_h and EP_h dataset while

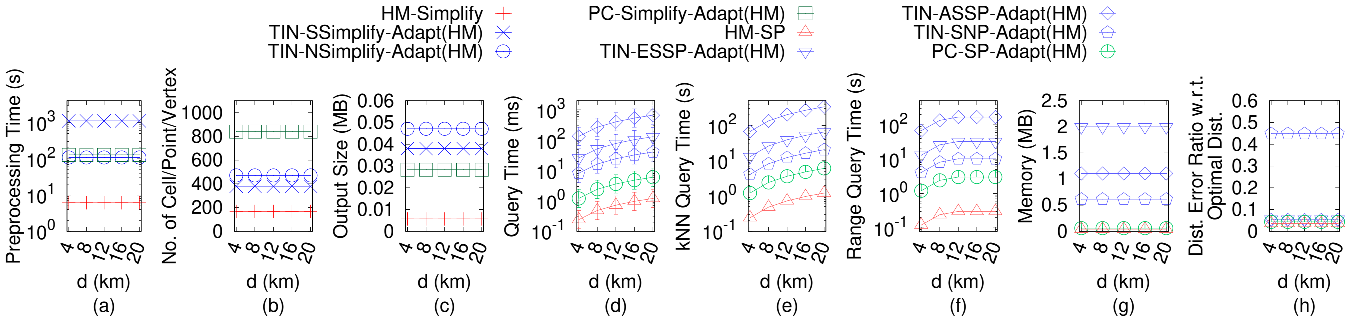


Figure 117: Effect of d on BH_h*-small* height map dataset with optimal distance in distance error ratio calculation

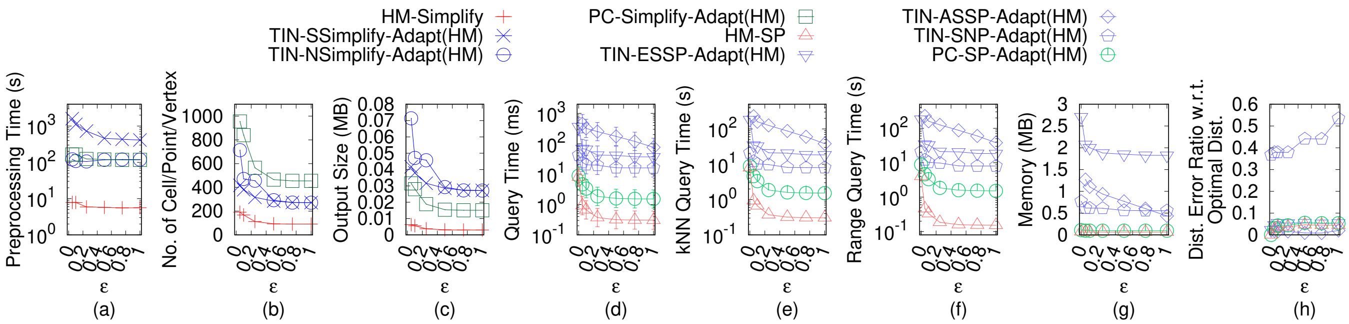


Figure 118: Effect of ϵ on EP_h*-small* height map dataset with optimal distance in distance error ratio calculation

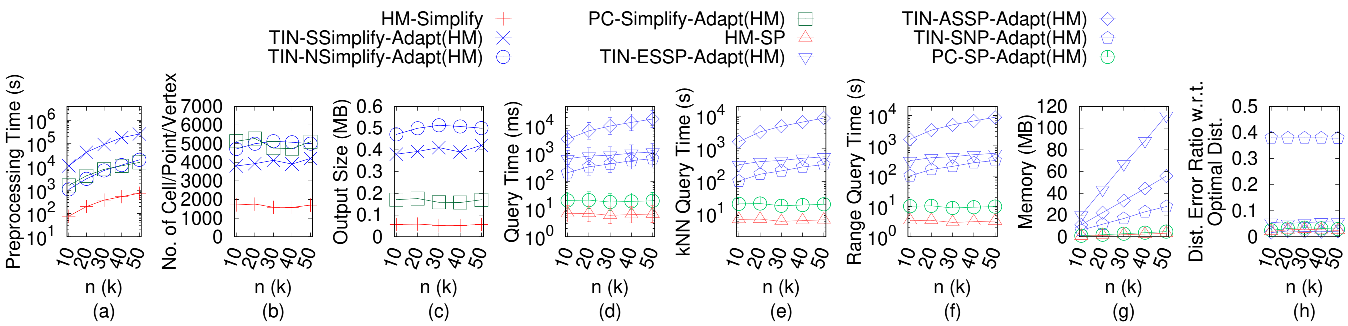


Figure 119: Effect of n on EP_h*-small* height map dataset with optimal distance in distance error ratio calculation

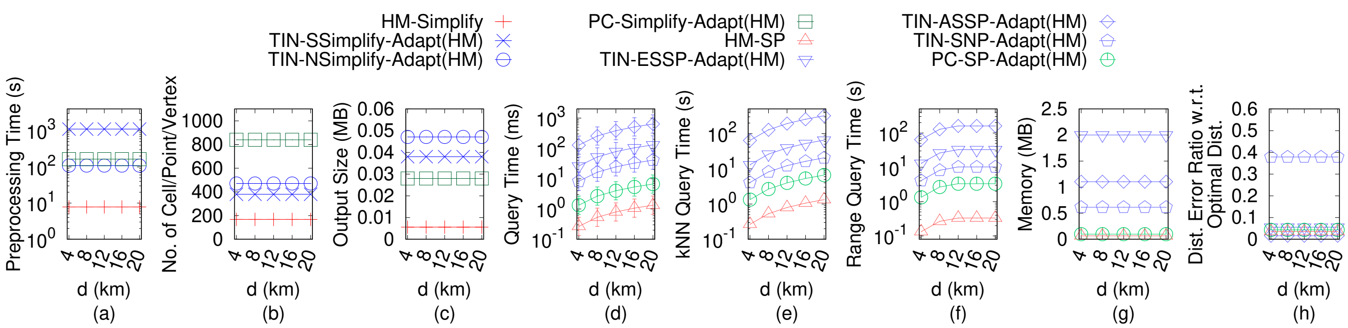


Figure 120: Effect of d on EP_h*-small* height map dataset with optimal distance in distance error ratio calculation

fixing ϵ at 0.25 for baseline comparisons. *HM-Simplify* (in terms of output size, i.e., 6.8MB) and *HM-SP* on the simplified height map (in

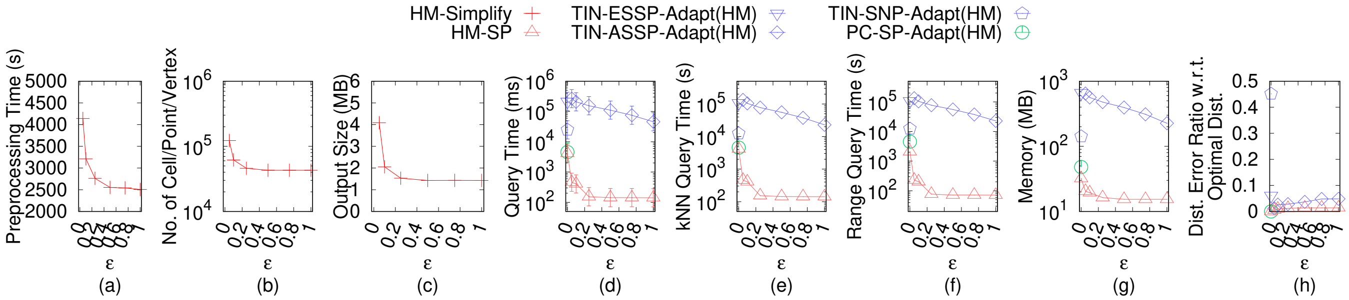


Figure 121: Effect of ϵ on GF_h height map dataset with optimal distance in distance error ratio calculation

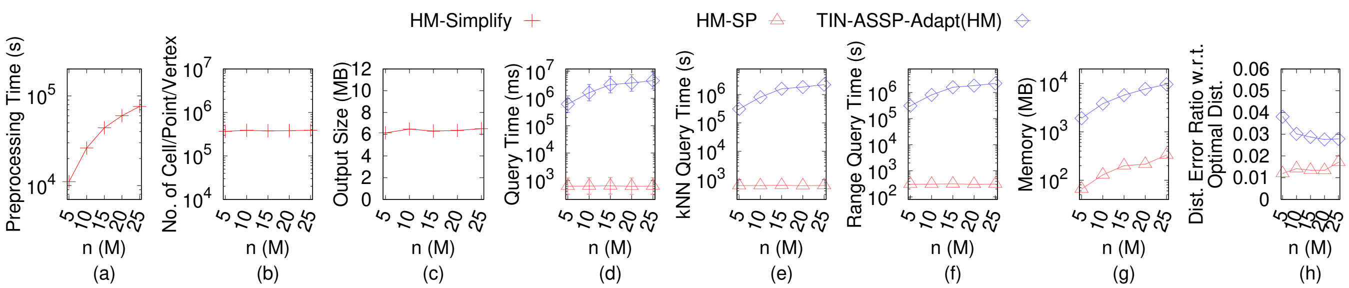


Figure 122: Effect of n on GF_h height map dataset with optimal distance in distance error ratio calculation

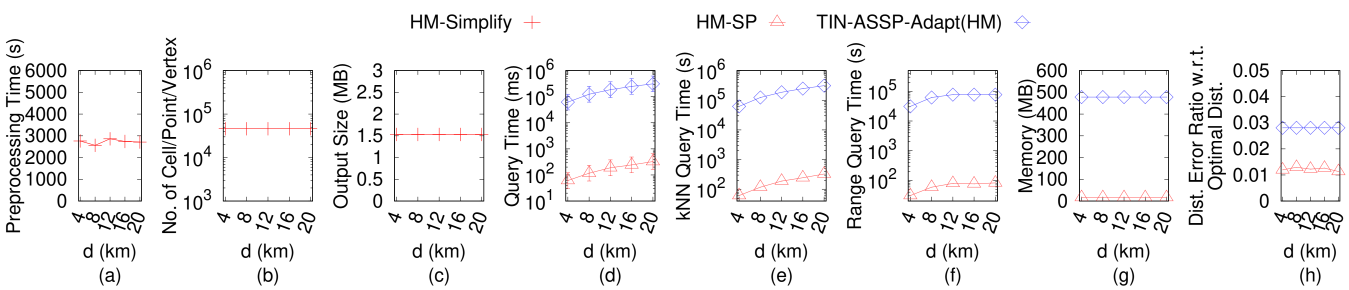


Figure 123: Effect of d on GF_h height map dataset with optimal distance in distance error ratio calculation

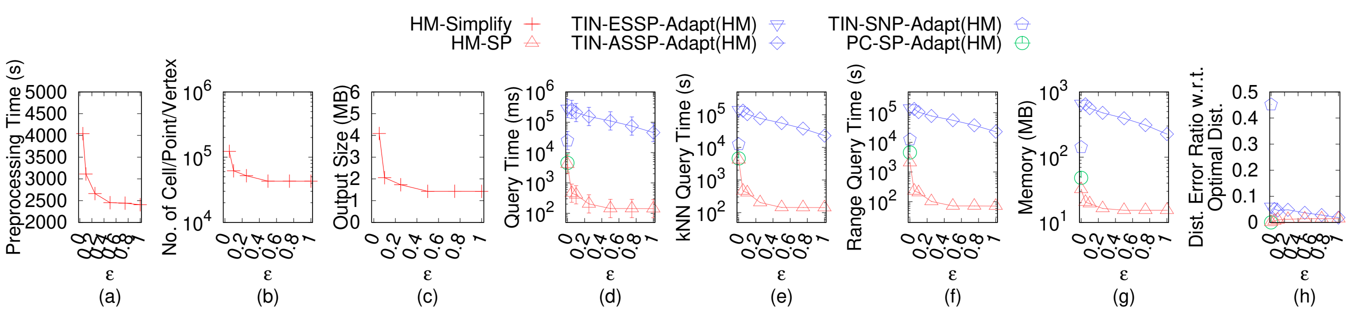


Figure 124: Effect of ϵ on LM_h height map dataset with optimal distance in distance error ratio calculation

terms of range query time, i.e., 310s ≈ 5.1 min, and memory usage, i.e., 310MB) are scalable on extremely large height map with 25M cells. Although the theoretical output size of *HM-Simplify* is only μ times smaller than the size of an original height map, it returns a simplified height map with an experimental size of 6.8MB from an original one with size 600MB and 25M cells, and performing range query on them with 500 objects takes 400s ≈ 6.7 min and 35,200s ≈ 9.8 hours, respectively. When n is smaller, i.e., datasets with looser density or fragmentation (since multi-resolution datasets have the same region), algorithms can run faster.

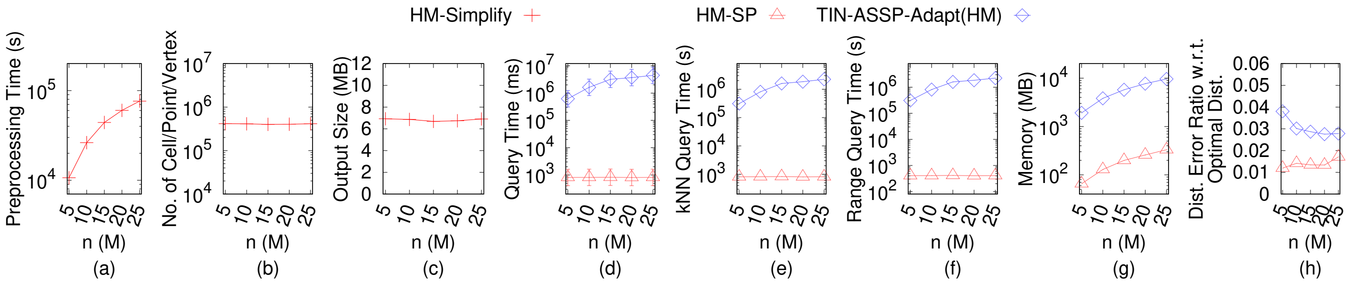


Figure 125: Effect of n on LM_h height map dataset with optimal distance in distance error ratio calculation

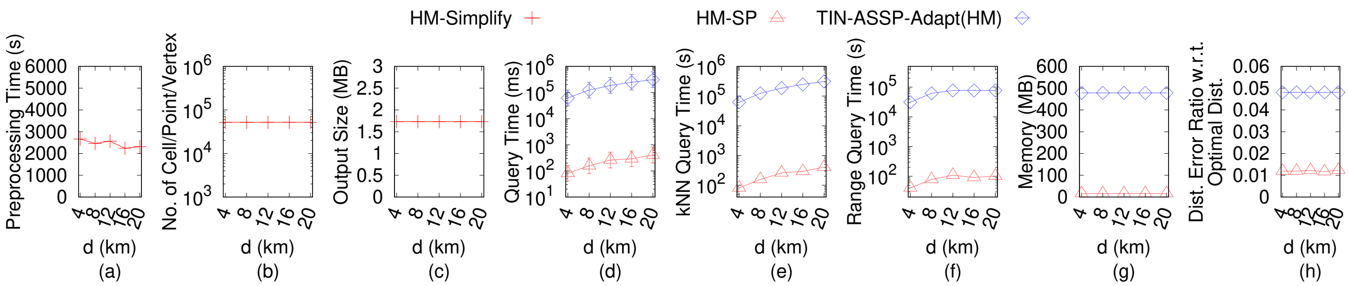


Figure 126: Effect of d on LM_h height map dataset with optimal distance in distance error ratio calculation

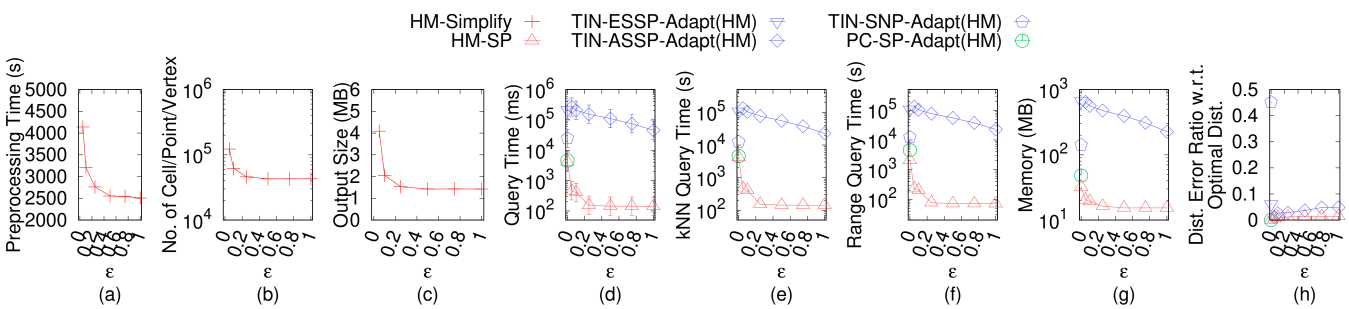


Figure 127: Effect of ϵ on RM_h height map dataset with optimal distance in distance error ratio calculation

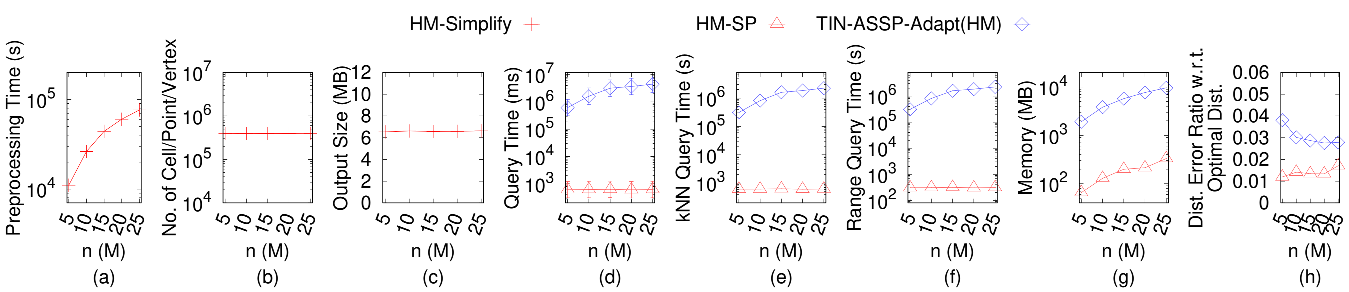


Figure 128: Effect of n on RM_h height map dataset with optimal distance in distance error ratio calculation

Effect of d: In Figure 111, Figure 113, Figure 115, Figure 117 and Figure 120, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on GF_h*-small*, LM_h*-small*, RM_h*-small*, BH_h*-small* and EP_h*-small* dataset while fixing ϵ at 0.1 and n at 1k for baseline comparisons. In Figure 123, Figure 126, Figure 129, Figure 132 and Figure 135, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on GF_h, LM_h, RM_h, BH_h and EP_h dataset while fixing ϵ at 0.25 and n at 0.5M for baseline comparisons. A smaller d reduces *kNN* and range query time, since our proximity query algorithm uses Dijkstra's algorithm once, we can terminate it earlier after visiting all query objects. As d increases, there is no upper bound on the increase in *kNN* query time (since we append the paths computed by Dijkstra's algorithm

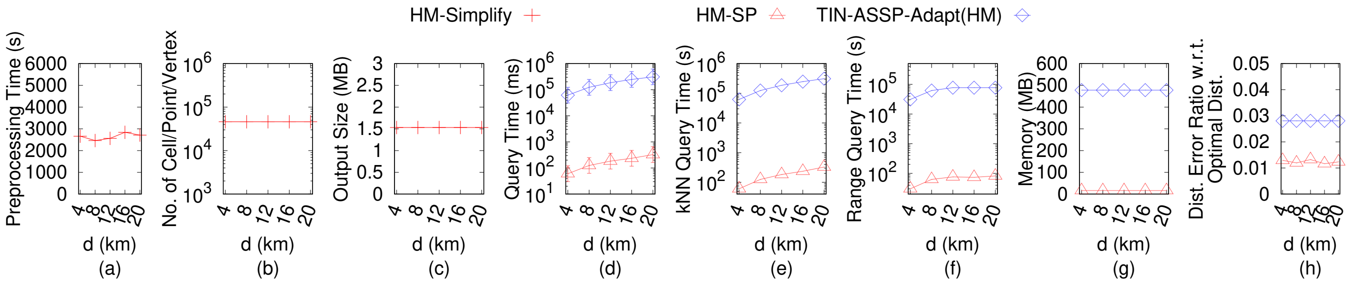


Figure 129: Effect of d on RM_h height map dataset with optimal distance in distance error ratio calculation

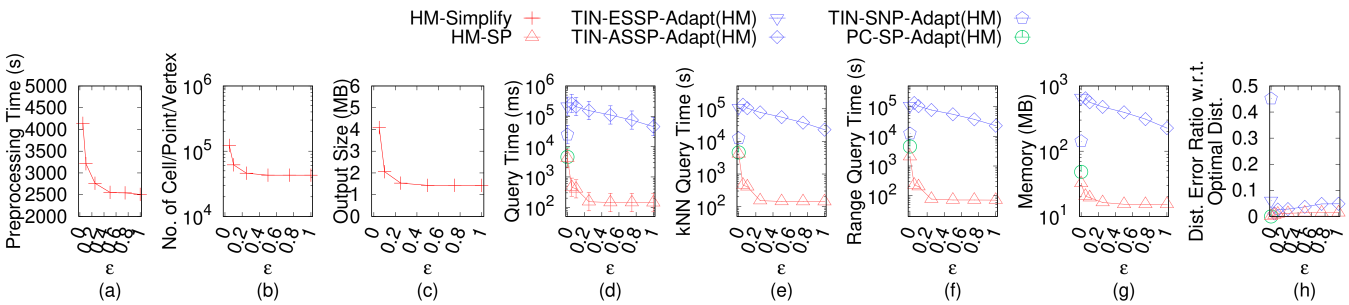


Figure 130: Effect of ϵ on BH_h height map dataset with optimal distance in distance error ratio calculation

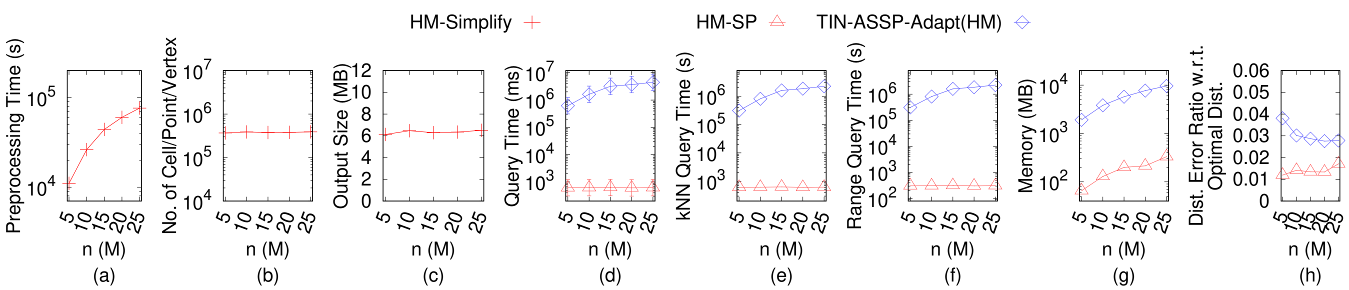


Figure 131: Effect of n on BH_h height map dataset with optimal distance in distance error ratio calculation

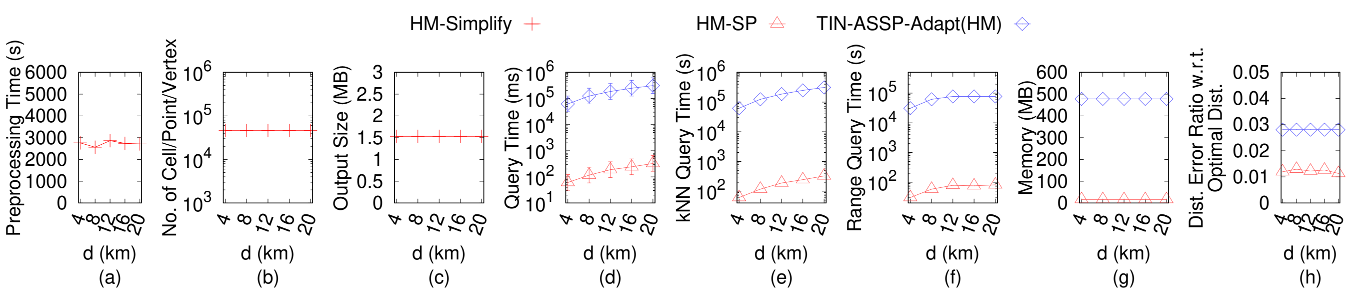


Figure 132: Effect of d on BH_h height map dataset with optimal distance in distance error ratio calculation

and the intra-paths as results, we cannot determine the distance correlations among these paths until we perform a linear scan, i.e., we terminate Dijkstra's algorithm based solely on d), but there is an upper bound on the increase in range query time (since we can also terminate Dijkstra's algorithm earlier if the searching distance exceeds r).

C.4.2 ***Ablation study for proximity query algorithms***. **Effect of k and r**: In Figure 136, Figure 138, Figure 140, Figure 142 and Figure 144, we tested 5 values of k in {200, 400, 600, 800, 1000} on GF_h, LM_h, RM_h, BH_h and EP_h dataset while fixing ϵ at 0.25 and n at 0.5M for ablation study. In Figure 137, Figure 139, Figure 141, Figure 143 and Figure 145, we tested 5 values of r in {2km, 4km,

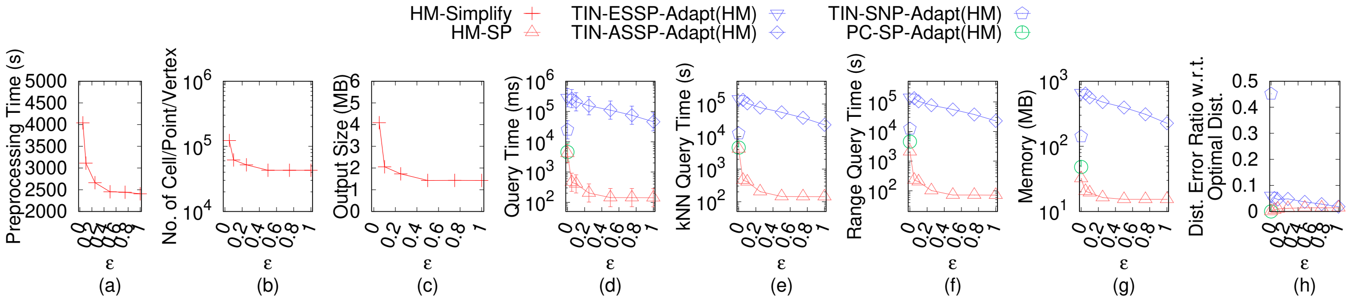


Figure 133: Effect of ϵ on EP_h height map dataset with optimal distance in distance error ratio calculation

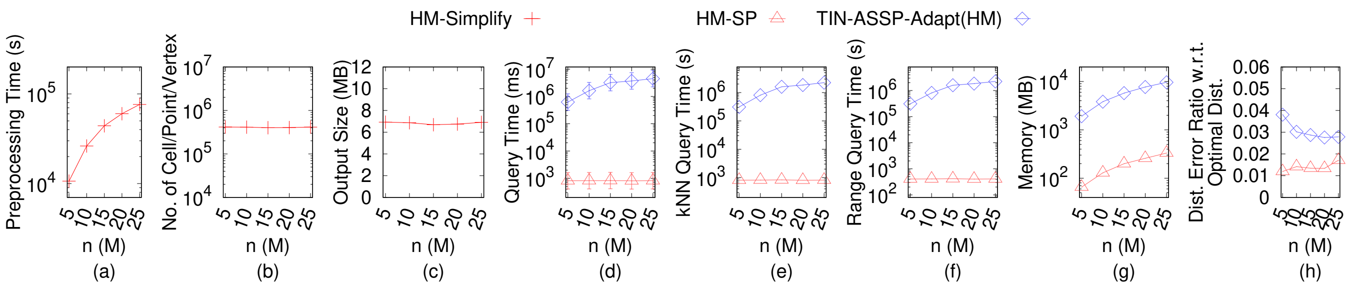


Figure 134: Effect of n on EP_h height map dataset with optimal distance in distance error ratio calculation

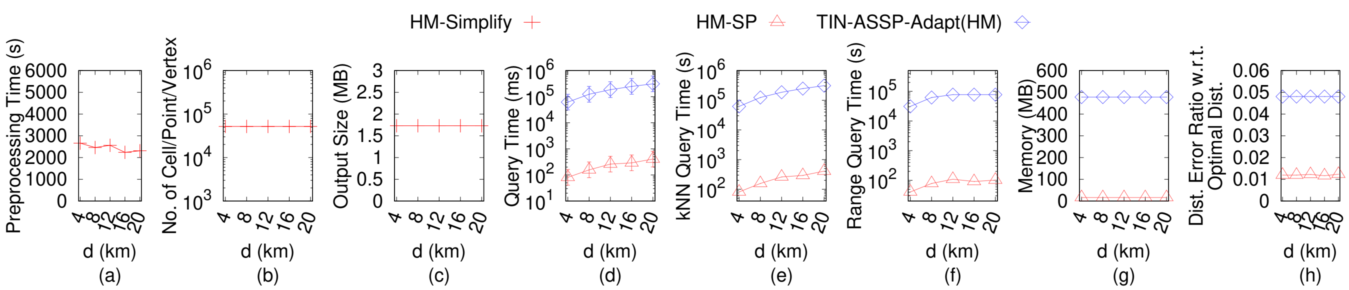


Figure 135: Effect of d on EP_h height map dataset with optimal distance in distance error ratio calculation

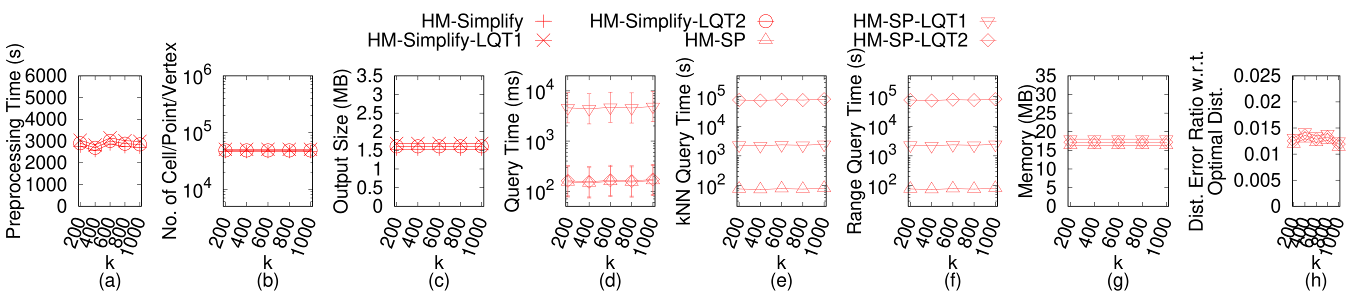


Figure 136: Ablation study for proximity query algorithms (effect of k on GF_h height map dataset) with optimal distance in distance error ratio calculation

6km, 8km, 10km} on GF_h, LM_h, RM_h, BH_h and EP_h dataset while fixing ϵ at 0.25 and n at 0.5M for ablation study for proximity query algorithms. On the simplified height map, *HM-SP* outperforms both *HM-SP-LQT1* and *HM-SP-LQT2*, since we use the efficient algorithm for querying. k does not affect *kNN* query time, since we append the paths computed by Dijkstra's algorithm and the intra-paths as the path results, and we do not know the distance correlations among these paths before we perform a linear scan on them. But, a

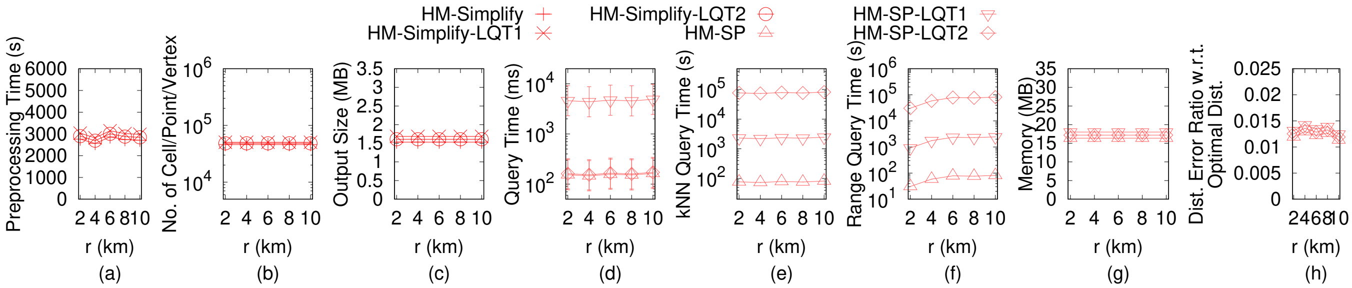


Figure 137: Ablation study for proximity query algorithms (effect of r on GF_h height map dataset) with optimal distance in distance error ratio calculation

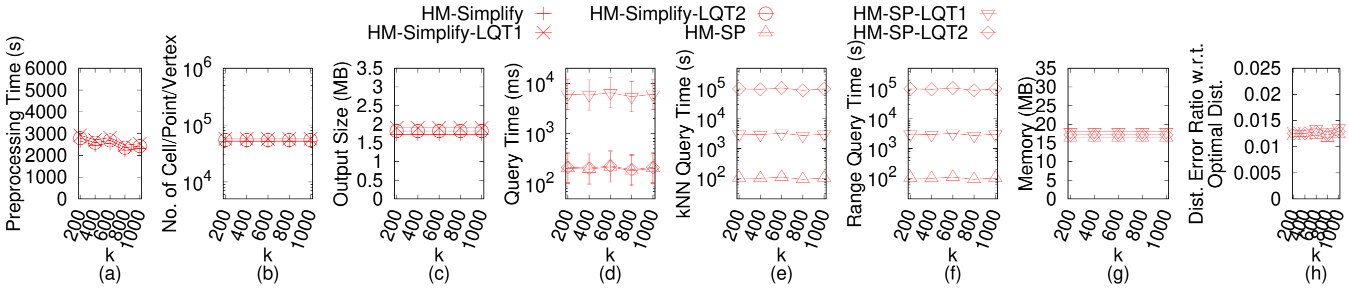


Figure 138: Ablation study for proximity query algorithms (effect of k on LM_h height map dataset) with optimal distance in distance error ratio calculation

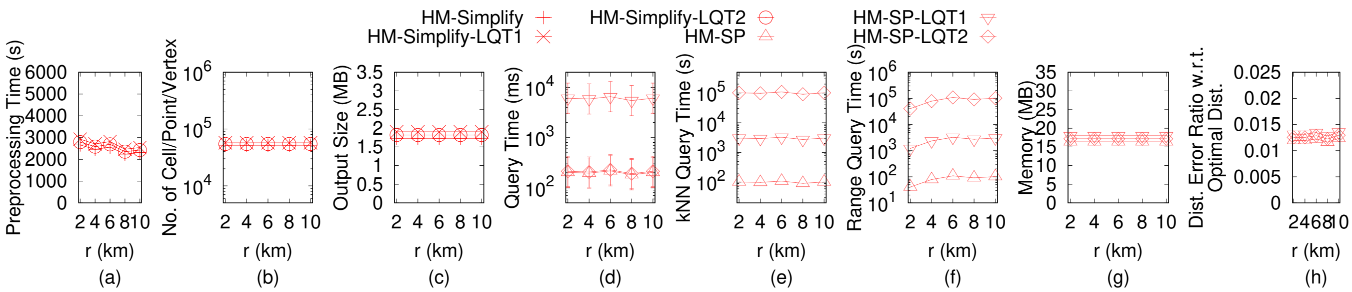


Figure 139: Ablation study for proximity query algorithms (effect of r on LM_h height map dataset) with optimal distance in distance error ratio calculation

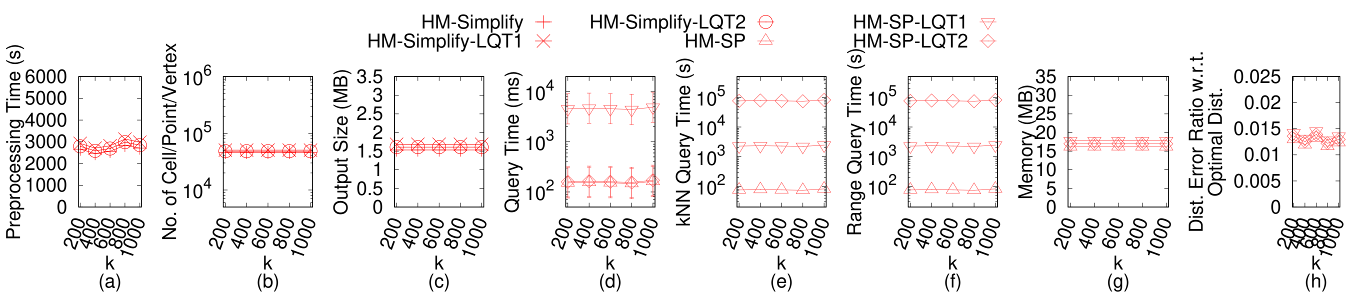


Figure 140: Ablation study for proximity query algorithms (effect of k on RM_h height map dataset) with optimal distance in distance error ratio calculation

smaller r reduces range query time, since we can terminate Dijkstra's algorithm earlier when the searching distance is larger than r.

C.4.3 ***Ablation study for simplification algorithms.*** In Figure 146, Figure 147, Figure 148, Figure 149 and Figure 150, we tested 6 values of ϵ in $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on *GF_h-small*, *LM_h-small*,

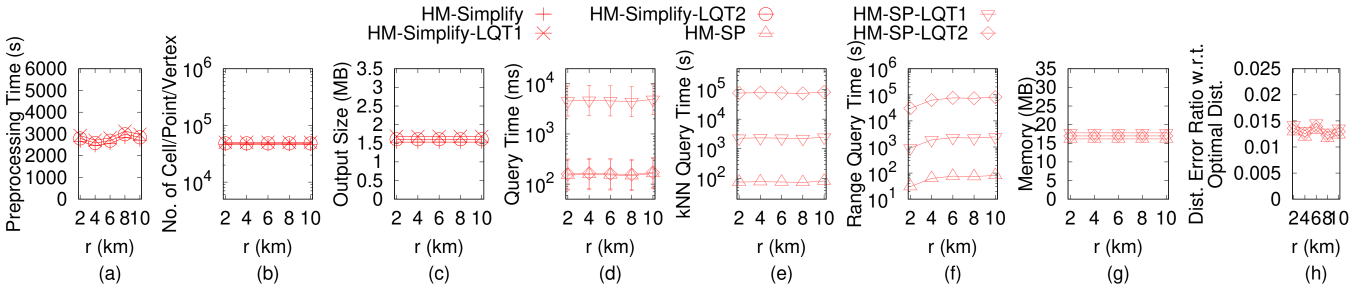


Figure 141: Ablation study for proximity query algorithms (effect of r on RM_h height map dataset) with optimal distance in distance error ratio calculation

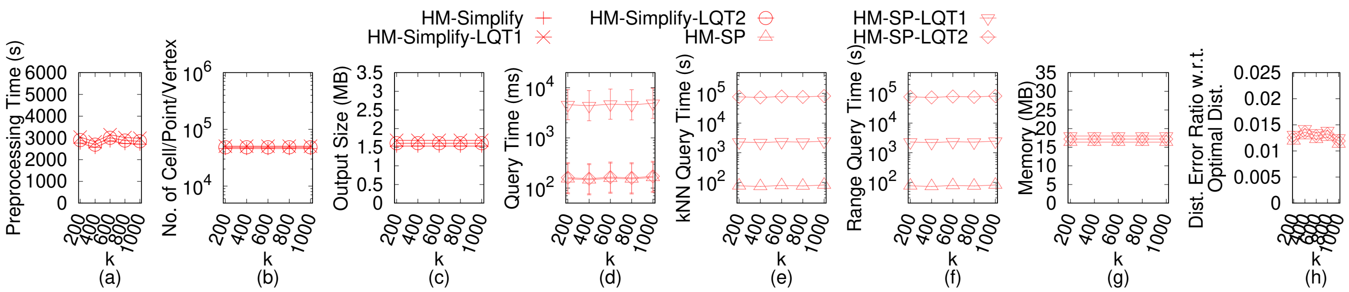


Figure 142: Ablation study for proximity query algorithms (effect of k on BH_h height map dataset) with optimal distance in distance error ratio calculation

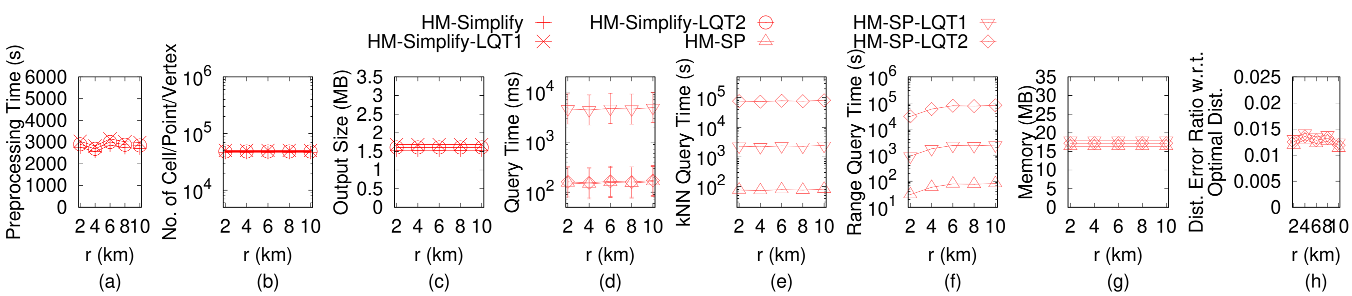


Figure 143: Ablation study for proximity query algorithms (effect of r on BH_h height map dataset) with optimal distance in distance error ratio calculation

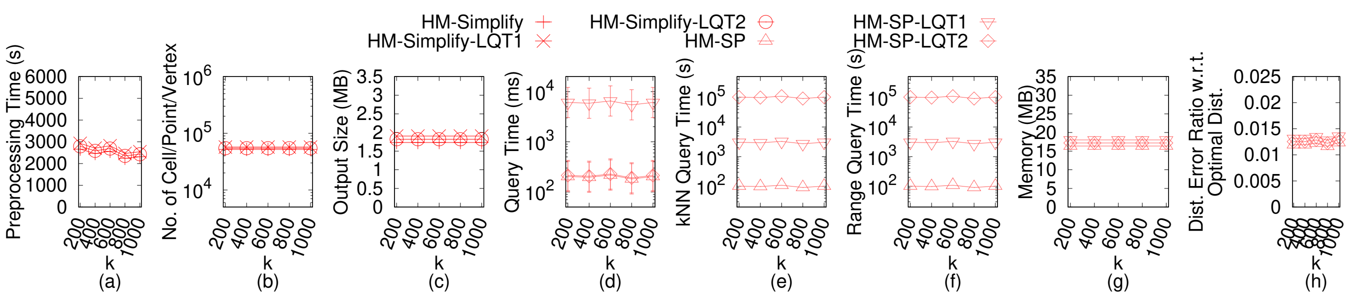


Figure 144: Ablation study for proximity query algorithms (effect of k on EP_h height map dataset) with optimal distance in distance error ratio calculation

RM_h-*small*, BH_h-*small* and EP_h-*small* dataset while fixing n at 0.5M for ablation study. *HM-Simplify* performs the best, showing the effectiveness of our merging and checking techniques, and our light structure compared with the heavy data structure in study [50]. Since *HM-Simplify-DS* has a large simplification time but *HM-SP-DS* on the simplified height map has a small shortest path query time,

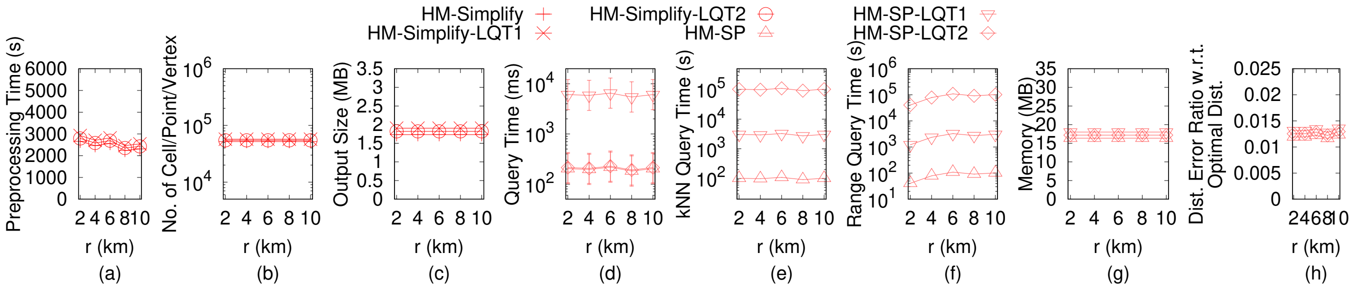


Figure 145: Ablation study for proximity query algorithms (effect of r on EP_h height map dataset) with optimal distance in distance error ratio calculation

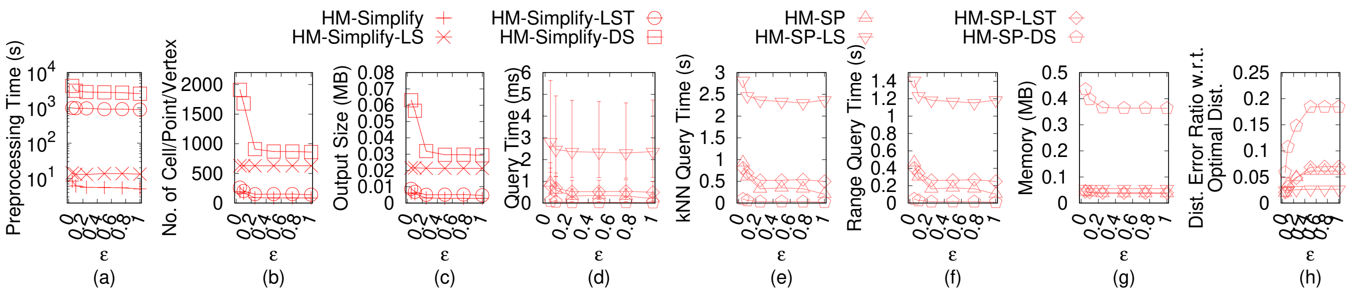


Figure 146: Ablation study for simplification algorithms on GF_h-*small* height map dataset with optimal distance in distance error ratio calculation

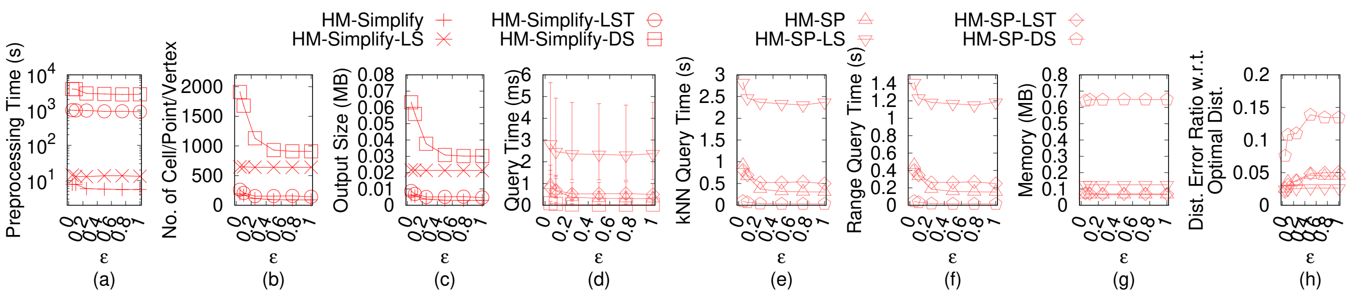


Figure 147: Ablation study for simplification algorithms on LM_h-*small* height map dataset with optimal distance in distance error ratio calculation

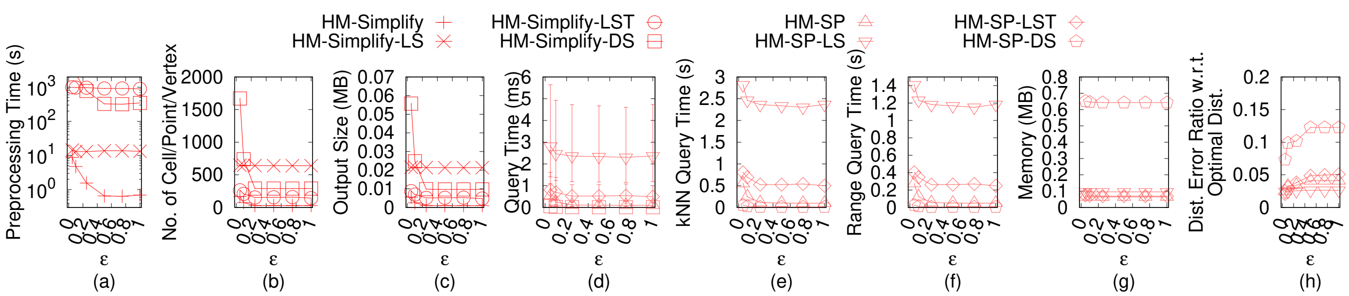


Figure 148: Ablation study for simplification algorithms on RM_h-*small* height map dataset with optimal distance in distance error ratio calculation

they are useful when we prioritize the shortest path query time over simplification time.

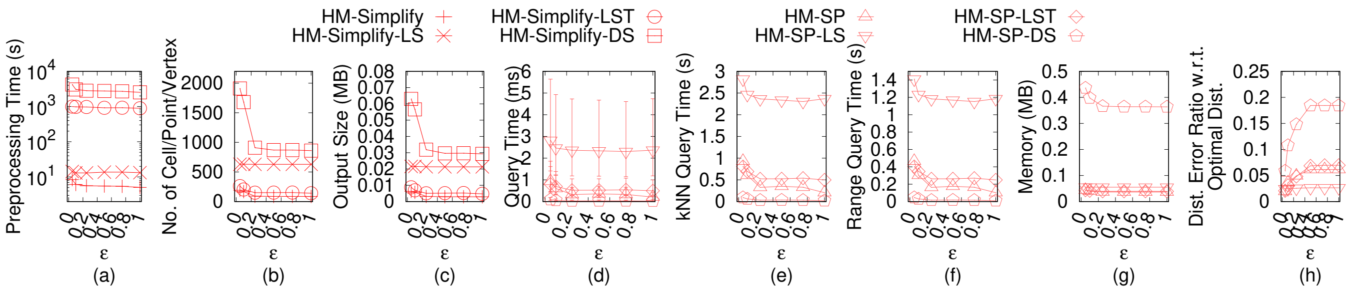


Figure 149: Ablation study for simplification algorithms on BH_h-*small* height map dataset with optimal distance in distance error ratio calculation

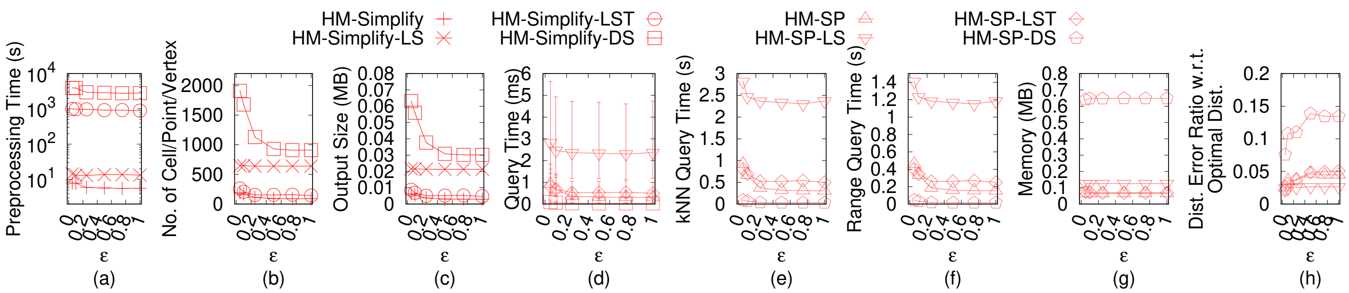


Figure 150: Ablation study for simplification algorithms on EP_h-*small* height map dataset with optimal distance in distance error ratio calculation

D PROOF

Proof of Theorem 2.1. Based on the Height Map Simplification Problem in Problem 1, we first need to find the Height Map Simplification Decision Problem in Problem 2.

Problem 2 (Height Map Simplification Decision Problem). *Given H, a non-negative integer i and ϵ, we want to find an ϵ-approximate simplified height map $\widetilde{H}$ of H with at most i cells.*

Then, the proof is by transforming Minimum T-Spanner Problem [26] in Problem 3, which is an *NP-complete* problem, to the Height Map Simplification Decision Problem.

Problem 3 (Minimum T-Spanner Decision Problem). *Given a graph G_{NPC} with a set of vertices $G_{NPC}.V$ and a set of edges $G_{NPC}.E$, a non-negative integer j and an error parameter t, we want to find a sub-graph $\widetilde{G}_{NPC}$ of G_{NPC} with at most j edges, such that for any pairs of vertices s and t in $G_{NPC}.V$, $|\Pi(\widetilde{s},\widetilde{t}|\widetilde{G}_{NPC})| \leq (1+\epsilon)|\Pi(s,t|G_{NPC})|$, where $\Pi(\widetilde{s},\widetilde{t}|\widetilde{G}_{NPC}$ (resp. $\Pi(s,t|G_{NPC})$) is the shortest path between s and t on $\widetilde{G}_{NPC}$ (resp. G_{NPC}).*

But, in order to do this transformation, we need the Height Map Graph Decision Simplification Problem in Problem 4. We transfer the Minimum T-Spanner Decision Problem to the Height Map Graph Simplification Decision Problem, and show that the Height Map Simplification Decision Problem is equivalent to the Height Map Graph Simplification Decision Problem.

Problem 4 (Height Map Graph Simplification Decision Problem). *Given a height map graph G of H, a non-negative integer i' and an error parameter ϵ, we want to find a simplified height map graph $\widetilde{G}$ of $\widetilde{H}$, with at most i' edges, such that for any pairs of vertices s and t in $G.V$, $(1-\epsilon)|\Pi(s,t|G)| \leq |\Pi(\widetilde{s},\widetilde{t}|\widetilde{G})| \leq (1+\epsilon)|\Pi(s,t|G)|$.*

We then construct a complete height map graph G_C, with a set of vertices $G_C.V$ and a set of edges $G_C.E$. In $G_C.V$, it contains all the vertices in G (i.e., the cell centers of H) and all possible new vertices in $\widetilde{G}$ (i.e., all possible added cells in $\widetilde{H}$). Figure 151 (a) shows a height map, Figure 151 (b) shows the complete height map graph in a 2D plane. In Figure 151 (b), (1) each orange point represents the vertex with the same x-, y- and z-coordinate values of the corresponding vertex in the original height map graph G, and (2) each green point represents vertices with the same x- and y-coordinate values of the possible new vertex, (3) the middle green point with an orange outline represents (i) the vertex with the same x-, y- and z-coordinate of the corresponding vertex in G, and (ii) vertices with the same x- and y-coordinate values of the possible new vertex. These points form a set of vertices in $G_C.V$. There is an edge connecting any pair of vertices in $G_C.V$, and these edges form $G_C.E$. Figure 151 (c) shows G_C with some possible vertices in 3D space (but we only show 5 of them for the sake of illustration). In addition, there should be an edge between each pair of points, we omit some of them for the sake of illustration. Clearly, G and $\widetilde{G}$ are both sub-graphs of G_C. Given a pair of vertices s and t in $G_C.V$, and the original height map graph G, let $\Pi(s,t|G_C)$ be the shortest path between s and t passing on G_C, and we set $\Pi(s,t|G_C) = \Pi(s,t|G)$. We can simply regard $\Pi(s,t|G_C)$ as a function, such that given s, t, and G, it can return a result. When s or t are on G_C, but not on G nor $\widetilde{G}$, we can simply regard $\Pi(s,t|G_C)$ as *NULL*.

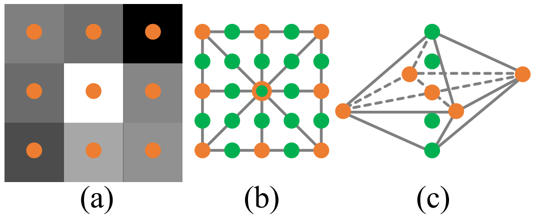


Figure 151: (a) A height map, (b) a complete height map graph in a 2D plane, and (c) a complete height map graph in a 3D space

The transformation from the Minimum T-Spanner Decision Problem to the Height Map Graph Simplification Decision Problem is as follows. We transfer G_{NPC} to G_C, transfer checking "can we find a sub-graph $\widetilde{G}_{NPC}$ of G_{NPC} with at most j edges, such that for any pairs of vertices s and t in $G_{NPC}.V$, $|\Pi(\widetilde{s},\widetilde{t}|\widetilde{G}_{NPC})| \leq (1+\epsilon)|\Pi(s,t|G_{NPC})|$" to "can we find a simplified height map graph $\widetilde{G}$ of G_C, with at most i' edges, such that for any pairs of vertices s and t in $G_C.V$, $(1-\epsilon)|\Pi(s,t|G)| = (1-\epsilon)|\Pi(s,t|G_C)| \leq |\Pi(\widetilde{s},\widetilde{t}|\widetilde{G})| \leq (1+\epsilon)|\Pi(s,t|G_C)| = (1+\epsilon)|\Pi(s,t|G)|$". Note that in the Height Map Graph Simplification Decision Problem, no matter whether the given graph is G or G_C, the given graph G or G_C will not affect the problem transformation, since the transformation is about the checking of the distance requirement, and given s and t, we have defined $\Pi(s,t|G_C) = \Pi(s,t|G)$. The transformation can be finished in polynomial time. Since the height map H and the height map graph G are equivalent, and i and i' can be any value, the Height Map Simplification Decision Problem is equivalent to the Height Map Graph Simplification Decision Problem. Thus, when the Height Map Graph Simplification Decision Problem is solved, the Height Map Simplification Decision Problem is solved equivalently, and the Minimum T-Spanner Decision Problem is also solved. Since the Minimum T-Spanner Decision Problem is *NP-complete*, the Height Map Simplification Decision Problem is *NP-hard*, and Height Map Simplification Problem is *NP-hard*. □

Lemma D.1. *Given a height map H, algorithm HM-Simplify returns a simplified height map $\widetilde{H}$ of H, such that for any pairs of points s_1 and t_1 on cells in C_{rema}, $(1-\epsilon)|\Pi(s_1,t_1|H)| \leq |\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_1,t_1|H)|$.*

Proof. We use mathematical induction to prove it. In algorithm *HM-Simplify*, even though it simplifies a height map using two different two simplification techniques, i.e., two cells merging and added cell with neighbor cells merging, the logic is the same, and we always perform the same distance checking, i.e., *R2R* distance checking, *R2D* distance checking and *D2D* distance checking. Thus, there is no need to distinguish these two simplification techniques in the following proof, and we regard any one step of the simplification process in these two simplification techniques as one equivalent iteration.

For the base case, we show that after the first simplification iteration, the inequality holds. Let C_{add} be the added cell in this iteration.

- Firstly, we show that $(1-\epsilon)|\Pi(s_1,t_1|H)| \leq |\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})|$. Along $\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})$ from s_1 to t_1 (resp. from t_1 to s_1), let $\overline{p}$ (resp. $\overline{q}$) be the point on cell that $\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})$ and the remaining neighbor cells of adjacent added cells of C_{add} intersects for the first time. We have $|\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})| = |\Pi(\widetilde{s_1},\widetilde{\overline{p}}|\widetilde{H})| + |\Pi(\widetilde{\overline{p}},\widetilde{\overline{q}}|\widetilde{H})| + |\Pi(\widetilde{\overline{q}},\widetilde{t_1}|\widetilde{H})|$. Since $\overline{p}$ and $\overline{q}$ are points on cells in C_{rema}, and their corresponding cells are remaining neighbor cells of adjacent added cells of C_{add}, we have $(1-\epsilon)|\Pi(\overline{p},\overline{q}|H)| \leq |\Pi(\widetilde{\overline{p}},\widetilde{\overline{q}}|\widetilde{H})|$ due to the *R2R* distance checking. Since s_1 and t_1 are points on cells in C_{rema}, and $\overline{p}$ and $\overline{q}$ are also points on cells in C_{rema}, and there is no difference between $\widetilde{H}$ and H (apart from the changes of C_{add}), we have $(1-\epsilon)|\Pi(s_1,\overline{p}|H)| = (1-\epsilon)|\Pi(\widetilde{s_1},\widetilde{\overline{p}}|\widetilde{H})| \leq |\Pi(\widetilde{s_1},\widetilde{\overline{p}}|\widetilde{H})|$ and $(1-\epsilon)|\Pi(\overline{q},t_1|H)| = (1-\epsilon)|\Pi(\widetilde{\overline{q}},\widetilde{t_1}|\widetilde{H})| \leq |\Pi(\widetilde{\overline{q}},\widetilde{t_1}|\widetilde{H})|$. Thus, we have $|\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})| = |\Pi(\widetilde{s_1},\widetilde{\overline{p}}|\widetilde{H})| + |\Pi(\widetilde{\overline{p}},\widetilde{\overline{q}}|\widetilde{H})| + |\Pi(\widetilde{\overline{q}},\widetilde{t_1}|\widetilde{H})| \geq (1-\epsilon)|\Pi(s_1,\overline{p}|H)| + (1-\epsilon)|\Pi(\overline{p},\overline{q}|H)| + (1-\epsilon)|\Pi(\overline{q},t_1|H)| \geq (1-\epsilon)|\Pi(s_1,t_1|H)|$.
- Secondly, we show that $|\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_1,t_1|H)|$. Along $\Pi(s_1,t_1|H)$ from s_1 to t_1 (resp. from t_1 to s_1), let $\overline{p}'$ (resp. $\overline{q}'$) be the point on cell that $\Pi(s_1,t_1|H)$ and the remaining neighbor cells of adjacent added cells of C_{add} intersects for the first time. We have $|\Pi(s_1,t_1|H)| = |\Pi(s_1,\overline{p}'|H)| + |\Pi(\overline{p}',\overline{q}'|H)| + |\Pi(\overline{q}',t_1|H)|$. Since $\overline{p}'$ and $\overline{q}'$ are points on cells in C_{rema}, and their corresponding cells are remaining neighbor cells of adjacent added cells of C_{add}, we have $|\Pi(\widetilde{\overline{p}}',\widetilde{\overline{q}}'|\widetilde{H})| \leq (1+\epsilon)|\Pi(\overline{p}',\overline{q}'|H)|$ due to the *R2R* distance checking. Since s_1 and t_1 are points on cells in C_{rema}, and $\overline{p}'$ and $\overline{q}'$ are also in C_{rema}, and there is no difference between $\widetilde{H}$ and H (apart from the changes of C_{add}), we have $|\Pi(\widetilde{s_1},\widetilde{\overline{p}}'|\widetilde{H})| \leq (1+\epsilon)|\Pi(\widetilde{s_1},\widetilde{\overline{p}}'|\widetilde{H})| = (1+\epsilon)|\Pi(s_1,\overline{p}'|H)|$ and $|\Pi(\widetilde{\overline{q}}',\widetilde{t_1}|\widetilde{H})| \leq (1+\epsilon)|\Pi(\widetilde{\overline{q}}',\widetilde{t_1}|\widetilde{H})| = (1+\epsilon)|\Pi(\overline{q}',t_1|H)|$. Thus, we have $(1+\epsilon)|\Pi(s_1,t_1|H)| = (1+\epsilon)|\Pi(s_1,\overline{p}'|H)| + (1+\epsilon)|\Pi(\overline{p}',\overline{q}'|H)| + (1+\epsilon)|\Pi(\overline{q}',t_1|H)| \geq |\Pi(\widetilde{s_1},\widetilde{\overline{p}}'|\widetilde{H})| + |\Pi(\widetilde{\overline{p}}',\widetilde{\overline{q}}'|\widetilde{H})| + |\Pi(\widetilde{\overline{q}}',\widetilde{t_1}|\widetilde{H})| \geq |\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})|$.

For the hypothesis case, assume that after the i-th simplification iteration, for any pairs of points s_1 and t_1 on cells in C_{rema}, we have $(1-\epsilon)|\Pi(s_1,t_1|H)| \leq |\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_1,t_1|H)|$. We show that for the $(i+1)$-th simplification iteration, the inequality holds. Let C_{add} be the added cell in this iteration.

- Firstly, we show that $(1-\epsilon)|\Pi(s_1,t_1|H)| \leq |\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})|$. Along $\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})$ from s_1 to t_1 (resp. from t_1 to s_1), let $\overline{p}$ (resp. $\overline{q}$) be the point on cell that $\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})$ and the remaining neighbor cells of adjacent added cells of C_{add} intersects for the first time. We have $|\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})| = |\Pi(\widetilde{s_1},\widetilde{\overline{p}}|\widetilde{H})| + |\Pi(\widetilde{\overline{p}},\widetilde{\overline{q}}|\widetilde{H})| + |\Pi(\widetilde{\overline{q}},\widetilde{t_1}|\widetilde{H})|$. Since $\overline{p}$ and $\overline{q}$ are points on cells in C_{rema}, and their corresponding cells are remaining neighbor cells of adjacent added cells of C_{add}, we have $(1-\epsilon)|\Pi(\overline{p},\overline{q}|H)| \leq |\Pi(\widetilde{\overline{p}},\widetilde{\overline{q}}|\widetilde{H})|$ due to the *R2R* distance checking. Since s_1 and t_1 are points on cells in C_{rema}, and $\overline{p}$ and $\overline{q}$ are also points on cells in C_{rema}, we have $(1-\epsilon)|\Pi(s_1,\overline{p}|H)| \leq |\Pi(\widetilde{s_1},\widetilde{\overline{p}}|\widetilde{H})|$ and $(1-\epsilon)|\Pi(\overline{q},t_1|H)| \leq |\Pi(\widetilde{\overline{q}},\widetilde{t_1}|\widetilde{H})|$ due to the *R2R* distance checking after the i-th simplification iteration. Thus, we have $|\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})| = |\Pi(\widetilde{s_1},\widetilde{\overline{p}}|\widetilde{H})| + |\Pi(\widetilde{\overline{p}},\widetilde{\overline{q}}|\widetilde{H})| + |\Pi(\widetilde{\overline{q}},\widetilde{t_1}|\widetilde{H})| \geq (1-\epsilon)|\Pi(s_1,\overline{p}|H)| + (1-\epsilon)|\Pi(\overline{p},\overline{q}|H)| + (1-\epsilon)|\Pi(\overline{q},t_1|H)| \geq (1-\epsilon)|\Pi(s_1,t_1|H)|$.
- Secondly, we show that $|\Pi(\widetilde{s_1},\widetilde{t_1}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_1,t_1|H)|$. Along $\Pi(s_1,t_1|H)$ from s_1 to t_1 (resp. from t_1 to s_1), let $\overline{p}'$ (resp.

$\overline{q}'$) be the point on cell that $\Pi(s_1, t_1|H)$ and the remaining neighbor cells of adjacent added cells of C_{add} intersects for the first time. We have $|\Pi(s_1, t_1|H)| = |\Pi(s_1, \overline{p}'|H)| + |\Pi(\overline{p}', \overline{q}'|H)| + |\Pi(\overline{q}', t_1|H)|$. Since $\overline{p}'$ and $\overline{q}'$ are points on cells in C_{rema}, and their corresponding cells are remaining neighbor cells of adjacent added cells of C_{add}, we have $|\Pi(\widetilde{\overline{p}'}, \widetilde{\overline{q}'}|\widetilde{H})| \le (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|H)|$ due to the *R2R* distance checking. Since s_1 and t_1 are points on cells in C_{rema}, and $\overline{p}'$ and $\overline{q}'$ are also points on cells in C_{rema}, we have $|\Pi(\widetilde{s_1}, \widetilde{\overline{p}'}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_1, \overline{p}'|H)|$ and $|\Pi(\widetilde{\overline{q}'}, \widetilde{t_1}|\widetilde{H})| \le (1+\epsilon)|\Pi(\overline{q}', t_1|H)|$ due to the *R2R* distance checking after the i-th simplification iteration. Thus, we have $(1+\epsilon)|\Pi(s_1, t_1|H)| = (1+\epsilon)|\Pi(s_1, \overline{p}'|H)| + (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|H)| + (1+\epsilon)|\Pi(\overline{q}', t_1|H)| \ge |\Pi(\widetilde{s_1}, \widetilde{\overline{p}'}|\widetilde{H})| + |\Pi(\widetilde{\overline{p}'}, \widetilde{\overline{q}'}|\widetilde{H})| + |\Pi(\widetilde{\overline{q}'}, \widetilde{t_1}|\widetilde{H})| \ge |\Pi(\widetilde{s_1}, \widetilde{t_1}|\widetilde{H})|$.

Thus, we have proved that for any pairs of points s_1 and t_1 on cells in C_{rema}, $(1-\epsilon)|\Pi(s_1, t_1|H)| \le |\Pi(\widetilde{s_1}, \widetilde{t_1}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_1, t_1|H)|$. □

Lemma D.2. *Given a height map H, algorithm HM-Simplify returns a simplified height map $\widetilde{H}$ of H, such that for any pairs of points s_2 on cells in C_{rema} and points t_2 on cells in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_2, t_2|H)| \le |\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_2, t_2|H)|$.*

Proof. We use mathematical induction to prove it. Similar to the proof of Lemma D.1, there is no need to distinguish two simplification techniques, and we regard any one step of the simplification process in the two simplification techniques as one equivalent iteration.

For the base case, we show that after the first simplification iteration, the inequality holds. Let C_{add} be the added cell in this iteration. Since this is the first iteration, there are no other deleted cells except the cells belonging to C_{add}, we just need to show that the inequality holds when t_2 is any one of the points of the deleted cells belong to adjacent added cells of C_{add}.

- Firstly, we show that $(1-\epsilon)|\Pi(s_2, t_2|H)| \le |\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})|$. Along $\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})$ from $\widetilde{t_2}$ to $\widetilde{s_2}$, let $\overline{m}$ be the point on cell that $\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})$ and the remaining neighbor cells of adjacent added cells of C_{add} intersects for the first time. We have $|\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})| = |\Pi(\widetilde{s_2}, \widetilde{\overline{m}}|\widetilde{H})| + |\Pi(\widetilde{\overline{m}}, \widetilde{t_2}|\widetilde{H})|$. Since $\overline{m}$ is a point on cell in C_{rema}, which is a remaining neighbor cell of adjacent added cells of C_{add}, and t_2 is a point on cell in $C - C_{rema}$, we have $(1-\epsilon)|\Pi(\overline{m}, t_2|H)| \le |\Pi(\widetilde{\overline{m}}, \widetilde{t_2}|\widetilde{H})|$ due to the *R2D* distance checking. Since s_2 and $\overline{m}$ are points on cells in C_{rema}, we have $(1-\epsilon)|\Pi(s_2, \overline{m}|H)| \le |\Pi(\widetilde{s_2}, \widetilde{\overline{m}}|\widetilde{H})|$ from Lemma D.1. Thus, we have $|\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})| = |\Pi(\widetilde{s_2}, \widetilde{\overline{m}}|\widetilde{H})| + |\Pi(\widetilde{\overline{m}}, \widetilde{t_2}|\widetilde{H})| \ge (1-\epsilon)|\Pi(s_2, \overline{m}|H)| + (1-\epsilon)|\Pi(\overline{m}, t_2|H)| \ge (1-\epsilon)|\Pi(s_2, t_2|H)|$.
- Secondly, we show that $|\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_2, t_2|H)|$. Along $\Pi(s_2, t_2|H)$ from $\widetilde{t_2}$ to $\widetilde{s_2}$, let $\overline{m}'$ be the point on cell that $\Pi(s_2, t_2|H)$ and the remaining neighbor cells of adjacent added cells of C_{add} intersects for the first time. We have $|\Pi(s_2, t_2|H)| = |\Pi(s_2, \overline{m}'|H)| + |\Pi(\overline{m}', t_2|H)|$. Since $\overline{m}'$ is a point on cell in C_{rema}, which is a remaining neighbor cell of adjacent added cells of C_{add}, and t_2 is a point on cell in $C - C_{rema}$, we have $|\Pi(\widetilde{\overline{m}}, \widetilde{t_2}|\widetilde{H})| \le (1+\epsilon)|\Pi(\overline{m}, t_2|H)|$ due to the *R2D* distance checking. Since s_2 and $\overline{m}'$ are are points on cells in C_{rema}, we have $|\Pi(\widetilde{s_2}, \widetilde{\overline{m}'}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_2, \overline{m}'|H)|$ from Lemma D.1. Thus, we have $(1+\epsilon)|\Pi(s_2, t_2|H)| = (1+\epsilon)|\Pi(s_2, \overline{m}'|H)| + (1+\epsilon)|\Pi(\overline{m}', t_2|H)| \ge |\Pi(\widetilde{s_2}, \widetilde{\overline{m}'}|\widetilde{H})| + |\Pi(\widetilde{\overline{m}'}, \widetilde{t_2}|\widetilde{H})| \ge |\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})|$.

For the hypothesis case, assume that after the i-th simplification iteration, for any pairs of points s_2 on cells in C_{rema} and points t_2 on cells in $C - C_{rema}$, we have $(1-\epsilon)|\Pi(s_2, t_2|H)| \le |\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_2, t_2|H)|$. We show that for the $(i+1)$-th simplification iteration, the inequality holds. Let C_{add} be the added cell in this iteration. Since the difference of $\widetilde{H}$ after the i-th simplification iteration and the $(i+1)$-th simplification iteration is due to the changes of C_{add}, we just need to show that the inequality holds when t_2 is any one of the points of the deleted cells belong to adjacent added cells C_{add}. The proof is exactly the same as in the base case.

Thus, we have proved that for any pairs of points s_2 on cells in C_{rema} and points t_2 on cells in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_2, t_2|H)| \le |\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_2, t_2|H)|$. □

Lemma D.3. *Given a height map H, algorithm HM-Simplify returns a simplified height map $\widetilde{H}$ of H, such that for any pairs of points s_3 and t_3 on cells in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_3, t_3|H)| \le |\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_3, t_3|H)|$.*

Proof. Similar to the proof of Lemma D.1, there is no need to distinguish two simplification techniques, and we regard any one step of the simplification process in the two simplification techniques as one equivalent iteration. There are two sub-cases. (1) $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ does not pass on cells in C_{rema}. (2) $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ passes on cells in C_{rema}.

(1) We prove the first sub-case, i.e., $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ does not pass on cells in C_{rema}. We use mathematical induction to prove it.

For the base case, we show that after the first simplification iteration, the inequality holds. Let C_{add} be the added cell in this iteration. Since this is the first iteration, there are no other deleted cells except the cells belonging to C_{add}, we just need to show that the inequality holds when s_3 and t_3 are any one of the points of the deleted cells belong to C_{add}. Due to the *D2D* distance checking, we have $(1-\epsilon)|\Pi(s_3, t_3|H)| \le |\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_3, t_3|H)|$.

For the hypothesis case, assume that after the i-th simplification iteration, for any pairs of points s_3 and t_3 on cells in $C - C_{rema}$, we have $(1-\epsilon)|\Pi(s_3, t_3|H)| \le |\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_3, t_3|H)|$. We show that for the $(i+1)$-th simplification iteration, the inequality holds. Let C_{add} be the added cell in this iteration. Since the difference of $\widetilde{H}$ after the i-th simplification iteration and the $(i+1)$-th simplification iteration is due to the changes of C_{add}, we just need to show that the inequality holds when t_3 is any one of the points of the deleted cells belong to adjacent added cells of C_{add}. The proof is exactly the same as in the base case.

Thus, we have proved that for any pairs of points s_3 and t_3 on cells in $C - C_{rema}$, when $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ does not pass on cells in C_{rema}, $(1-\epsilon)|\Pi(s_3, t_3|H)| \le |\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| \le (1+\epsilon)|\Pi(s_3, t_3|H)|$.

(2) We prove the second sub-case, i.e., $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ passes on cells in C_{rema}. We use the Lemma D.1 and Lemma D.2 to prove it.

- Firstly, we show that $(1-\epsilon)|\Pi(s_3, t_3|H)| \le |\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})|$. Along $\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})$ from $\widetilde{s_3}$ to $\widetilde{t_3}$ (resp. from $\widetilde{t_3}$ to $\widetilde{s_3}$), let $\overline{p}$ (resp. $\overline{q}$) be the point on cell that $\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})$ and the remaining neighbor cells of adjacent added cells of $O^{-1}(s_3)$ (resp. $O^{-1}(t_3)$) intersects for the first time. We have $|\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| = |\Pi_1(\widetilde{s_3}, \widetilde{\overline{p}}|\widetilde{H})| + |\Pi_2(\widetilde{\overline{p}}, \widetilde{\overline{q}}|\widetilde{H})| + |\Pi_1(\widetilde{\overline{q}}, \widetilde{t_3}|\widetilde{H})|$. Since $\overline{p}$ and $\overline{q}$ are points on cells in C_{rema}, we have $(1-\epsilon)|\Pi(\overline{p}, \overline{q}|H)| \leq |\Pi_2(\widetilde{\overline{p}}, \widetilde{\overline{q}}|\widetilde{H})|$ by Lemma D.1. Since s_3 and t_3 are points on cells in $C - C_{rema}$, and $\overline{p}$ and $\overline{q}$ are in C_{rema}, we have $(1-\epsilon)|\Pi(s_3, \overline{p}|H)| \leq |\Pi_1(\widetilde{s_3}, \widetilde{\overline{p}}|\widetilde{H})|$ and $(1-\epsilon)|\Pi(\overline{q}, t_3|H)| \leq |\Pi_1(\widetilde{\overline{q}}, \widetilde{t_3}|\widetilde{H})|$ by Lemma D.2. Thus, we have $|\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| = |\Pi_1(\widetilde{s_3}, \widetilde{\overline{p}}|\widetilde{H})| + |\Pi_2(\widetilde{\overline{p}}, \widetilde{\overline{q}}|\widetilde{H})| + |\Pi_1(\widetilde{\overline{q}}, \widetilde{t_3}|\widetilde{H})| \geq (1-\epsilon)|\Pi(s_3, \overline{p}|H)| + (1-\epsilon)|\Pi(\overline{p}, \overline{q}|H)| + (1-\epsilon)|\Pi(\overline{q}, t_3|H)| \geq (1-\epsilon)|\Pi(s_3, t_3|H)|$.

- Secondly, we show that $|\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_3, t_3|H)|$. Along $\Pi(s_3, t_3|H)$ from s_3 to t_3 (resp. from t_3 to s_3), let $\overline{p}'$ (resp. $\overline{q}'$) be the point on cell that $\Pi(s_3, t_3|H)$ and the remaining neighbor cells of adjacent added cells of $O^{-1}(s_3)$ (resp. $O^{-1}(t_3)$) intersects for the first time. We have $|\Pi(s_3, t_3|H)| = |\Pi(s_3, \overline{p}'|H)| + |\Pi(\overline{p}', \overline{q}'|H)| + |\Pi(\overline{q}', t_3|H)|$. Since $\overline{p}'$ and $\overline{q}'$ are points on cells in C_{rema}, we have $|\Pi_2(\widetilde{\overline{p}'}, \widetilde{\overline{q}'}|\widetilde{H})| \leq (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|H)|$ by Lemma D.1. Since s_3 and t_3 are points on cells in $C - C_{rema}$, and $\overline{p}'$ and $\overline{q}'$ are points on cells in C_{rema}, we have $|\Pi_1(\widetilde{s_3}, \widetilde{\overline{p}'}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_3, \overline{p}'|H)|$ and $|\Pi_1(\widetilde{\overline{q}'}, \widetilde{t_3}|\widetilde{H})| \leq (1+\epsilon)|\Pi(\overline{q}', t_3|H)|$ by Lemma D.2. Thus, we have $(1+\epsilon)|\Pi(s_3, t_3|H)| = (1+\epsilon)|\Pi(s_3, \overline{p}'|H)| + (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|H)| + (1+\epsilon)|\Pi(\overline{q}', t_3|H)| \geq |\Pi_1(\widetilde{s_3}, \widetilde{\overline{p}'}|\widetilde{H})| + |\Pi_2(\widetilde{\overline{p}'}, \widetilde{\overline{q}'}|\widetilde{H})| + |\Pi_1(\widetilde{\overline{q}'}, \widetilde{t_3}|\widetilde{H})| \geq |\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})|$.

Thus, we have proved that for any pairs of points s_3 and t_3 on cells in $C - C_{rema}$, when $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ passes on cells in C_{rema}, $(1-\epsilon)|\Pi(s_3, t_3|H)| \leq |\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_3, t_3|H)|$.

In general, we have proved that for any pairs of points s_3 and t_3 on cells in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_3, t_3|H)| \leq |\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_3, t_3|H)|$. ☐

Proof of Theorem 4.1. Firstly, we prove the simplification time. In each simplification iteration of the *R2R*, *R2D* and *D2D* distance checking, since we only check the cells related to the neighbor cells of adjacent added cells of an added cell, there are $O(1)$ such cells. Since we use Dijkstra's algorithm in $O(n \log n)$ time for distance calculation, the distance checking needs $O(1)$ time. In both of the two cells merging and added cell with neighbor cells merging, we always expand by one cell in four directions. Let i be the total number of iterations we need to perform. That is, we keep removing $1 \times 2, 3 \times 4, \ldots, (2i-1) \times 2i$ until we have deleted all n points, and we have $1 \times 2 + 3 \times 4 + \cdots + (2i-1) \times 2i = n$, which is equivalent to $\frac{i(i+1)(4i-1)}{3} = n$. We solve i and obtain $i = O(\sqrt[3]{n})$. In general, we need $O(\sqrt[3]{n})$ iterations, where each iteration needs $O(n \log n)$ for distance checking. Thus, the simplification time is $O(n\sqrt[3]{n} \log n)$.

Secondly, we prove the number of cells in $\widetilde{H}$ and output size. Our experiments show that there are $O(\mu)$ deleted cells belonging to each added cell in average. Since there are total n cells on H, we obtain that there are $O(\frac{n}{\mu})$ cells on $\widetilde{H}$.

Thirdly, we prove that $\widetilde{H}$ is an ϵ-approximation of H. We need to show that for any pairs of points s and t on H, $(1-\epsilon)|\Pi(s,t|H)| \leq |\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s,t|H)|$. There are three cases. (1) For the *both cells remaining case*, from Lemma D.1, we know that for any pairs of points s_1 and t_1 on cells in C_{rema}, $(1-\epsilon)|\Pi(s_1, t_1|H)| \leq |\Pi(\widetilde{s_1}, \widetilde{t_1}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_1, t_1|H)|$. (2) For the *one cell deleted and one cell remaining case*, from Lemma D.2, we know that for any pairs of points s_2 on cells in C_{rema} and points t_2 on cells in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_2, t_2|H)| \leq |\Pi(\widetilde{s_2}, \widetilde{t_2}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_2, t_2|H)|$. (3) For the *both cells deleted case*, there are two more sub-cases: (i) $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ does not pass on cells in C_{rema}, which contains a special case of *different and non-adjacent belonging cell in both cells deleted case* (i.e., $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ only passes on added cell and other adjacent added cells of it), *different and adjacent belonging cell in both cells deleted case* and *same belonging cell in both cells deleted case.* (ii) $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ passes on cells in C_{rema}, which contains *different and non-adjacent belonging cell in both cells deleted case.* From Lemma D.3, we know that for any pairs of points s_3 and t_3 on cells in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_3, t_3|H)| \leq |\Pi(\widetilde{s_3}, \widetilde{t_3}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s_3, t_3|H)|$ for both these two sub-cases. In general, we have considered all three cases for s and t, and we obtain that $\widetilde{H}$ is an ϵ-approximation of H. ☐

Proof of Theorem 4.3. Firstly, we prove the query time of both the *kNN* and range query algorithm.

- For algorithm *HM-SP* on H, given a query point i on cell c_i, we just need to perform one Dijkstra's algorithm on H.
- For algorithm *HM-SP* on $\widetilde{H}$, given a query point i on cell c_i, if c_i is a remaining cell, we just need to perform one Dijkstra's algorithm on $\widetilde{H}$; if c_i is a deleted cell, we just need to perform $\widetilde{N}(O^{-1}(c_i))$ Dijkstra's algorithm on $\widetilde{H}$. Since $\widetilde{N}(O^{-1}(c_i))$ is a constant, it can be omitted in the big-O notation.

Since performing one Dijkstra's algorithm on H and $\widetilde{H}$ are $O(n \log n)$ and $O(\frac{n}{\mu} \log \frac{n}{\mu})$ (i.e., the shortest path query time for algorithm *HM-SP* on H and $\widetilde{H}$), respectively, the query time of both the *kNN* and range query by using algorithm *HM-SP* is $O(n \log n)$ on H and is $O(\frac{n}{\mu} \log \frac{n}{\mu})$ on $\widetilde{H}$.

Secondly, we prove the error ratio of both the *kNN* and range query algorithm (using the height map as the 3D surface for calculating the optimal distance).

- For algorithm *HM-SP* on H, it returns the exact shortest path passing on H, so it also returns the exact result for the *kNN* and range query.
- For algorithm *HM-SP* on $\widetilde{H}$, we give some notation first. For the *kNN* query and the range query, both of which return a set of points, we can simplify the notation by denoting the set of points returned using the shortest distance on H computed by algorithm *HM-SP* on H as X, where X contains either (1) k nearest points to query point i, or (2) points within a range of distance r to i. Similarly, we denote the set of points returned using the shortest distance on $\widetilde{H}$ computed by algorithm *HM-SP* on $\widetilde{H}$ as X', where X' contains either (1) k nearest points to query point i, or (2) points within a range of distance r to i. In Figure 1 (a), suppose that the exact k nearest points ($k = 2$) of a is c, d, i.e., $X = \{c, d\}$. Suppose that our *kNN* query algorithm finds the k nearest points ($k = 2$) of a is b, c, i.e., $X' = \{b, c\}$. Recall that we let p_f (resp. p'_f) be the point in X (resp. X') that is furthest from i. We further let q_f (resp. q'_f) be the point in X (resp. X') that is furthest from i calculated by algorithm *HM-SP* on $\widetilde{H}$. Recall the error ratio of *kNN* and range queries is $\beta = \frac{|\Pi(i, p'_f|H)|}{|\Pi(i, p_f|H)|} -$

1. According to Theorem 4.2, we have $|\Pi(\widetilde{i}, \widetilde{p'_f}|\widetilde{H})| \geq (1-\epsilon)|\Pi(i, p'_f|H)|$. Thus, we have $\beta \leq \frac{|\Pi(\widetilde{i}, \widetilde{p'_f}|\widetilde{H})|}{(1-\epsilon)|\Pi(i, p_f|H)|} - 1$. By the definition of p_f and q_f, we have $|\Pi(i, p_f|H)| \geq |\Pi(i, q_f|H)|$. Thus, we have $\beta \leq \frac{|\Pi(\widetilde{i}, \widetilde{p'_f}|\widetilde{H})|}{(1-\epsilon)|\Pi(i, q_f|H)|} - 1$. By the definition of p'_f and q'_f, we have $|\Pi(\widetilde{i}, \widetilde{p'_f}|\widetilde{H})| \leq |\Pi(\widetilde{i}, \widetilde{q'_f}|\widetilde{H})|$. Thus, we have $\beta \leq \frac{|\Pi(\widetilde{i}, \widetilde{q'_f}|\widetilde{H})|}{(1-\epsilon)|\Pi(i, q_f|H)|} - 1$. According to Theorem 4.2, we have $|\Pi(\widetilde{i}, \widetilde{q_f}|\widetilde{H})| \leq (1+\epsilon)|\Pi(i, q_f|H)|$. Then, we have $\beta \leq \frac{(1+\epsilon)|\Pi(\widetilde{i}, \widetilde{q'_f}|\widetilde{H})|}{(1-\epsilon)|\Pi(\widetilde{i}, \widetilde{q_f}|\widetilde{H})|} - 1$. By our *kNN* and range query algorithm, we have $|\Pi(\widetilde{i}, \widetilde{q'_f}|\widetilde{H})| \leq |\Pi(\widetilde{i}, \widetilde{q_f}|\widetilde{H})|$. Thus, we have $\beta \leq \frac{1+\epsilon}{1-\epsilon} - 1 = \frac{2\epsilon}{1-\epsilon}$. So, algorithm *HM-SP* on $\widetilde{H}$ has an error ratio $\frac{2\epsilon}{1-\epsilon}$ for the *kNN* and range query.

There is no error ratio guarantee of both the *kNN* and range query algorithm when using the *TIN* as the 3D surface for calculating the ground-truth distance. This is because given a pair of points s and t on H, $\widetilde{H}$ and T, there is no relationship between $|\Pi(s,t|H)|$ and $|\Pi(s,t|T)|$ for algorithm *HM-SP* on H, and there is no relationship between $|\Pi(\widetilde{s},\widetilde{t}|\widetilde{H})|$ and $|\Pi(s,t|T)|$ for algorithm *HM-SP* on $\widetilde{H}$. □

Theorem D.4. *Compared with the exact shortest path passing on a TIN (that is converted from a height map), i.e., ground-truth distance, algorithm HM-SP's versions on a height map and a simplified height map are both the approximate shortest path passing on a TIN.*

Proof. Since we compare with the exact shortest surface path passing on a *TIN*, algorithm *HM-SP* on both the height map and the simplified height map returns the approximate shortest path passing on a *TIN*. □

Proof of Theorem 4.4. Firstly, we prove the simplification time. After using algorithm *HM-Simplify* for simplification, it needs additional time for data structure construction. According to study [50], the data structure construction time is $O(\mu' \log \mu')$, where μ' is the number of vertices of the input graph. Since the input graph in our case is the simplified height map graph with $O(\frac{n}{\mu})$ vertices (i.e., $\widetilde{H}$ has $O(\frac{n}{\mu})$ cells), the data structure construction time is $O(\frac{n}{\mu}\log\frac{n}{\mu})$. Since the original data structure in study [50] uses Euclidean distance as the distance metric, where each computation can be finished in $O(1)$ time, but we use the shortest distance on the simplified height map graph as the the distance metric, where each computation can be finished in $O(\frac{n}{\mu}\log\frac{n}{\mu})$ time, we need to multiple it with the original data structure construction time $O(\frac{n}{\mu}\log\frac{n}{\mu})$. So the total simplification time is $O(n\sqrt[3]{n}\log n + \frac{n^2}{\mu^2}\log^2\frac{n}{\mu})$.

Secondly, we prove the number of edges in $G_{\widetilde{H}}$ and output size. According to study [50], the number of edges in $G_{\widetilde{H}}$ and output size are $O(\mu' \log \mu')$. Since the input graph in our case is the simplified height map graph with $O(\frac{n}{\mu})$ vertices (i.e., $\widetilde{H}$ has $O(\frac{n}{\mu})$ cells), the number of edges in $G_{\widetilde{H}}$ and output size are $O(\frac{n}{\mu}\log\frac{n}{\mu})$.

Thirdly, we prove that for any pairs of points s and t on H, algorithm *HM-Simplify-DS* has $|\Pi(\widetilde{s},\widetilde{t}|G_{\widetilde{H}})| \leq (1+\epsilon')(1+\epsilon)|\Pi(s,t|H)|$. Since $G_{\widetilde{H}}$ is a $(1+\epsilon')$-approximate data structure on $\widetilde{H}$ according to study [50], and $\widetilde{H}$ is an ϵ-approximation of H, we obtain that for any pairs of points s and t on H, algorithm *HM-Simplify-DS* has $|\Pi(\widetilde{s},\widetilde{t}|G_{\widetilde{H}})| \leq (1+\epsilon')(1+\epsilon)|\Pi(s,t|H)|$. □

Proof of Theorem 4.5. Firstly, we prove the shortest path query time. According to study [50], the data structure $G_{\widetilde{H}}$ can return the shortest path result in $O(1)$ time. Thus, the shortest path query time is $O(1)$.

Secondly, we prove the *kNN* and range query time. Since we need to use algorithm *HM-SP-DS* $O(n')$ times for both the *kNN* and range query, the *kNN* and range query time is $O(n')$.

Thirdly, we prove the memory usage. Since the output size of algorithm *HM-Simplify-DS* is $O(\frac{n}{\mu}\log\frac{n}{\mu})$, the memory usage of algorithm *HM-Simplify-DS* is also $O(\frac{n}{\mu}\log\frac{n}{\mu})$.

Fourthly, we prove the error guarantee. Since $G_{\widetilde{H}}$ is a $(1+\epsilon')(1+\epsilon)$-approximate graph of (the height map graph of) H, we finish the proof.

Fifthly, we prove the error ratio of both the *kNN* and range query algorithm (using the height map as the 3D surface for calculating the optimal distance). We give some notation first. For the *kNN* query and the range query, both of which return a set of points, we can simplify the notation by denoting the set of points returned using the shortest distance on H computed by algorithm *HM-SP* on H as X, where X contains either (1) k nearest points to query point i, or (2) points within a range of distance r to i. Similarly, we denote the set of points returned using the shortest distance on $G_{\widetilde{H}}$ computed by algorithm *HM-SP-DS* on $G_{\widetilde{H}}$ as X', where X' contains either (1) k nearest points to query point i, or (2) points within a range of distance r to i. We let p_f (resp. p'_f) be the point in X (resp. X') that is furthest from i. We further let q_f (resp. q'_f) be the point in X (resp. X') that is furthest from i calculated by algorithm *HM-SP-DS* on $G_{G_{\widetilde{H}}}$. Recall the error ratio of *kNN* and range queries is $\beta = \frac{|\Pi(i,p'_f|H)|}{|\Pi(i,p_f|H)|} - 1$. Note that we have $|\Pi(\widetilde{i}, \widetilde{p'_f}|G_{\widetilde{H}})| \geq |\Pi(i, p'_f|H)|$. Thus, we have $\beta \leq \frac{|\Pi(\widetilde{i}, \widetilde{p'_f}|G_{\widetilde{H}})|}{|\Pi(i, p_f|H)|} - 1$. By the definition of p_f and q_f, we have $|\Pi(i,p_f|H)| \geq |\Pi(i,q_f|H)|$. Thus, we have $\beta \leq \frac{|\Pi(\widetilde{i}, \widetilde{p'_f}|G_{\widetilde{H}})|}{|\Pi(i, q_f|H)|} - 1$. By the definition of p'_f and q'_f, we have $|\Pi(\widetilde{i}, \widetilde{p'_f}|G_{\widetilde{H}})| \leq |\Pi(\widetilde{i}, \widetilde{q'_f}|G_{\widetilde{H}})|$. Thus, we have $\beta \leq \frac{|\Pi(\widetilde{i}, \widetilde{q'_f}|G_{\widetilde{H}})|}{|\Pi(i, q_f|H)|} - 1$. Note that we have $|\Pi(\widetilde{i}, \widetilde{q_f}|G_{\widetilde{H}})| \leq (1+\epsilon')(1+\epsilon)|\Pi(i,q_f|H)|$. Then, we have $\beta \leq \frac{(1+\epsilon')(1+\epsilon)|\Pi(\widetilde{i}, \widetilde{q'_f}|G_{\widetilde{H}})|}{|\Pi(\widetilde{i}, \widetilde{q_f}|G_{\widetilde{H}})|} - 1$. By our *kNN* and range query algorithm, we have $|\Pi(\widetilde{i}, \widetilde{q'_f}|G_{\widetilde{H}})| \leq |\Pi(\widetilde{i}, \widetilde{q_f}|G_{\widetilde{H}})|$. Thus, we have $\beta \leq (1+\epsilon')(1+\epsilon) - 1 = \epsilon' \cdot \epsilon + \epsilon' + \epsilon$. So, algorithm *HM-SP-DS* on $G_{\widetilde{H}}$ has an error ratio $\epsilon' \cdot \epsilon + \epsilon' + \epsilon$ for the *kNN* and range query.

There is no error ratio guarantee of both the *kNN* and range query algorithm when using the *TIN* as the 3D surface for calculating the ground-truth distance. This is because given a pair of points s and t on $G_{\widetilde{H}}$ and T, there is no relationship between $|\Pi(\widetilde{s},\widetilde{t}|G_{\widetilde{H}})|$ and $|\Pi(s,t|T)|$ for algorithm *HM-SP-DS* on $G_{\widetilde{H}}$. □

Theorem D.5. *The simplification time, number of vertices in the simplified TIN $\widetilde{T}$ of T and output size of algorithm TIN-SSimplify-Adapt(HM) are $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$, $O(n)$ and $O(n)$, respectively. Given a height map H, it first convert H to a TIN T, and then returns $\widetilde{T}$ such that $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$ for any pairs of vertices s and t on T, where $\Pi(s,t|\widetilde{T})$ is the shortest surface path between s and t passing on $\widetilde{T}$.*

Proof. Firstly, we prove the simplification time. It first needs to convert the height map to a *TIN* in $O(n)$ time. Then, in each vertex removal iteration, it places $O(\frac{1}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$ Steiner points [33, 40] on each face adjacent to the deleted vertex, and use algorithm *TIN-ESSP* [27, 63, 70] in $O(n^2)$ time to check the distances between these Steiner points on the original *TIN* and the simplified *TIN*, so this step needs $O(\frac{n^2}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$ time. Since there are total $O(n)$ vertex removal iterations, the simplification time for simplifying a *TIN* is $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$. In general, the total simplification time is $O(n+\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon}) = (\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$.

Secondly, we prove the number of vertices in $\widetilde{T}$ and output size. Although this algorithm could simplify a *TIN*, our experimental results show the simplified *TIN* still has $O(n)$ vertices. Thus, the number of vertices in $\widetilde{T}$ and output size are both $O(n)$.

Thirdly, we prove that for any pairs of vertices s and t on T, algorithm *TIN-SSimplify-Adapt(HM)* has $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$. In each vertex removal iteration, it performs a check between any pairs of Steiner points u and v (on the faces that are adjacent to the deleted vertex) on T whether $(1-\epsilon)|\Pi(u,v|T)| \leq |\Pi(u,v|\widetilde{T})| \leq (1+\epsilon)|\Pi(u,v|T)|$. According to study [40], given any pair of points c and q (on the faces that are adjacent to the deleted vertex) on T, if $(1-\epsilon)|\Pi(u,v|T)| \leq |\Pi(u,v|\widetilde{T})| \leq (1+\epsilon)|\Pi(u,v|T)|$, then $(1-\epsilon)|\Pi(p,q|T)| \leq |\Pi(p,q|\widetilde{T})| \leq (1+\epsilon)|\Pi(p,q|T)|$. Following the similar proof in Theorem 4.1, we know that for any pairs of points s' and t' on any faces of T, we have $(1-\epsilon)|\Pi(s',t'|T)| \leq |\Pi(s',t'|\widetilde{T})| \leq (1+\epsilon)|\Pi(s',t'|T)|$. This is because if the distances between any pair of points on the faces near the deleted vertex do not change a lot, then the distances between any pair of points on the faces far away from the deleted vertex cannot change a lot. Since s and t can be any vertices of T, and s' and t' can be any points on any faces of T, we obtain that for any pairs of vertices s and t on T, algorithm *TIN-SSimplify-Adapt(HM)* has $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$. □

Theorem D.6. *The simplification time, number of vertices in the simplified TIN $\widetilde{T}$ of T and output size of algorithm TIN-NSimplify-Adapt(HM) are $O(n^2\log n)$, $O(n)$ and $O(n)$, respectively. Given a height map H, it first converts H to a TIN T, and then returns $\widetilde{T}$ such that $(1-\epsilon)|\Pi_N(s,t|T)| \leq |\Pi_N(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi_N(s,t|T)|$ for any pairs of vertices s and t on T, where $\Pi_N(s,t|\widetilde{T})$ is the shortest network path between s and t passing on $\widetilde{T}$.*

Proof. Firstly, we prove the simplification time. It first needs to convert the height map to a *TIN* in $O(n)$ time. Then, in each vertex removal iteration, it uses algorithm *TIN-SNP* [43] in $O(n\log n)$ time to check the distances between any pair of vertices that are neighbors of the deleted vertex on the original *TIN* and the simplified *TIN*. Since there are only $O(1)$ vertices that are neighbors of the deleted vertex, this step needs $O(n\log n)$ time. Since there are total $O(n)$ vertex removal iterations, the simplification time for simplifying a *TIN* is $O(n^2\log n)$. In general, the total simplification time is $O(n+n^2\log n) = O(n^2\log n)$.

Secondly, we prove the number of vertices in $\widetilde{T}$ and output size. Although this algorithm could simplify a *TIN*, our experimental results show the simplified *TIN* still has $O(n)$ vertices. Thus, the number of vertices in $\widetilde{T}$ and output size are both $O(n)$.

Thirdly, we prove that for any pairs of vertices s and t on T, algorithm *TIN-NSimplify-Adapt(HM)* has $(1-\epsilon)|\Pi_N(s,t|T)| \leq |\Pi_N(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi_N(s,t|T)|$. In each vertex removal iteration, it performs a check between any pairs of vertices u and v (adjacent to the deleted vertex) on T whether $(1-\epsilon)|\Pi_N(u,v|T)| \leq |\Pi_N(u,v|\widetilde{T})| \leq (1+\epsilon)|\Pi_N(u,v|T)|$. If the distances between any pair of vertices adjacent to the deleted vertex do not change a lot, then the distances between any pair of vertices far away from the deleted vertex cannot change a lot. So we obtain that for any pairs of vertices s and t on T, algorithm *TIN-NSimplify-Adapt(HM)* has $(1-\epsilon)|\Pi_N(s,t|T)| \leq |\Pi_N(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi_N(s,t|T)|$. The detailed proof can be found in study [43]. □

Theorem D.7. *The simplification time, number of points in the simplified point cloud $\widetilde{P}$ of P and output size of algorithm PC-Simplify-Adapt(HM) are $O(n^2\log n)$, $O(n)$ and $O(n)$, respectively. Given a height map H, it first converts H to a point cloud P, and then returns $\widetilde{P}$ such that $(1-\epsilon)|\Pi(s,t|P)| \leq |\Pi(s,t|\widetilde{P})| \leq (1+\epsilon)|\Pi(s,t|P)|$ for any pairs of points s and t on P, where $\Pi(s,t|\widetilde{P})$ is the shortest path between s and t passing on $\widetilde{P}$.*

Proof. Firstly, we prove the simplification time. It first needs to convert the height map to a point cloud in $O(n)$ time. Then, in each point removal iteration, we adapt it by constructing a point cloud graph and useing algorithm *PC-SP* [69] in $O(n\log n)$ time to check the distances between any pair of point that are neighbors of the deleted point on the original point cloud and the simplified point cloud. Since there are only $O(1)$ points that are neighbors of the deleted point, this step needs $O(n\log n)$ time. Since there are total $O(n)$ point removal iterations, the simplification time for simplifying a point is $O(n^2\log n)$. In general, the total simplification time is $O(n+n^2\log n) = O(n^2\log n)$.

Secondly, we prove the number of points in $\widetilde{P}$ and output size. Although this algorithm could simplify a point cloud, our experimental results show the simplified point cloud still has $O(n)$ points. Thus, the number of points in $\widetilde{P}$ and output size are both $O(n)$.

Thirdly, we prove that for any pairs of points s and t on P, algorithm *PC-Simplify-Adapt(HM)* has $(1-\epsilon)|\Pi(s,t|P)| \leq |\Pi(s,t|\widetilde{P})| \leq (1+\epsilon)|\Pi(s,t|P)|$. In each point removal iteration, it performs a check between any pairs of points u and v (adjacent to the deleted point) on P whether $(1-\epsilon)|\Pi(u,v|P)| \leq |\Pi(u,v|\widetilde{P})| \leq (1+\epsilon)|\Pi(u,v|P)|$. If the distances between any pair of points adjacent to the deleted point do not change a lot, then the distances between any pair of points far away from the deleted point cannot change a lot. So we obtain that for any pairs of points s and t on P, algorithm *PC-Simplify-Adapt(HM)* has $(1-\epsilon)|\Pi(s,t|P)| \leq |\Pi(s,t|\widetilde{P})| \leq (1+\epsilon)|\Pi(s,t|P)|$. □

THEOREM D.8. *The simplification time, number of cells in the simplified $\widetilde{H}$ of H and output size of algorithm HM-Simplify-LQT1 are $O(n\sqrt[3]{n}\log n)$, $O(\frac{n}{\mu})$ and $O(\frac{n}{\mu})$, respectively. Given a height map H, it returns an ϵ-approximate simplified height map $\widetilde{H}$ of H.*

PROOF. The simplification time, number of cells in $\widetilde{H}$, output size and error guarantee of algorithm *HM-Simplify-LQT1* are the same as algorithm *HM-Simplify*. □

THEOREM D.9. *The simplification time, number of cells in the simplified $\widetilde{H}$ of H and output size of algorithm HM-Simplify-LQT2 are $O(n\sqrt[3]{n}\log n)$, $O(\frac{n}{\mu})$ and $O(\frac{n}{\mu})$, respectively. Given a height map H, it returns an ϵ-approximate simplified height map $\widetilde{H}$ of H.*

PROOF. The simplification time, number of cells in $\widetilde{H}$, output size and error guarantee of algorithm *HM-Simplify-LQT2* are the same as algorithm *HM-Simplify*. □

THEOREM D.10. *The simplification time, number of cells in the simplified $\widetilde{H}$ of H and output size of algorithm HM-Simplify-LS are $O(n\sqrt[3]{n}\log n)$, $O(n)$ and $O(n)$, respectively. Given a height map H, it returns an ϵ-approximate simplified height map $\widetilde{H}$ of H.*

PROOF. Firstly, we prove the number of cells in $\widetilde{H}$ and output size. Since it uses the naive merging technique that only merges two cells in Section 4.2, although it could simplify a height map, our experimental results show the simplified height map still has $O(n)$ cells. Thus, the number of cells in $\widetilde{H}$ and output size are both $O(n)$.

The simplification time and error guarantee of algorithm *HM-Simplify-LS* are the same as algorithm *HM-Simplify*. □

THEOREM D.11. *The simplification time, number of cells in the simplified $\widetilde{H}$ of H and output size of algorithm HM-Simplify-LST are $O(n^3\sqrt[3]{n}\log n)$, $O(\frac{n}{\mu})$ and $O(\frac{n}{\mu})$, respectively. Given a height map H, it returns an ϵ-approximate simplified height map $\widetilde{H}$ of H.*

PROOF. We prove the simplification time. Since it uses the naive checking technique that checks whether Inequality 1 is satisfied for all cells in Section 4.2, in each cell merging iteration, it needs to check the distance between any pairs of cells on $\widetilde{H}$ and H, i.e., run Dijkstra's algorithm in $O(n\log n)$ time for $O(n)$ cells, which needs $O(n^2\log n)$ time. According to Theorem 4.1, there are total $O(\sqrt[3]{N})$ cell merging iterations. So the total simplification time is $O(n^2\sqrt[3]{N}\log n)$.

The number of cells in the simplified $\widetilde{H}$ of H, output size and error guarantee of algorithm *HM-Simplify-LST* are the same as algorithm *HM-Simplify*. □

THEOREM D.12. *The shortest path query time, kNN and range query time and memory usage of algorithm TIN-ESSP-Adapt(HM) are $O(n^2)$, $O(n^2)$ and $O(n^2)$ on a TIN T, and are $O(n^2)$, $O(n^2)$ and $O(n^2)$ on a simplified TIN $\widetilde{T}$, respectively. Compared with $\Pi(s,t|T)$, i.e., the ground-truth distance, it returns the exact shortest surface path passing on a TIN (that is converted from the height map), and always has $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi_{TIN\text{-}ESSP\text{-}Adapt(HM)}(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$ for any pairs of vertices s and t on T, where $\Pi_{TIN\text{-}ESSP\text{-}Adapt(HM)}(s,t|\widetilde{T})$ is the approximate shortest surface path of algorithm TIN-ESSP-Adapt(HM) passing on a simplified TIN $\widetilde{T}$ (that is calculated by algorithm TIN-SSimplify-Adapt(HM)) between s and t. Compared with $\Pi(s,t|H)$, i.e., the optimal distance, its versions on a TIN and a simplified TIN are both the approximate shortest paths passing on a height map. When using the TIN as the 3D surface for calculating the ground-truth distance, it returns the exact kNN and range query result on T and has an error ratio $\frac{2\epsilon}{1-\epsilon}$ on $\widetilde{T}$ for both kNN and range queries, respectively.*

PROOF. Firstly, we prove the shortest path query time on both T and $\widetilde{T}$. The proof of the shortest path query time of algorithm *TIN-ESSP* on a *TIN* with $O(n)$ vertices is in [27, 63, 70]. Since there are both $O(n)$ vertices on T and $\widetilde{T}$, the shortest path query time is $O(n^2)$. But, since algorithm *TIN-ESSP-Adapt(HM)* first needs to convert the height map to a *TIN*, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time is $O(n+n^2)=O(n^2)$.

Secondly, we prove the *kNN* and range query time on both T and $\widetilde{T}$. Since it is a single-source-all-destination algorithm, we use it once for both the *kNN* and range query. So, the *kNN* and range query is $O(n^2)$.

Thirdly, we prove the memory usage on both T and $\widetilde{T}$. The proof of the memory usage of algorithm *TIN-ESSP* is in [27, 63, 70], which is similar to algorithm *TIN-ESSP-Adapt(HM)*. Thus, the memory usage is $O(n^2)$.

Fourthly, we prove the error guarantee. Compared with $\Pi(s,t|T)$, the proof that it returns the exact shortest path passing on a *TIN* is in [27, 63, 70]. Since the *TIN* is converted from the height map, so algorithm *TIN-ESSP-Adapt(HM)* returns the exact shortest surface path passing on a *TIN* (that is converted from the height map). Since the simplified *TIN* is calculated by algorithm *TIN-SSimplify-Adapt(HM)*, so it has $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi_{TIN\text{-}ESSP\text{-}Adapt(HM)}(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$ for any pairs of vertices s and t on T. Compared with $\Pi(s,t|H)$, since we regard $\Pi(s,t|H)$ as the exact shortest path passing on the height map, algorithm *TIN-ESSP-Adapt(HM)* on both the *TIN* and the simplified *TIN* returns the approximate shortest path passing on a height map.

Fifthly, we prove the error ratio of both the *kNN* and range query algorithm (using the *TIN* as the 3D surface for calculating the ground-truth distance). The proof is similar to algorithm *HM-SP*.

There is no error ratio guarantee of both the *kNN* and range query algorithm when using the height map as the 3D surface for calculating the optimal distance. This is because given a pair of points s and t on T, $\widetilde{T}$ and H, there is no relationship between $|\Pi(s,t|T)|$ and $|\Pi(s,t|H)|$ for algorithm *TIN-ESSP-Adapt(HM)* on T, and there is no relationship between $|\Pi_{TIN\text{-}ESSP\text{-}Adapt(HM)}(\widetilde{s},\widetilde{t}|\widetilde{T})|$ and $|\Pi(s,t|H)|$ for algorithm *TIN-ESSP-Adapt(HM)* on $\widetilde{T}$. □

THEOREM D.13. *The shortest path query time, kNN and range query time and memory usage of algorithm TIN-ASSP-Adapt(HM) are $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$, $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$ and $O(n)$, respectively. Compared with $\Pi(s,t|T)$, i.e., the ground-truth distance, it always has $|\Pi_{TIN\text{-}ASSP\text{-}Adapt(HM)}(s,t|T)| \leq (1+\epsilon)|\Pi(s,t|T)|$ for any pairs of vertices s and t on T, where $\Pi_{TIN\text{-}ASSP\text{-}Adapt(HM)}(s,t|T)$ is the shortest surface path of algorithm TIN-ASSP-Adapt(HM) passing on a TIN T*

(that is converted from the height map) between s and t. Compared with $\Pi(s,t|H)$, i.e., the optimal distance, it returns the approximate shortest path passing on a height map. When using the TIN as the 3D surface for calculating the ground-truth distance, it has an error ratio ϵ on $\widetilde{T}$ for both kNN and range queries.

Proof. Firstly, we prove the shortest path query time. The proof of the shortest path query time of algorithm *TIN-ASSP* is in [42]. Note that in Section 4.2 of [42], the shortest path query time of algorithm *TIN-ASSP-Adapt(HM)* is $O((n+n')(\log(n+n')+(\frac{l_{max}K}{l_{min}\sqrt{1-\cos\theta}})^2))$, where $n' = O(\frac{l_{max}K}{l_{min}\sqrt{1-\cos\theta}}n)$ and K is a parameter which is a positive number at least 1. By Theorem 1 of [42], we obtain that its error guarantee ϵ is equal to $\frac{1}{K-1}$. Thus, we can derive that the shortest path query time of algorithm *TIN-ASSP-Adapt(HM)* is $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}})+\frac{l_{max}^2}{(\epsilon l_{min}\sqrt{1-\cos\theta})^2})$. Since for n, the first term is larger than the second term, so we obtain the shortest path query time of algorithm *TIN-ASSP-Adapt(HM)* is $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$. But since algorithm *TIN-ASSP-Adapt(HM)* first needs to convert the height map to a *TIN*, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time of algorithm *TIN-ASSP-Adapt(HM)* is $O(n+\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}})) = O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$. In [68], it omits the constant term in the shortest path query time. After adding back these terms, the shortest path query time is the same.

Secondly, we prove the *kNN* and range query time. Since it is a single-source-all-destination algorithm, we use it once for both the *kNN* and range query. So, the *kNN* and range query is $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$.

Thirdly, we prove the memory usage. Since it is a Dijkstra's algorithm and there are total n vertices on the *TIN*, the memory usage is $O(n)$.

Fourthly, we prove the error guarantee. Compared with $\Pi(s,t|T)$, the proof of the error guarantee of algorithm *TIN-ASSP-Adapt(HM)* is in [42, 68]. Since the *TIN* is converted from the point cloud, so algorithm *TIN-ASSP-Adapt(HM)* always has $|\Pi_{\textit{TIN-ASSP-Adapt(HM)}}(s,t|T)| \leq (1+\epsilon)|\Pi(s,t|T)|$ for any pairs of vertices s and t on T. Compared with $\Pi(s,t|H)$, since we regard $\Pi(s,t|H)$ as the exact shortest path passing on the height map, algorithm *TIN-ASSP-Adapt(HM)* returns the approximate shortest path passing on a height map.

Fifthly, we prove the error ratio of both the *kNN* and range query algorithm (using the *TIN* as the 3D surface for calculating the ground-truth distance). The proof is similar to algorithm *HM-SP-DS*, by just changing $(1+\epsilon')(1+\epsilon)$ to $(1+\epsilon)$. Thus, we have $\beta \leq (1+\epsilon)-1=\epsilon$. So, algorithm *TIN-ASSP-Adapt(HM)* on T has an error ratio ϵ for the *kNN* and range query.

There is no error ratio guarantee of both the *kNN* and range query algorithm when using the height map as the 3D surface for calculating the optimal distance. This is because given a pair of points s and t on T and H, there is no relationship between $|\Pi_{\textit{TIN-ASSP-Adapt(HM)}}(s,t|T)|$ and $|\Pi(s,t|H)|$ for algorithm *TIN-ASSP-Adapt(HM)* on T. □

Theorem D.14. *The shortest path query time, kNN and range query time and memory usage of algorithm TIN-SNP-Adapt(HM) are $O(n\log n)$, $O(n\log n)$ and $O(n)$ on a TIN T, and are $O(n\log n)$, $O(n\log n)$ and $O(n)$ on a simplified TIN $\widetilde{T}$, respectively. Compared with $\Pi(s,t|T)$, i.e., the ground-truth distance, it always has $|\Pi_{\textit{TIN-SNP-Adapt(HM)}}(s,t|T)| \leq \alpha\cdot|\Pi(s,t|T)|$ for any pairs of vertices s and t on T, where $\Pi_{\textit{TIN-SNP-Adapt(HM)}}(s,t|T)$ is the shortest network path of algorithm TIN-SNP-Adapt(HM) passing on a TIN T (that is converted from the height map) between s and t, $\alpha=\max\{\frac{2}{\sin\theta},\frac{1}{\sin\theta\cos\theta}\}$, and returns the approximate shortest path passing on a simplified TIN $\widetilde{T}$ (that is calculated by algorithm TIN-NSimplify-Adapt(HM)). Compared with $\Pi(s,t|H)$, i.e., the optimal distance, its versions on a TIN and a simplified TIN are both the approximate shortest paths passing on a height map. When using the TIN as the 3D surface for calculating the ground-truth distance, there is no error ratio guarantee on T and $\widetilde{T}$ for both kNN and range queries.*

Proof. Firstly, we prove the shortest path query time on both T and $\widetilde{T}$. Since algorithm *TIN-SNP* only computes the shortest network path passing on T (that is converted from the height map with total n vertices) and $\widetilde{T}$ (that is calculated by algorithm *TIN-NSimplify-Adapt(HM)* total n vertices), it is a Dijkstra's algorithm, the shortest path query time is $O(n\log n)$. But since algorithm *TIN-SNP-Adapt(HM)* first needs to convert the height map to a *TIN*, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time is $O(n+n\log n)=O(n\log n)$.

Secondly, we prove the *kNN* and range query time on both T and $\widetilde{T}$. Since it is a single-source-all-destination algorithm, we use it once for both the *kNN* and range query. So, the *kNN* and range query is $O(n\log n)$.

Thirdly, we prove the memory usage. Since it is a Dijkstra's algorithm and there are total n vertices on the *TIN*, the memory usage is $O(n)$.

Fourthly, we prove the error guarantee. Recall that $\Pi_N(s,t|T)$ is the shortest network path passing on T (that is converted from the height map) between s and t, so actually $\Pi_N(s,t|T)$ is the same as $\Pi_{\textit{TIN-SNP-Adapt(HM)}}(s,t|T)$. Recall that $\Pi_E(s,t|T)$ is the shortest path passing on the edges of T (where these edges belong to the faces that $\Pi(s,t|T)$ passes) between s and t. Compared with $\Pi(s,t|T)$, we know $|\Pi_E(s,t|T)| \leq \alpha\cdot|\Pi(s,t|T)|$ (according to left hand side equation in Lemma 2 of [43]) and $|\Pi_N(s,t|T)| \leq |\Pi_E(s,t|T)|$ (since $\Pi_N(s,t|T)$ considers all the edges on T), so we have $|\Pi_{\textit{TIN-SNP-Adapt(HM)}}(s,t|T)| \leq \alpha\cdot|\Pi(s,t|T)|$ for any pairs of vertices s and t on T. Since the simplified *TIN* is calculated by algorithm *TIN-NSimplify-Adapt(HM)*, algorithm *TIN-SNP-Adapt(HM)* returns the approximate shortest path passing on a simplified *TIN*. Compared with $\Pi(s,t|H)$, since we regard $\Pi(s,t|H)$ as the exact shortest path passing on the height map, algorithm *TIN-SNP-Adapt(HM)* on both the *TIN* and the simplified *TIN* returns the approximate shortest path passing on a height map.

There is no error ratio guarantee of both the *kNN* and range query algorithm when using the *TIN* as the 3D surface for calculating the ground-truth distance). This is because given a pair of points s and t on T and $\widetilde{T}$, there is no relationship between $|\Pi_N(s,t|T)|$ and $|\Pi(s,t|T)|$ for algorithm *TIN-SNP-Adapt(HM)* on T, and there is no relationship between $|\Pi_{\textit{TIN-SNP-Adapt(HM)}}(\widetilde{s},\widetilde{t}|\widetilde{T})|$ and $|\Pi(s,t|T)|$ for algorithm *TIN-ESSP-Adapt(HM)* on $\widetilde{T}$.

There is no error ratio guarantee of both the *kNN* and range query algorithm when using the height map as the 3D surface for calculating the optimal distance. This is because given a pair of points s and t on T, $\widetilde{T}$ and H, there is no relationship between $|\Pi_N(s,t|T)|$ and $|\Pi(s,t|H)|$ for algorithm *TIN-SNP-Adapt(HM)* on T, and there is no relationship between $|\Pi_{\textit{TIN-SNP-Adapt(HM)}}(\widetilde{s},\widetilde{t}|\widetilde{T})|$ and $|\Pi(s,t|H)|$ for algorithm *TIN-SNP-Adapt(HM)* on $\widetilde{T}$. □

Theorem D.15. *The shortest path query time, kNN and range query time and memory usage of algorithm PC-SP-Adapt(HM) are $O(n\log n)$, $O(n\log n)$ and $O(n)$ on a point cloud P, and are $O(n\log n)$, $O(n\log n)$ and $O(n)$ on a simplified point cloud $\widetilde{P}$, respectively. Compared with $\Pi(s,t|P)$, it returns the exact shortest path passing on a point cloud (that is converted from the height map), and always has $(1-\epsilon)|\Pi(s,t|P)| \leq |\Pi_{\textit{PC-SP-Adapt(HM)}}(s,t|\widetilde{P})| \leq (1+\epsilon)|\Pi(s,t|P)|$ for any pairs of points s and t on P, where $\Pi_{\textit{PC-SP-Adapt(HM)}}(s,t|\widetilde{P})$ is the approximate shortest path of algorithm PC-SP-Adapt(HM) passing on a simplified point cloud $\widetilde{P}$ (that is calculated by algorithm PC-Simplify-Adapt(HM)) between s and t. Compared with $\Pi(s,t|H)$, i.e., the optimal distance, its version on a point cloud is the exact shortest path passing on a height map, and its version on a simplified point cloud always has $(1-\epsilon)|\Pi(s,t|H)| \leq |\Pi_{\textit{PC-SP-Adapt(HM)}}(s,t|\widetilde{P})| \leq (1+\epsilon)|\Pi(s,t|H)|$ for any pairs of points s and t on H. Compared with $\Pi(s,t|T)$, i.e., the ground-truth distance, its versions on a point cloud and a simplified point cloud are both the approximate shortest path passing on a TIN (that is converted from the height map). When using the height map as the 3D surface for calculating the optimal distance, it returns the exact kNN and range query result on H and has an error ratio $\frac{2\epsilon}{1-\epsilon}$ on $\widetilde{H}$ for both kNN and range queries, respectively.*

Proof. Firstly, we prove the shortest path query time on both P and $\widetilde{P}$. Since algorithm *PC-SP* only computes the shortest path passing on P (that is converted from the height map with total n points) and $\widetilde{P}$ (that is calculated by algorithm *PC-Simplify-Adapt(HM)* total n points), it is a Dijkstra's algorithm, the shortest path query time is $O(n\log n)$. But since algorithm *PC-SP-Adapt(HM)* first needs to convert the height map to a point cloud, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time is $O(n+n\log n)=O(n\log n)$.

Secondly, we prove the *kNN* and range query time on both P and $\widetilde{P}$. Since it is a single-source-all-destination algorithm, we use it once for both the *kNN* and range query. So, the *kNN* and range query is $O(n\log n)$.

Thirdly, we prove the memory usage. Since it is a Dijkstra's algorithm and there are total n points on the point cloud, the memory usage is $O(n)$.

Fourthly, we prove the error guarantee. Compared with $\Pi(s,t|P)$, the proof that it returns the exact shortest path passing on a point cloud is in [69]. Since the point cloud is converted from the height map, so algorithm *PC-SP-Adapt(HM)* returns the exact shortest path passing on a point cloud (that is converted from the height map). Since the simplified point cloud is calculated by algorithm *PC-Simplify-Adapt(HM)*, so it has $(1-\epsilon)|\Pi(s,t|P)| \leq |\Pi_{\textit{PC-SP-Adapt(HM)}}(s,t|\widetilde{P})| \leq (1+\epsilon)|\Pi(s,t|P)|$ for any pairs of points s and t on P. Compared with $\Pi(s,t|H)$, since we regard $\Pi(s,t|H)$ as the exact shortest path passing on the height map, and the height map graph and the point cloud are the same, algorithm *PC-SP-Adapt(HM)* on the point cloud returns the exact shortest path passing on a height map, algorithm *PC-SP-Adapt(HM)* on the simplified point cloud has $(1-\epsilon)|\Pi(s,t|H)| \leq |\Pi_{\textit{PC-SP-Adapt(HM)}}(s,t|\widetilde{P})| \leq (1+\epsilon)|\Pi(s,t|H)|$ for any pairs of points s and t on H. Compared with $\Pi(s,t|T)$, since we regard $\Pi(s,t|T)$ as the exact shortest surface path passing on the *TIN*, algorithm *PC-SP-Adapt(HM)* on both the point cloud and the simplified point cloud returns the approximate shortest path passing on a *TIN*.

Fifthly, we prove the error ratio of both the *kNN* and range query algorithm (using the height map as the 3D surface for calculating the optimal distance). Since the height map graph and the point cloud are the same, the proof is similar to algorithm *HM-SP*.

There is no error ratio guarantee of both the *kNN* and range query algorithm when using the *TIN* as the 3D surface for calculating the ground-truth distance. This is because given a pair of points s and t on P, $\widetilde{P}$ and T, there is no relationship between $|\Pi(s,t|P)|$ and $|\Pi(s,t|T)|$ for algorithm *PC-SP-Adapt(HM)* on P, and there is no relationship between $|\Pi_{\textit{PC-SP-Adapt(HM)}}(\widetilde{s},\widetilde{t}|\widetilde{P})|$ and $|\Pi(s,t|T)|$ for algorithm *PC-SP-Adapt(HM)* on $\widetilde{P}$. □

Theorem D.16. *The shortest path query time, kNN and range query time and memory usage of algorithm HM-SP-LQT1 on the simplified height map are $O(\frac{n^2}{\log n}\log\frac{n}{\mu})$, $O(\frac{n^2}{\log n}\log\frac{n}{\mu})$ and $O(\frac{n}{\mu})$, respectively. Compared with $\Pi(s,t|H)$, i.e., the optimal distance, it returns an approximate shortest path passing on ϵ-approximate simplified height map $\widetilde{H}$ of H. Compared with $\Pi(s,t|T)$, i.e., the ground-truth distance, it returns an approximate shortest path passing on a TIN (that is converted from the height map). When using the height map as the 3D surface for calculating the optimal distance, it has an error ratio $\frac{2\epsilon}{1-\epsilon}$ on $\widetilde{H}$ for both kNN and range queries, respectively.*

Proof. Firstly, we prove the shortest path query time. Since it needs to use Dijkstra's algorithm with each cell in $\widetilde{N}(O^{-1}(s))$ or $\widetilde{N}(O^{-1}(t))$ as a source to compute inter-path, and the size of $\widetilde{N}(O^{-1}(s))$ or $\widetilde{N}(O^{-1}(t))$ is $O(n)$, so its shortest path query time is $O(n)$ times the shortest path query time of algorithm *HM-SP* on the simplified height map. Thus, the shortest path query time is $O(\frac{n^2}{\log n}\log\frac{n}{\mu})$.

Secondly, we prove the *kNN* and range query time. Since we just need to use algorithm *HM-SP-LQT1* on the simplified height map once for both the *kNN* and range query, the *kNN* and range query time is $O(\frac{n^2}{\log n}\log\frac{n}{\mu})$.

The memory usage and error guarantee of algorithm *HM-SP-LQT1* on the simplified height map are the same as algorithm *HM-SP* on the simplified height map. The error guarantee of algorithm *HM-SP-LQT1* on the *TIN* is the same as algorithm *HM-SP* on the *TIN*. The error ratio of both the *kNN* and range query algorithm of algorithm *HM-SP-LQT1* on the simplified height map (using the height map as the 3D surface for calculating the optimal distance) is similar to algorithm *HM-SP* on the simplified height map. The reason that there is no error ratio guarantee of both the *kNN* and range query algorithm of algorithm *HM-SP-LQT1* on the simplified height map (using the *TIN* as the 3D surface for calculating the ground-truth distance) is still similar to algorithm *HM-SP* on the simplified height map. □

Theorem D.17. *The shortest path query time, kNN and range query time and memory usage of algorithm HM-SP-LQT2 on the simplified height map are $O(\frac{n}{\mu}\log\frac{n}{\mu})$, $O(\frac{nn'}{\log n}\log\frac{n}{\mu})$ and $O(\frac{n}{\mu})$, respectively. Compared with $\Pi(s,t|H)$, i.e., the optimal distance, it returns an approximate shortest path passing on ϵ-approximate simplified height map $\widetilde{H}$ of H. Compared with $\Pi(s,t|T)$, i.e., the ground-truth distance, it returns an approximate shortest path passing on a TIN (that is converted from the height map). When using the height map as the 3D surface for calculating the optimal distance, it has an error ratio $\frac{2\epsilon}{1-\epsilon}$ on $\widetilde{H}$ for both kNN and range queries, respectively.*

Proof. We prove the *kNN* and range query time. Since we need to use algorithm *HM-SP* on the simplified height map n' times for both the *kNN* and range query, the *kNN* and range query time is $O(\frac{nn'}{\log n}\log\frac{n}{\mu})$.

The shortest path query time, memory usage and error guarantee of algorithm *HM-SP-LQT2* on the simplified height map are the same as algorithm *HM-SP* on the simplified height map. The error guarantee of algorithm *HM-SP-LQT2* on the *TIN* is the same as algorithm *HM-SP* on the *TIN*. The error ratio of both the *kNN* and range query algorithm of algorithm *HM-SP-LQT2* on the simplified height map (using the height map as the 3D surface for calculating the optimal distance) is similar to algorithm *HM-SP* on the simplified height map. The reason that there is no error ratio guarantee of both the *kNN* and range query algorithm of algorithm *HM-SP-LQT2* on the simplified height map (using the *TIN* as the 3D surface for calculating the ground-truth distance) is still similar to algorithm *HM-SP* on the simplified height map. □

Theorem D.18. *The shortest path query time, kNN and range query time and memory usage of algorithm HM-SP-LS on the simplified height map are $O(n\log n)$, $O(n\log n)$ and $O(n)$, respectively. Compared with $\Pi(s,t|H)$, i.e., the optimal distance, it returns an approximate shortest path passing on ϵ-approximate simplified height map $\widetilde{H}$ of H. Compared with $\Pi(s,t|T)$, i.e., the ground-truth distance, it returns an approximate shortest path passing on a TIN (that is converted from the height map). When using the height map as the 3D surface for calculating the optimal distance, it has an error ratio $\frac{2\epsilon}{1-\epsilon}$ on $\widetilde{H}$ for both kNN and range queries, respectively.*

Proof. Firstly, we prove the shortest path query time. Since it is applied on the simplified height map calculated by algorithm *HM-Simplify-LS* with $O(n)$ cells on $\widetilde{H}$, and we use Dijkstra's algorithm on $\widetilde{H}$ for once, the shortest path query time is $O(n\log n)$.

Secondly, we prove the *kNN* and range path query time. Since we just need to use Dijkstra's algorithm once for both the *kNN* and range query, the *kNN* and range query time is $O(n\log n)$.

Thirdly, we prove the memory usage. Since there are $O(n)$ cells on $\widetilde{H}$, the memory usage is $O(n)$.

The error guarantee of algorithm *HM-SP-LS* on the simplified height map is the same as algorithm *HM-SP* on the simplified height map. The error guarantee of algorithm *HM-SP-LS* on the *TIN* is the same as algorithm *HM-SP* on the *TIN*. The error ratio of both the *kNN* and range query algorithm of algorithm *HM-SP-LS* on the simplified height map (using the height map as the 3D surface for calculating the optimal distance) is similar to algorithm *HM-SP* on the simplified height map. The reason that there is no error ratio guarantee of both the *kNN* and range query algorithm of algorithm *HM-SP-LS* on the simplified height map (using the *TIN* as the 3D surface for calculating the ground-truth distance) is still similar to algorithm *HM-SP* on the simplified height map. □

Theorem D.19. *The shortest path query time, kNN and range query time and memory usage of algorithm HM-SP-LST on the simplified height map are $O(\frac{n}{\mu}\log\frac{n}{\mu})$, $O(\frac{n}{\mu}\log\frac{n}{\mu})$ and $O(\frac{n}{\mu})$, respectively. Compared with $\Pi(s,t|H)$, i.e., the optimal distance, it returns an approximate shortest path passing on ϵ-approximate simplified height map $\widetilde{H}$ of H. Compared with $\Pi(s,t|T)$, i.e., the ground-truth distance, it returns an approximate shortest path passing on a TIN (that is converted from the height map). When using the height map as the 3D surface for calculating the optimal distance, it has an error ratio $\frac{2\epsilon}{1-\epsilon}$ on $\widetilde{H}$ for both kNN and range queries, respectively.*

Proof. The shortest path query time, *kNN* and range query time, memory usage and error guarantee of algorithm *HM-SP-LST* on the simplified height map are the same as algorithm *HM-SP* on the simplified height map. The error guarantee of algorithm *HM-SP-LST* on the *TIN* is the same as algorithm *HM-SP* on the *TIN*. The error ratio of both the *kNN* and range query algorithm of algorithm *HM-SP-LST* on the simplified height map (using the height map as the 3D surface for calculating the optimal distance) is similar to algorithm *HM-SP* on the simplified height map. The reason that there is no error ratio guarantee of both the *kNN* and range query algorithm of algorithm *HM-SP-LST* on the simplified height map (using the *TIN* as the 3D surface for calculating the ground-truth distance) is still similar to algorithm *HM-SP* on the simplified height map. □