Lower-Bound Distance Queries under Partial Information

Swastik Biswas
New Jersey Institute of Technology
sb2785@njit.edu

Sohrab Namazi Nia
New Jersey Institute of Technology
sn773@njit.edu

Jees Augustine
Microsoft Corporation
augustine.jees@gmail.com

Suraj Shetiya
IIT Bombay
surajs@cse.iitb.ac.in

Senjuti Basu Roy
New Jersey Institute of Technology
senjutib@njit.edu

Gautam Das
University of Texas at Arlington
gdas@uta.edu

ABSTRACT

Given a metric space graph $G(V, E)$ in which only a subset of edge distances is known, we study a query-based model for answering lower bound distance queries on unknown edges, without access to any black-box distance oracle. For a queried pair of objects (u, v) with unknown distance, the lower bound corresponds to the minimum feasible value of $d(u, v)$ that satisfies all known distances and the triangle inequality constraints. We develop algorithmic techniques that operate directly on the partially observed graph to answer such queries efficiently. Our framework carefully balances preprocessing overhead, query processing time, and tightness of the produced bounds, while providing provable guarantees under metric assumptions. The proposed methods significantly improve both theoretical understanding and practical performance for lower bound inference, as validated through extensive experiments on multiple large-scale real-world datasets.

PVLDB Reference Format:
Swastik Biswas, Sohrab Namazi Nia, Jees Augustine, Suraj Shetiya, Senjuti Basu Roy, and Gautam Das. Lower-Bound Distance Queries under Partial Information. PVLDB, 19(11): 3807 - 3819, 2026.
doi:10.14778/3836663.3836727

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Swastik1993/metric-space-lb.git.

1 INTRODUCTION

Consider a complete undirected metric space graph $G(V, E)$ with $|V| = n$ objects and positive edge weights satisfying the triangle inequality. We are given distances only for a subset of edges $E' \subseteq E$, where $|E'| = m$, while the remaining edge weights are unknown. Given a query for an unknown edge $e = (u, v) \in E \setminus E'$, our goal is to compute the lower and upper bounds on $d(u, v)$ that are consistent with the known distances and all triangle inequality constraints. Importantly, no distance oracle is available to reveal additional unknown distances. Therefore, each query must be answered solely using the known subgraph $G' = (V, E')$ and the metric properties.

Our designed solutions return a range within which the unknown exact distance is guaranteed to lie in. The lower (respectively upper) boundary of that range corresponds to the lower (respectively upper) bound on the distance between the pair of query objects. The upper bound query processing problem has been shown to correspond to the classical shortest path computation problem between objects u and v in the known subgraph $G' = (V, E')$ [48]. Therefore, any existing works that study shortest paths in graphs *exactly*/*approximately* can be used to answer the upper bound distance problem. Shortest path computations in graphs have been heavily studied for decades, and hence are not the focus of our paper (Section 6 has more details). But surprisingly, the lower bound problem has not been well-studied at all. Thus the primary focus of our paper is on answering lower bound distance queries, i.e., computing the lower bound of the unknown distance between a pair of query objects. Unlike the upper bound, as we will explain, computing the lower bound is more complex, which we formalize through a process called edge folding.

Applications. This setting naturally captures what we refer to as the *edge resolution capability* of a metric space graph: the ability to compare unresolved edges and infer that one edge must be shorter than another even when their exact distances are unknown. Consider a road network represented as a metric space graph. In a disaster-response scenario, rescue teams may need to be dispatched through partially damaged transportation networks where exact travel distances along some routes are unavailable due to disrupted connectivity, unavailable mapping services, or incomplete infrastructure information. Suppose there are two candidate routes to a destination. If one can infer that the upper bound distance of one route is no greater than the lower bound distance of the other route, then the former route can be safely selected without acquiring additional distance information. Such edge resolution capability relies on deriving tight lower and upper bounds on unknown distances using only a partially observed set of distances together, in settings where obtaining additional distance information may be entirely infeasible due to natural disasters, cyberattacks, military conflicts, or connectivity failures.

The aforementioned scenario naturally extends to k-NN applications. Suppose a rescue coordination center must identify the k closest hospitals or shelters to a disaster site when exact travel distances are unavailable due to damaged infrastructure or disrupted connectivity. By deriving tight lower and upper bounds from the available distance information, the process can confidently certify candidates that must belong to the nearest set and safely prune candidates that cannot. Consequently, accurate estimated bounds

Proceedings of the VLDB Endowment, Vol. 19, No. 11 ISSN 2150-8097.
doi:10.14778/3836663.3836727

enable effective proximity search without requiring any additional travel-distance computations.

However, similar challenges may also arise in privacy-preserving retrieval considering biological databases [17, 51], medical imaging applications [27, 32, 46], [2, 6], [7], [28], and video database search [14], where the underlying data or embeddings may be inaccessible due to privacy, operational, or compliance constraints. Only a limited subset of pairwise similarities or distances may be revealed by domain experts (e.g., doctors).

We make two key distinctions here. While graph-based indexing techniques for proximity search in vector databases [33], where distances are typically defined using ℓ_2 distance or cosine similarity, can be viewed as a specific type of metric-space graph, our setting is fundamentally different. Classical vector database methods assume direct access to the underlying embeddings and can therefore perform additional similarity computations. In contrast, we assume that the embeddings themselves are unavailable due to privacy constraints. Conventional vector-database proximity-search techniques do not thus readily apply. Moreover, our notion of a metric-space graph is more general and is not restricted to embedding-based graphs, requiring only that the underlying distance function satisfies the metric properties.

Key Related Works. The earliest related work we are aware of is by Shasha and Wang [48], whose ADM algorithm precomputes lower and upper bounds for all missing edges via dynamic programming, making it unsuitable for a per-edge problem. In comparison, Augustine et al. [4] propose SPLUB, which computes a lower bound for a queried missing edge. While more query-oriented, these methods are not explicitly designed to adapt to or leverage sparsity in the solution design. Pivot-based approaches such as LAESA [43] and TLAESA [35] are somewhat related but fundamentally different. They rely on selecting landmark nodes and invoking a distance oracle during preprocessing to compute distances between landmarks and all other nodes. Such methods are not applicable in our setting, where no additional oracle calls are permitted beyond the initially known distances. Similarly, query-dependent landmark approaches such as [40] focus on computing upper bounds via additional structural assumptions; extending such techniques to compute tight lower bounds without further oracle access remains non-trivial (refer to Section 3.1, which clearly illustrates its shortcoming).

1.1 Technical Contributions.

In **Section 2.2**, we define the approximate lower bound **(ALBQ)** problem, which returns an approximate/loose lower bound distance of a queried edge. The focus of our problem setting is on the trade-off between pre-processing/query time and quality of the result.

In **Section 3**, we introduce a computational framework for computing **ALBQ** based on *edge landmarks*, a generalization of node landmarks [35, 36, 43] that uses edges as references when a distance oracle is unavailable. Given k selected edge landmarks, lower bounds for a queried unknown edge are derived via the triangle inequality in $O(k)$ time. We show that selecting k edge landmarks to maximize the aggregate lower-bound quality is NP-complete, and prove that the objective is monotone and submodular [16]. Leveraging these properties, **Section 3.3** presents a greedy algorithm, GEL, with a $(1 - \frac{1}{e})$ approximation guarantee [31, 37], enabling $O(k)$ query-time lower-bound computation.

In **Section 4.1**, we further tighten the lower bounds of GEL by incorporating shortest-path tree constraints. Using *Lowest Common Ancestor (LCA)* and *Range Minimum Query (RMQ)* structures [5], potential improvement in lower bound by the addition of a landmark-edge can be evaluated efficiently during query processing. This optimization improves bound tightness without increasing preprocessing time, storage requirements, or query latency.

In **Section 4.2**, we introduce SGEL, a sampling-based variant of GEL that significantly reduces preprocessing cost. Combined with the greedy landmark selection strategy, SGEL achieves a preprocessing complexity of $O(sk(m + n \log n))$, where s is the sample size and k is the number of landmarks. We further derive the minimum sample size required to achieve a confidence level of δ and error bound of ϵ, providing a principled trade-off between computational efficiency and approximation quality.

The resulting approach achieves a principled trade-off between approximation quality and computational cost, reducing preprocessing time from prohibitive quadratic or higher complexity to a scalable $O(sk(m + n \log n))$ framework, while retaining strong theoretical guarantees.

Section 5 presents a comprehensive experimental evaluation on four real-world datasets derived from travel and taxi ride services (NYC and Porto), road-network distances (POI London), a vector database, together with a large-scale synthetic dataset. We compare against several state-of-the-art techniques, including the recent TriScheme [4], as well as intuitive landmark-selection baselines.

Our experimental evaluation demonstrates that the proposed edge-landmark framework, particularly SGEL, achieves strong lower-bound quality, edge resolution capability (ERC), and downstream effectiveness across diverse datasets ranging from road networks to high-dimensional vector databases. SGEL consistently outperforms TriScheme and heuristic baselines in ERC, k-NN certification recall, and ERC capability in MST pruning and clustering, while remaining scalable to large graphs. Contrarily, SPLUB is computationally infeasible for larger datasets. The experiments further show that SGEL closely matches the lower-bound quality of GEL while reducing preprocessing cost by up to an order of magnitude, making it a practical alternative for large-scale deployments.

Related work is presented in § 6, and we conclude in § 7.

2 PRELIMINARIES & PROBLEM DEFINITION

In this section, we first introduce a toy running example, describe our data model, define necessary notations, and finally formalize the problem.

Example 1. *Consider a road network, where vertices correspond to locations and edge weights capture travel time/distance. Due to operational constraints (e.g., damaged infrastructure, unavailable sensors, communication limitations), obtaining additional shortest-path distance measurements is infeasible. Suppose distances are known only for a subset of location pairs (Figure 1). Assuming the distance function satisfies the metric properties, the objective is to estimate distances between unresolved pairs.*

2.1 Data Model

Object Set: We are given a set $O = \{o_i, o_2, \ldots, o_n\}$ of n objects, where the distance between each pair o_i, o_j is a value in the range

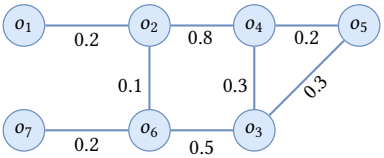


Figure 1: Example road network with 7 locations and the known distances between them.

$[0, \infty)$. The distance function d between any pair of objects satisfies the properties of a metric space, in particular, the triangle inequality property.

$$d(o_i, o_j) \leq d(o_i, o_k) + d(o_k, o_j)$$
$$d(o_i, o_j) \geq |d(o_i, o_k) - d(o_k, o_j)|$$

From Example 1(Figure 1), $\{o_1 \cdots o_7\}$ represent the objects (vectors), and the edge weights represent the shortest path distances. We also refer to the distance function d as the distance oracle in the paper.

Graph representation: Consider a complete undirected metric space graph $G(V, E)$, where the objects in O correspond to vertices V in G. In this graph, the distances are non-negative and satisfy the triangle inequality. As we only know m edge weights in G, this can be represented as a subgraph $G'(V, E' \subseteq E)$ where $|E'| = m$. The remaining $\binom{n}{2} - m$ distances are unknown and, hence, are missing in G'.

2.2 Concepts & Problem definition

Definition 1. **Exact Lower (Upper resp.) bound** *: Given a graph $G'(V, E')$ and a pair of objects, the exact lower (upper resp.) bound of edge-weight between them is the smallest (largest resp.) possible value that can be assigned to the distance between the object-pair without violating any triangle inequalities.*

We refer to the exact lower (upper resp.) as the tightest lower (upper resp.) interchangeably. Computing exact $[l, u]$ may be expensive, and a looser bound is often sufficient [4]. A loose lower (upper resp.) bound for an object-pair (o_i, o_j) is a value $l' \leq l$ ($u' \geq u$ resp.). A trivial lower and upper bound on an object pair could easily be assigned as 0 and ∞, respectively. Clearly, such loose bounds may not be useful later on. Note that a loose lower/upper bound violates one or more of the triangle inequalities in the metric space.

Applying the triangle inequality repeatedly, the upper bound of distance between o_i and o_j can be shown to be equal to the length of the shortest-path (sp) distance between the corresponding vertices [48] in G.

$$d(o_i, o_j) \leq sp(o_i, o_j) \tag{1}$$

Hence, $UB(o_i, o_j) = sp(o_i, o_j)$. Because of this property, algorithms designed for exact (approximate resp.) shortest path in general graph settings are applicable to compute exact (approximate resp.) upper bound distances.

Using Example 1, the exact upper bound for (o_5, o_7) could be obtained using the shortest path (SP) between o_5 and o_7 ($o_5 \rightarrow o_3 \rightarrow o_6 \rightarrow o_7$). Thus, $UB(o_5, o_7) = 1.3$.

Lower Bound: Unlike the upper bound (UB), computing the lower bound (LB) is more complicated. To compute the LB between (o_i, o_j), a naive approach is to enumerate every simple path between o_i and o_j and apply the triangle inequality. The largest edge-length for (o_i, o_j) that satisfies all the triangle inequalities is the lower bound for (o_i, o_j).

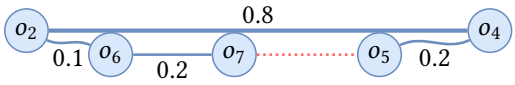


Figure 2: Computing $LB(o_5, o_7)$ by folding on (o_2, o_4)

A more efficient approach is the application of generalized triangle inequality [4, 48], which is termed as *edge folding*, formally defined below.

Definition 2 (Edge folding). *Given a simple path p (represented as a set) connecting o_i to o_l and an edge (o_j, o_k) that belongs to the path (i.e., $(o_j, o_k) \in p$), the edge-folding operation is defined as applying the generalized triangle inequality on the given path around (o_j, o_k). Mathematically, it is defined as,*

$$Fold(p, (o_j, o_k)) = 2 * d(o_j, o_k) - \texttt{len}(p) \tag{2}$$

where $\texttt{len}(p)$ is the length of the given path. Note that p is a set of edges in E' (i.e. $p \subseteq E'$) connecting o_i and o_l. Path p is represented as a set, $p = \{(o_i, o_x), (o_x, o_y), \ldots, (o_j, o_k), \ldots, (o_z, o_l)\}$.

Intuitively, given a path, to obtain a non-zero lower bound, the folding operation needs to be performed on the longest edge in the given path. Figure 2 demonstrates how Equation (2) is applied to compute $LB(o_5, o_7)$ using (o_2, o_4) as the long edge.

For an edge (o_k, o_l) that serves as the longest edge in the folding operation of $LB(o_i, o_j)$, Augustine et al. [4] show that the paths connecting o_i to o_k and o_j to o_l must be shortest paths. We formally define an exact lower bound using this property as follows.

Exact lower bound using edge folding. To obtain the exact lower bound, edge folding needs to be repeated over every edge, and the maximum will be the tightest lower bound.

$$\begin{aligned} LB(o_i, o_j) = & MAXIMUM_{\forall (o_k, o_l) \in E} \\ & \{-sp(o_i, o_k) + d(o_k, o_l) - sp(o_l, o_j), \\ & \quad - sp(o_i, o_l) + d(o_l, o_k) - sp(o_k, o_j)\} \end{aligned} \tag{3}$$

where sp refers to the length of the shortest path. Note that the lower and upper bounds rely upon the existence of a path with known edge weights between the object-pair (o_i, o_j). In the absence of a path connecting o_i and o_j, the trivial lower (upper, respectively) bound of 0 (∞, respectively) is exact/tight.

Using Example 1, $LB(o_5, o_7)$ is calculated by applying the folding formula with edges (o_2, o_4), (o_3, o_6), ... as the longest edge and the lower bounds corresponding to these are 0.3, 0, ... respectively. The exact lower bound, which corresponds to the maximum of these values, is 0.3.

Approximate Lower Bound distance query(ALBQ): Given a graph $G'(V, E')$ and a query edge e, on the shortest-path distance between u and v, compute an estimate of the *tightest feasible lower bound* (i.e., the largest valid lower bound) on the shortest-path distance between u and v, using only the available structural and metric constraints in G'.

3 ALGORITHMIC FRAMEWORK FOR ALBQ

In this section, we present an algorithmic framework for the **ALBQ** problem. The framework we propose accepts a parameter k, preprocess the graph to obtain a simple data structure, which will then be used to perform query answering in $O(k)$ time.

3.1 k-Edge Landmarks

The overarching idea of this task is to identify a smaller subset $L \subseteq E'$ of the known edges, called *Edge Landmarks*, and make use of them in effectively answering the lower bound queries. While we formally define the problem of selecting the edge landmarks later in the section, our goal is to select k edges (which we denote as L) called the *Edge Landmarks* during the pre-processing stage, where k is an input parameter. We use L during query processing utilizing Equation 4.

Comparison with landmark-based methods. Classical metric-space indexing methods such as LAESA [36, 43] and TLAESA [35] rely on *node landmarks*, requiring access to a distance oracle to precompute distances between selected landmark nodes and all other objects. In contrast, our setting assumes that only a partially observed metric-space graph is available and no additional distance queries can be issued. Consequently, node-landmark techniques are not directly applicable. Our notion of *edge landmarks* is inspired by node landmarks but is specifically designed for settings without oracle access.

Comparison with shortest-path (SP) landmarks [40]. SP-landmark methods use landmark nodes and shortest-path trees to derive upper bounds by stitching paths through landmarks. While a lower-bound counterpart could be envisioned, this related work does not trivially adapt. Furthermore, even an extended SP-landmark approach may yield only trivial lower bounds in cases where edge landmarks produce informative bounds (Refer to Section 1 of the Technical Report [9] for further details). We formulate edge-landmark selection as an optimization problem, establish its monotone submodular structure, and provide approximation guarantees. A key subproblem that arises is the problem of finding the longest edge in the stitched path efficiently, which we investigate in Section 4.1.

3.2 ALBQ with k-Edge Landmarks

We now formally define the concept of edge-landmarks.

Definition 3. ***Edge landmarks**: Edge landmarks are a set of edges in the graph $G'(V, E')$ denoted by L that are used in the edge folding operation (Equation* (3)*) to answer LB queries for a given missing edge.*

By leveraging Equation 3, one can compute the lower bound (LB) values involving every edge of graph G'. The tightest lower bound value for any object pair (o_i, o_j) is thus the maximum of these LB values.

Given a set of edges L, the lower bound found for any missing object-pair can be computed by applying triangle equality on each of the edges $e \in L$.

This process can be best described by the equation,

$$\begin{aligned} LB((o_i, o_j),\ L)\ =\ & MAXIMUM_{\forall (o_x, o_y) \in L} \\ & \{-sp(o_i, o_x) + d(o_x, o_y) - sp(o_y, o_j), \\ & -sp(o_i, o_y) + d(o_y, o_x) - sp(o_x, o_j)\} \end{aligned} \tag{4}$$

where L is the subset of edges of G' used in the computation.

For a given set of edge landmarks L, a naive computation of lower bounds for an object-pair (o_i, o_j) involves, (i) computing the shortest path (SP) between the vertices in L and the given object-pair (o_i, o_j), and (ii) go over the k edges in the set L to compute the LB value. Hence, the overall query processing using a given set L of landmarks is dominated by the SP computation, which takes $O(m + n\log(n))$ time for the naive approach.

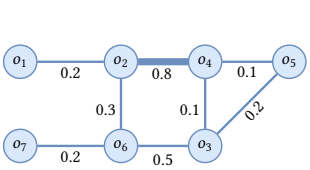


Figure 3: Edge landmark $\{(o_2, o_4)\}$ in Toy Example 1.

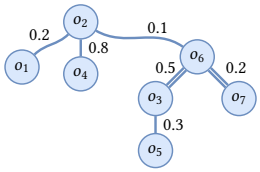


Figure 4: SP tree with source o_2.

A significant portion of the query time is repeatedly spent computing SP distances from the endpoints of edges in L to the two objects in each object-pair. While the object-pair changes from one query to another, the landmark set L remains fixed. This naturally suggests pre-computing the distances from the vertices incident to L to all other vertices in the graph so that we can reuse them across queries. However, this is not a lightweight step. Since L contains k edges (and up to $2k$ endpoints), we must run Dijkstra's algorithm from each of these vertices, resulting in a preprocessing cost of $O(k(m + n\log(n)))$. In addition, storing the distances from these vertices to all n vertices in the graph requires $O(kn)$ space. Although this upfront effort reduces the per-query computation to $O(k)$—as each lower bound can then be computed using simple lookups—it comes at the expense of considerable preprocessing time and memory usage.

To summarize our data structures, k-edge landmarks is a set of k known edges in G'. Along with the landmark edges, the SP distances from each of the endpoints of the k-edges is computed and stored to aid in query processing.

Using Example 1, consider the set of edge landmarks of size $k = 1$ to be $\{(o_2, o_4)\}$ as shown in Figure 3. Let us consider a call to `GEL` on the missing edge (o_5, o_6). Using the edge landmark $\{(o_2, o_4)\}$, we get $LB(o_5, o_7) \geq 0.8 - 0.2 - 0.1 - 0.2 = 0.3$.

`Edge Landmark Ideation:` As we have seen so far, k-edge landmarks can be used to compute the lower bound value for any set of object-pairs in G'. Instead of going over all the edges in G', if we only go over L during the folding operation, we obtain a lower bound value which is loose compared to the one computed from Equation 3. Thus, the chosen set of edge landmarks L determines the quality of the approximation. In further sections, we answer the problem of selecting the k-edge landmarks.

k-Edge Landmark (`K-EL`) Problem: Given a graph $G'(V, E')$ and value k, find a set $L \subset E'$ with size k (i.e., $\texttt{len}(L) = k$), which maximizes the sum of lower bounds over all the object-pairs in G'.

Theorem 1. *k-EL problem is $\mathcal{NP}$-complete*

Proof. (*Sketch*): The proof is non-trivial, and it reduces an instance of the known NP-complete problem set cover [21] to an instance of k-EL problem. We refer to the technical report [9] for further details. □

3.3 Algorithm `GEL`

In this subsection, we prove the monotonicity and sub-modularity properties of the k-EL problem. Then, we propose `GEL`, which makes use of the monotone and sub-modular nature of the optimization

function to solve the k-EL problem. Finally, we analyze the time complexity of our approach and prove the approximation factor.

Monotonicity and sub-modularity of k-EL problem: The goal of the k-EL problem is to find a set of k edge landmarks that maximizes the sum of lower bound scores between all pairs of object-pairs in G'. Based on the problem definition, our approach targets the optimization function that maximizes the lower bound score over all edges in G'.

Given a set of landmarks, the goal function F is defined as the sum of the lower bound scores over all object-pairs. The mathematical formula that captures this formulation is,

$$F(L) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} LB((o_i, o_j),\ L) \tag{5}$$

where L is the set of landmarks and the function LB is defined in Equation 4. The objective function in Equation 5 is both monotone and submodular; a submodular function is a set function, where adding an element to a smaller set provides a larger gain than adding it to a larger set. The proof is provided in Theorem 2.

Theorem 2. *The goal function F for k-EL problem (Equation 5) is monotone and submodular.*

Proof. (*Sketch*) As each lower bound value is non-negative, any addition to the set of landmarks cannot decrease the score. Hence, the function is monotone.
Consider a subset of $S_1 \subset S_2$. By definition of monotone functions, the contribution of S_1 is at most that of S_2 i.e., $F(S_1) \leq F(S_2)$. If we add a new edge (o_p, o_q) to both S_1 and S_2, the contribution of $LB((o_i, o_j), S_1 \cup \{(o_p, o_q)\}) - LB((o_i, o_j), S_1)$ represents the gain for the edge (o_i, o_j). As $LB((o_i, o_j), S_1) \leq LB((o_i, o_j), S_2)$ by definition, the contribution to the subset S_1 is at least that of S_2. Hence, the function is submodular. □

3.3.1 Variations in the optimization function. In our formal definition, we define the optimization goal as maximizing the sum of lower bounds over all object-pairs. There are other relevant goal functions that one could consider. Some examples are (i) ℓ_2-norm optimization goal (ℓ_2-NOG): square of the sum of lower bounds over all object-pairs, or in general ℓ_p norm goal, (ii) Relative Lower Bound (RLB): sum of the ratio of lower bound using landmarks to OPT for every unknown edge in the graph, which is given by $\sum_{\{(o_i,o_j) \in E - E' | LB(e) \neq 0\}} \frac{LB((o_i,o_j),\ L)}{LB(o_i,o_j)}$, where L is the set of k-*edge landmarks*. We will soon see that our framework allows functions that satisfy monotone and sub-modularity properties, with no changes in the techniques. We will also add sufficient notes/details about the applicability of our methods to the ℓ_2 normed objective, and the RLB objective throughout the paper.

Note that the alternative optimization functions, such as ℓ_2-NOG and RLB, are both monotone and submodular. The proofs are similar to Theorem 2, which we omit due to lack of space.

The monotonicity and submodular property of the optimization function of k-EL problem naturally give rise to an approximation algorithm. In this section, we define the naive greedy approach, referred to as greedy edge landmarks and denoted by GEL. Later in the paper, we define an efficient sampling-based greedy approach.

3.3.2 GEL *Algorithm.* The set of landmarks L is initialized with an empty set. In the first iteration, the edge that contributes the most to all pairs' lower bound score (given in Equation 5) is added to L. Once L is updated, the next edge is chosen based on the marginal contribution to the function F. The optimization function that represents this goal is given by,

$$argmax_{e \in E' \setminus L}\ F(L \cup \{e\}) - F(L) \tag{6}$$

In second and subsequent iterations, the edge that maximizes the optimization function given in Equation 6 is added to L. The iteration stops when k edges are added to the landmark set L. GEL algorithm to compute the k-landmarks is given in Algorithm 1.

Algorithm 1: GEL – **Greedy k-ELM**

Input: Metric space graph $G'(V, E')$, size of landmarks k
Output: Set L of k edge landmarks

```
Compute all-pairs shortest paths;
L ← {};
while len(L) ≤ k do
    total[e] ← 0   ∀e ∈ E' \ L;
    foreach (o_i, o_j) ∈ E \ E' do
        foreach e ∈ E' do
            total[e] ←
              total[e] + LB((o_i, o_j), L ∪ {e}) − LB((o_i, o_j), L);
    Add the edge with the highest score in total to L;
return L;
```

Analysis of the GEL: The total taken by GEL algorithm 1 is $O(kn^2m)$ as proved in Theorem 3.

Theorem 3. *The time taken by* GEL *to select the k-edge landmarks is* $O(kn^2m)$.

Proof. (*Sketch*) The greedy algorithm starts by computing all-pairs shortest paths in time $O(nm + n^2 \log(n))$ time. For every object pair, the contribution towards F of any known edge can be computed in $O(1)$ time. As there are k iterations, the overall time to compute the k edge landmarks is $O(kn^2m)$. □

The optimization function in Equation 5 is both monotone and sub-modular. The work by Nemhauser *et al.* [31, 37] provides an approximation factor of $1 - \frac{1}{e}$ for any formulation that follows monotone and sub-modular properties. Hence, our greedy formulation also follows the approximation guarantee of $1 - \frac{1}{e}$, where e is the natural logarithm.

4 ALGORITHMIC OPTIMIZATIONS

In this section, we focus on two main aspects of optimization (i) Improving quality/tightness of the lower bounds produced without increasing the worst case query times, and (ii) Improving the preprocessing time to make our algorithms practical.

4.1 Optimizations for quality

We strengthen lower bounds by leveraging shortest-path (SP) trees rooted at landmark vertices. For any object pair (o_i, o_j), the unique tree path between them yields additional generalized triangle inequality constraints based on the path length and maximum edge weight, resulting in strictly tighter lower bounds compared to edge-based constraints alone. To preserve efficiency, we then preprocess

each SP-tree to support constant-time path queries using Lowest Common Ancestor (LCA) and Range Minimum/Maximum Query (RMQ) data structures [18] [5]. This enables computing both the total path length and the maximum edge on the path in $O(1)$ time after linear preprocessing, allowing efficient evaluation of each candidate edge's contribution within the greedy framework..

4.1.1 Key Definitions.

Definition 4 (Lowest Common Ancestor (LCA)). *Given a rooted SP tree τ_{o_r} and two vertices o_i and o_j, their* lowest common ancestor, *denoted by $\mathtt{LCA}_{o_r}(o_i, o_j)$, is the common ancestor of o_i and o_j that has the minimum distance to both o_i and o_j.*

To illustrate the definitions in this section, consider τ_{o_2} as shown in Figure 4. In τ_{o_2}, the LCA of o_5 and o_7 is o_6, which is expressed as $\mathtt{LCA}_{o_2}(o_5, o_7) = o_6$.

We now define the *vertex-root edge set* as below.

Definition 5 (Vertex-root edge set). *Given a tree τ_{o_r} and a vertex o_i, the set of edges that belong to the path between o_i and the root node o_r is termed the vertex-root edge set and is denoted by $\mathcal{VR}_{o_r}(o_i)$.*

In τ_{o_2} (Figure 4), vertex-root edge set for o_3 is given by the two edges connecting o_3 to the root o_2 and is represented as $\mathcal{VR}_{o_2}(o_3) = \{(o_3, o_6), (o_6, o_2)\}$.

We are now ready to define the vertex-vertex edge set.

Definition 6 (Vertex-vertex edge set). *Given a tree τ_{o_r} and two vertices o_i and o_j, the set of edges that belong to path between o_i and o_j in the SP-tree τ_{o_r} is denoted by $\mathcal{VV}_{o_r}(o_i, o_j)$ and is named as the vertex-vertex edge set.*

The vertex-vertex edge set for the vertices o_i and o_j in the tree τ_{o_r} can be represented using vertex-root edge set of o_i, o_j and $\mathtt{LCA}_{o_r}(o_i, o_j)$. This can be represented as follows,

$$\mathcal{VV}_{o_r}(o_i, o_j) = (\mathcal{VR}_{o_r}(o_i) \cup \mathcal{VR}_{o_r}(o_j)) \setminus \mathcal{VR}_{o_r}(\mathtt{LCA}_{o_r}(o_i, o_j))$$

In simpler terms, this is equivalent to the union of the vertex-root edge set of o_i and the vertex-root edge set of o_j, and then taking a set difference with the vertex-root set of the LCA of o_i and o_j, i.e., $\mathtt{LCA}_{o_r}(o_i, o_j)$.

In τ_{o_2} (Figure 4), vertex-vertex edge set for (o_3, o_7) is given by the edges connecting o_3 to the node o_7 and is represented as

$$\mathcal{VR}_{o_2}(o_3, o_7) = \{(o_3, o_6), (o_6, o_7)\}$$

This can also be represented using the above formula, by computing the three sets as follows:

$$\mathcal{VR}_{o_2}(o_3) = \{(o_3, o_6), (o_6, o_2)\}$$
$$\mathcal{VR}_{o_2}(o_7) = \{(o_7, o_6), (o_6, o_2)\}$$
$$\mathcal{VR}_{o_2}(\mathtt{LCA}_{o_2}(o_3, o_7)) = \mathcal{VR}_{o_2}(o_6) = \{(o_6, o_2)\}$$

Computing the same term using the formula, we get $\mathcal{VV}_{o_2}(o_3, o_7) = \{(o_3, o_6), (o_6, o_2), (o_7, o_6)\} \setminus \{(o_6, o_2)\}$. Hence,

$$\mathcal{VV}_{o_2}(o_3, o_7) = \{(o_3, o_6), (o_6, o_7)\}$$

.

4.1.2 Tree based constraints.

Consider a set of landmarks L and the SP-trees for the vertices in L (i.e. $\forall_{o_r \in L}\ \tau_{o_r}$). Every tree τ_{o_r} contains a path that connects o_i and o_j. We can apply the generalized triangle inequality on this path to produce additional constraints not already covered by edge landmark constraints defined in Equation 4. Formally, the combined equation is given by,

$$\begin{aligned} LB((o_i, o_j),\ L)\ =\ & MAXIMUM_{\forall (o_x, o_y) \in L} \\ & \{-sp(o_i, o_x) + d(o_x, o_y) - sp(o_y, o_j), \\ & \ - sp(o_i, o_y) + d(o_y, o_x) - sp(o_x, o_j), \\ & 2 \cdot \mathtt{max}_{e \in \mathcal{VV}_{o_x}(o_i, o_j)} \mathtt{len}(e) - \sum_{e \in \mathcal{VV}_{o_x}(o_i, o_j)} \mathtt{len}(e), \\ & 2 \cdot \mathtt{max}_{e \in \mathcal{VV}_{o_y}(o_i, o_j)} \mathtt{len}(e) - \sum_{e \in \mathcal{VV}_{o_y}(o_i, o_j)} \mathtt{len}(e) \quad \} \end{aligned} \tag{7}$$

Note that we multiply 2 with the length of the longest edge, as it is already contained once within the Vertex-vertex path $\mathcal{VV}_{o_x}(o_i, o_j)$.

To illustrate our approach with an example, consider the SP-tree rooted at the node o_2 as shown in Figure 4. Consider the lower bound computation for the object-pair (o_3, o_7). Using tree τ_{o_2}, the vertex-vertex path set $\mathcal{VV}_{o_2}(o_3, o_7)$ contains the edges $\{(o_3, o_6), (o_6, o_7)\}$, which are marked by a double edge in the figure. By applying Equation 7 on this path, the LB is computed as $2 \cdot 0.5 - (0.5 + 0.2) = 0.3$.

We now focus our attention on proving that the function given in Equation 7 is monotone and submodular.

Theorem 4. *The optimization function given in Equation 5, with Equation 7 as the function for LB, is monotone and submodular.*

Proof. (*Sketch*) The proof is akin to the ones in the previous section and omitted for brevity. □

4.1.3 Efficient Computation.

The greedy algorithm, at each step, relies on computing the lower bound contribution of an edge. With the modification of the function *LB* in Equation 7, our assessment function needs to be updated to compute the LB score "*quickly*" (with Equation 4 computation consumed $O(1)$ time per edge). There are two main requirements for the data structure, given a tree τ_{o_r}, and vertex pair (o_i, o_j) (i) find the length of the path connecting (o_i, o_j), *i.e.*, $\sum_{e \in \mathcal{VV}_{o_r}(o_i, o_j)} \mathtt{len}(e)$, and (ii) find the maximum edge along the path connecting (o_i, o_j), *i.e.*, $\max_{e \in \mathcal{VV}_{o_r}(o_i, o_j)} \mathtt{len}(e)$.

As the SP-tree from all the vertices in the graph has been stored as the first step of the greedy algorithm, we preprocess this data structure to answer the sum and max queries and to compute the contributions of the m edges towards an unknown object-pair (Step 8 in the GEL algorithm). Hence, computing contributions can be framed as a lower bound query answering problem over the updated set of landmarks (an additional edge).

Query answering: To enable query answering, we store additional data structures. To understand these data structures and their connections to our work, we formally define RMQ (Range Minimum Queries) queries.

Definition 7 (Range Minimum Query). *Given an array A, start index s, and end index e, Range Minimum Query finds the minimum value within the sub-array between $A[i]$ and $A[j]$, both endpoints included.*

Using linear space and linear time, Bender et al. [5] create a data structure that can answer RMQ queries in $O(1)$ time.

RMQ and LCA queries (defined in Definition 4) have been shown to be equivalent [5], and there exists a linear space/time transform between the two problems. Hence, the same data structure helps us answer both RMQ and LCA in $O(1)$ time with linear space and linear preprocessing time. Note that multiplying the RMQ array

by -1 and performing the preprocessing would be equivalent to performing Range Maximum Queries.

If one were to store the length of the path from the root of the tree to each node in the tree, i.e., $\mathcal{VR}_{o_r}(o_i)$ for all o_i, one could exploit the LCA queries to obtain the length of the path as follows,

$$\texttt{len}(\mathcal{VR}_{o_r}(o_i)) + \texttt{len}(\mathcal{VR}_{o_r}(o_j)) - 2 \cdot \texttt{len}(\mathcal{VR}_{o_r}(\texttt{LCA}_{o_r}(o_i, o_j)))$$

Similarly, finding the longest edge in a path of a given SP-tree can be viewed as a natural extension of the Range Maximum Queries over a tree. A transformation of the RMQ in trees to a "*Cartesian tree*" where each node of the tree represents an edge was studied by Chazelle [12], and later an algorithm which consumes $O(n \log(n))$ time was proposed by Demaine et al. [18]. The key idea behind the transformation is to remove edges in ascending order from the SP-tree and maintain a decremental connectivity data structure to obtain a Cartesian tree. Performing a LCA on this tree is equivalent to answering RMQ queries on the original tree.

4.2 Sampling based GEL

A large part of the time in each iteration of GEL was consumed in computing the contribution of each edge. To elaborate, as F adds the lower bound values over all object pairs, the exact computation relies on computing the lower bound values for $O(n^2)$ object pairs. To avoid the excessive computation, we propose a sampling approach to estimate the sum of lower-bound contributions. Although this sampling-based approach speeds up the running time of our algorithm, it comes at the cost of an approximation factor. We formally prove the relationship between sample size and error, and the running time of our approach later in this section.

At a high level, our approach begins with an empty set of landmark-edges L. In each iteration, an edge that contributes the most to F when added to L is computed. For this computation, s sample object-pairs which do not have an edge in G' are chosen uniformly at random. The contribution of an edge $e \in E'$ is computed based on the incremental improvement of F when e is added to L (using Equation 6) over the object-pairs in s. This value is then used to compute the next edge to be added to L. The sampling algorithm for the k-EL problem is given in Algorithm 2. We discuss the technical challenges of the sampling-based approach involved in the computation of the contribution to LB later.

Note about sampling: As the intended scenario of usage is when the graph is sparse, we use a rejection sampling-based approach which, in expectation, consumes $\frac{1-p}{p}$, where $p = \frac{\binom{n}{2}-m}{\binom{n}{2}}$. In case, m is near to $\binom{n}{2}$, rejection sampling is inefficient, as p tends to 0, *i.e*, $p \to 0$ when $m \to \binom{n}{2}$. Instead, when the graph is dense, we explicitly store the missing edges in an array. A missing edge can then be sampled uniformly at random in $O(1)$ time by performing a random lookup in this array. In both cases, we design an unbiased sampler to obtain a missing edge.

One key difference between the traditional method and the sampling-based greedy approach is the computation of the contribution of an edge (step 9 of Algorithm 2). Note that we discussed the expensive preprocessing approach, which computed the all-pairs SP initially. This was used in Section 4.1.3 to propose an efficient $O(1)$ time approach to compute contribution as well as for answering queries. But, a challenge in the sampling setting is that the shortest

Algorithm 2: SGEL – **Sampling Greedy k-ELM**

Input: Metric space graph $G'(V, E')$
1 , size of landmarks k, sample size s **Output:** Set L of k edge landmarks
2 $L \leftarrow \{\}$;
3 **while** $\texttt{len}(L) \leq k$ **do**
4 $total[e] \leftarrow 0 \quad \forall e \in E' \setminus L$;
5 $samples \leftarrow$ Sample s object-pairs (with replacement) from $E \setminus E'$;
6 **foreach** $(o_i, o_j) \in samples$ **do**
7 **foreach** $e \in E'$ **do**
8 $total[e] \leftarrow total[e] + LB((o_i, o_j), L \cup \{e\}) - LB((o_i, o_j), L)$;
9 Add the edge with highest score in the $total$ array to L;
10 **return** L;

paths from the n nodes are not available. Hence, the approach is not applicable here. Instead, for a sampled missing edge (o_i, o_j), we now look at computing the contribution using Equation 7. For an edge $e \in E'$, the idea is to compute the lower bound using edge landmarks alone (no optimization from Section 4) using SPLUB [4], and for the contribution using SP-Tree, we propose a Dynamic Programming approach which efficiently computes the LB. We only discuss the high-level details of the DP below due to lack of space.

4.2.1 Contribution of an edge. Consider a missing edge $(o_i, o_j) \in E \setminus E'$ that was sampled as part of the s samples. As part of the contribution calculation using edge landmarks alone, SP-Tree has already been constructed (for SPLUB [4]). For a given edge $(o_x, o_y) \in E'$, the contribution using SP-Tree needs to consider the contribution using τ_{o_x} and τ_{o_y}. The contribution is considered as the maximum value using τ_{o_x}, τ_{o_y}, and edge-landmarks. Note that constructing τ_{o_x} or τ_{o_y} is expensive and hence, these are never materialized. Additionally, for our discussion, we consider the computation using τ_{o_x}, the computation for τ_{o_y} follows along similar lines.

To summarize, given the SP-Tree τ_{o_i} and τ_{o_j}, our goal is to find the LCA of nodes o_i and o_j in τ_{o_x}. Note that the shortest path from o_i to o_x in τ_{o_i} will be the same path from o_x to o_i in τ_{o_x}. Hence, a naive approach can consider two pointers at the node o_x in the two SP-Trees τ_{o_i} and τ_{o_j} respectively. Both pointers go up the tree in lock step. If at any step (say when both nodes are at o_j), the parent nodes differ, i.e., say $\texttt{parent}(\tau_{o_i}, o_j) \neq \texttt{parent}(\tau_{o_j}, o_j)$, then o_j, then o_j is returned as the LCA node. Using this node, say o_j, one can easily find the path length to the root and the maximum edge in the path from o_j to the root node in $O(1)$ time. However, computing the LCA node from every node in the tree can consume linear time, and hence, we could spend quadratic time per sample.

To avoid this expensive preprocessing, we exploit the observation that the tree constraint can be rewritten using only $\texttt{LCA}_{o_x}(o_i, o_j)$ and root-to-vertex paths in the SP-trees rooted at o_i and o_j. This allows us to replace the naive approach with a DP, which memoizes reverse-LCA computation. Starting from a vertex o_x, we simultaneously traverse upward in τ_{o_i} and τ_{o_j} while the corresponding nodes remain identical. After resolving the query o_x, all nodes that

lie in the path from o_x to the LCA node are updated to point to the LCA node of o_x, as they have the same result. In summary, our approach memoizes previously computed results in a lookup table and is reused whenever encountered, allowing future traversals to terminate early. Since each vertex is inserted into the lookup table at most once, all n LCAs can be computed in $O(n)$ time and $O(n)$ space per sampled edge. Consequently, the preprocessing cost is reduced from $O(n^2)$ to linear time while still supporting constant-time contribution evaluation thereafter.

4.2.2 Number of samples. A key component of the algorithm is s, the number of samples. As the quality of the result in each iteration relies on the number s, we capture the relationship between the error ϵ with a confidence $1-\delta$ and sample size s in Theorem 5.

Theorem 5. *Given a metric space graph $G(V,E)$ and an edge e, the number of object-pair samples s required to obtain an estimate on the sum of lower bounds between all pairs of objects within an error of ϵ and a confidence of $1-\delta$ is $s \geq \frac{c^2 \ln(2/\delta)}{2\epsilon^2}$, where c is the diameter of the graph.*

Proof. (*Sketch*) For this proof, we apply Hoeffding's inequality.

$$P(|S - \mathbb{E}[S]| \geq t) \leq 2\, exp(-\frac{2t^2}{\sum_{i=1}^{s}(b_i - a_i)^2})$$

with $a_i = 0$ and $b_i = c$ where c is the diameter of the graph G. Simplifying the equations, we get the minimum value of s that satisfies the constraints on ϵ and δ. □

Based on the theorem above, we design a sampling-based greedy approach called SGEL, which is presented in Algorithm 2. The key difference from the previous approach is the sub-problem: *given a missing edge (o_i, o_j) as a sample, find the additional contribution to the LB of (o_i, o_j) if the known edge e was added to the set L.* This sub-problem can be solved efficiently using a variation of the SPLUB algorithm presented in Augustine et al. [4], which consumes $O(m + n\log(n))$ time. Hence, the edge contribution when using Equation 4 can be computed in $O(m + n\log(n))$ time per sampled edge. With k rounds of s samples each, we get a total of sk iterations, each of which consumes $O(m + n\log(n))$ time. Thus, when using Equation 4, the time complexity is $O(m + n\log(n))$.

Note about other error measures: The proof can be tailored to address the value of sample size for the other error measures like ℓ_2-NOG, RLB etc. For example, for ℓ_2-NOG one can consider the square of the diameter as the value of b_i and $a_i = 0$ in Hoeffding's inequality. As the error for RLB is limited to $[0,1]$, one can consider $b_i = 1$ and $a_i = 0$ to obtain the sample size s needed for the proof.

We now look at the analysis of our approach.

Analysis of SGEL: The preprocessing time of SGEL relies on the number of samples chosen. For our analysis, consider the sample size to be s based on the values of δ and ϵ.

For each sample in the s samples, our approach computes the LB of an edge using Equation 4. As the Equation relies on SP values between the two endpoints of the sampled edge and the rest of the graph, Dijkstra's algorithm needs to be run from both endpoints on the sampled edge. As there are s samples in each of the k iterations, the total amount of time consumed to compute the k-edge landmarks is $O(sk(m + n\log(n)))$ *i.e.* $2sk$ shortest paths. The expected running time is presented below.

Theorem 6. *The expected time needed to calculate the k edge-landmarks is $n(1-(1-\frac{2}{n})^{sk})(m + n\log(n))$.*

5 EXPERIMENTAL EVALUATION

5.1 Experimental setup

System details. All methods are implemented in C++[1] and run on an AMD EPYC 7753 CPU @ 2.45GHz, Linux (Kubuntu 22.04.5 LTS), 64GB RAM. Table 1 presents the dataset descriptions.

- **Porto dataset**: Taxi trajectories in Porto, Portugal. Start/end coordinates are clustered into 4900 clusters; each trip maps to a weighted edge between clusters.
- **POI dataset**: Tourist POIs from 12 cities (Lonely Planet); pairwise shortest-path distances are obtained via OpenStreetMap.
- **NYC dataset**: NYC Taxi Trip data aggregated over zones; each trip yields an (origin, destination) pair with an observed travel distance. When needed, we apply a standard LP-based metric-repair procedure [45] to enforce metric consistency.
- **Uniform dataset**: 2000 points sampled uniformly from a 10D ball (synthetic), using ℓ_1 and ℓ_2 distances.
- **SIFT-128 dataset**: A vector database of 128-dimension. We construct a sparse weighted graph using the M24 setting in HNSWLIB with an average degree of 30.62.

Table 1: Statistics of the datasets used in the experiments.

Dataset	|V|	|E|
London (POI)	164	13,366
NYC	119	33,153
Uniform	2,000	7,998,000
Porto	4,900	12,002,550
SIFT-128	50,000	765,581

Graph construction/topology. For real-world datasets (with the exception of the SIFT-128 dataset), we use the well-known Renyi-Erdos graph generation technique to select a graph topology. The edge weights are added based on the distance between the objects in the dataset. For synthetic experiments, we use both Renyi-Erdos and Waxman [49] graph generation.

Baselines. For **ALBQ** (approximate lower bounds), we compare: (i) GEL, SGEL, (ii) TriScheme [4] (referred to as TriScheme in tables), and (iii) two landmark-selection heuristics.

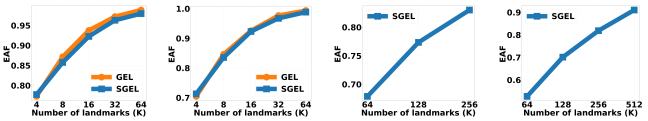
The two heuristics that sample edges are: (i) k *Longest Edges* which chooses the k longest edges in the graph as the k-edge landmarks, which is referred to as kLE in our experiments, and (ii) k *Farthest Edges*, which selects k edges which are "*farthest*" from each other to get a good "*spread of edges*" in the metric space graph. To measure distance between two edges, we use shortest paths between the endpoints of the edges and choose the ones that are farther apart. We refer to emph$k$ Farthest Edges as kFE in our experiments.

Parameters. We vary the number of nodes n, the number of known-distance edges (reported as density d), and the number of edge landmarks k. In some cases, we present the results using % densities, which is the number of known distances over the total number of edges. For SGEL, we use a sampling budget s derived from (δ, ϵ); unless otherwise specified we set $\delta = 0.01$ and $\epsilon = 0.1$, which yields $s \geq 265$ (assuming diameter $c = 1$), and we use $s = 265$ by default.

[1] Codebase : https://github.com/Swastik1993/metric-space-lb.git

Table 2: Edge resolution capability (ERC, %) - higher is better.

Dataset	NYC ($d = 0.10$)	London POIs ($d = 0.10$)	Porto ($d = 0.02$)	Uniform ($d = 0.05$)
SPLUB	**24.1822**	**58.3357**	**38.4200**	**99.9998**
GEL	**18.9649**	**47.8974**	–	–
SGEL	**18.4452**	**47.1709**	**21.6874**	**99.7700**
TriScheme	13.6484	38.2280	16.9262	34.8000
kLE	10.7674	18.0723	17.6754	99.7840
kFE	14.2829	31.8087	18.1448	99.7840



(a) London **(b) NYC** **(c) Porto** **(d) SIFT**

Figure 5: EAF varying number of landmarks. London and NYC compare GEL and SGEL, while Porto and SIFT report SGEL only.

5.2 Quality Evaluation

5.2.1 Edge Resolution Capability (ERC). We measure the *edge resolution capability*: given two edges e_1 and e_2, 'is $dist(e_1) < dist(e_2)$?" is resolved if $\text{LB}(e_2) > \text{UB}(e_1)$ or $\text{LB}(e_1) > \text{UB}(e_2)$.

Results Across Datasets. Table 2 reports ERC across datasets at representative densities considering a random sample of 10,000 unknown edges. GEL achieves the highest ERC where computable (NYC and London), providing an upper baseline for resolvability. SGEL is consistently strong and is the best method on **Porto** and **Uniform**, where it improves substantially over TriScheme and both heuristics. As the number of nodes increases, for example, Porto and Uniform outperform the heuristics kLE and kFE, and TriScheme. For SGEL, we observe that ERC initially increases with graph density, but plateaus beyond a certain point because rejection sampling becomes progressively less effective.

5.2.2 Empirical Approximation Factor. We define the *Empirical Approximation Factor (EAF)* of an approximate lower-bound method M relative to the tightest/Optimal lower bound as

$$\text{EAF}(G, M) = \frac{1}{|Q|} \sum_{(o_i,o_j)\in Q} \frac{LB_M(o_i, o_j)}{LB(o_i, o_j)} \tag{8}$$

where Q denotes the set of queried object pairs, $LB_M(o_i, o_j)$ is the lower bound returned by method M for pair (o_i, o_j), and $LB(o_i, o_j)$ (Equation 3) is the corresponding tightest lower bound. Thus, EAF measures the average approximation factor of M over Q.

We use a random sample size of 10,000 unknown edges to calculate the EAF values. On the London and NYC datasets (Figure 5, GEL achieves EAF values of 0.9886 and 0.9936, respectively, while SGEL achieves EAF values of 0.9796 and 0.9882. These results indicate that SGEL remains very close to the exact GEL lower bounds while substantially reducing preprocessing cost.

On the Porto dataset, exact GEL is not feasible due to very high preprocessing time, but SGEL still achieves an EAF of 0.8299, significantly outperforming TriSearch, kLE, and kFE. For the SIFT-128 dataset, SGEL achieves an EAF of 0.8187 with 256 landmarks, which further improves to 0.9103 when using 512 landmarks.

5.2.3 Downstream Application: kNN Certification. We evaluate how bounds support kNN search. Since certification is conservative, all methods achieve 100% precision by construction; we therefore report **recall**. Specifically, we report Recall@K (in-K): fraction of true top-K neighbors certified as *definitely in*. We also report **Recall@K (out-of-K)**: fraction of non-top-k-neighbors certified as *definitely not in*. For query evaluation, we randomly select a fixed set of source nodes from the original graph: **50 query nodes** for London datasets and **100 query nodes** for the Porto datasets. The value of top-K is varied for each configuration. For the $n-1$ edges that are incident on the given query node q, approximate lower bound is computed. Exact upper bound is produced using Dijkstra's shortest path algorithm from q to the rest of the $n-1$ nodes in the graph. We additionally conduct a kNN certification experiment on the SIFT dataset to demonstrate that it also applies to high-dimensional vector data by using a fixed set of 200 query nodes.

Results. Tables 3, 6 summarize certification recall on Porto and POI London. On **Porto**, SGEL is the only method that consistently certifies true neighbors: TriScheme certifies *none* of the true top-K neighbors (Recall@K(in)=0 across $K \in \{100, 200, 300, 400, 500\}$), while SGEL achieves non-trivial recall that increases with landmark size (k). For out-of-K certification, SGEL also yields the strongest recall, improving over TriScheme and the heuristics across all k. On **POI London**, GEL provides the highest recall (as expected), while SGEL remains competitive and consistently outperforms kLE/kFE, with TriScheme typically between Exact and Sampling. Overall, these results show that SGEL translates tighter bounds into materially better certification performance, particularly on large graphs. On the SIFT dataset, SGEL consistently outperforms TriScheme (Table 7), improving the out-of-(K) pruning recall by approximately (2×) and the in-(K) certification recall by approximately (1.5×). The presented ERC values on SIFT are weaker than those observed on the other datasets, which is expected given the high dimensionality of the underlying vector space. Nevertheless, the results demonstrate that our lower-bound estimation techniques still convincingly outperform the SOTA in challenging high-dimensional real world vector databases.

5.2.4 Downstream Application: Minimum Spanning Tree. We next evaluate minimum spanning tree (MST) construction using Prim's algorithm [39]. The effectiveness is measured using the percentage of Edge Resolution Capability (ERC %). For London, using 0.10 density, 64 landmarks and 300 samples. For NYC, we use density 0.10, $k = 64$, and 300 samples. Table 8 reports the MST results for London and NYC. SPLUB has the highest ERC% as expected, while GEL and SGEL remain close to SPLUB in ERC, and TriSearch gives much weaker edge resolution.

For Porto, we use density 0.02, 256 landmarks, and 300 samples. SPLUB and GEL were not computed in this large-graph run due to timeouts. Table 8 reports that SGEL has the highest ERC, saving 68.17% of oracle calls, compared with 51.48% for TriSearch and about 44% for kFE and kLE.

5.2.5 Downstream Application: Clustering. We finally evaluate the effectiveness by quantifying ERC of our proposed solution for PAM [29] clustering technique. For London, we use the configuration as that of the MST experiment. Table 9 reports the PAM results for cluster size 10. On London, SPLUB resolves 43.83% of edges, while GEL and SGEL resolves 43.09% and 42.26%, respectively. TriSearch resolves only 25.29%. On NYC, the same trend

Table 3: Porto: Recall@K % (in-K) for certified kNN.

K	100	200	300	400	500
TriScheme	0.000	0.000	0.000	0.000	0.000
SGEL	**0.110**	**0.118**	**0.128**	**0.175**	**0.224**
kLE	0.000	0.025	0.047	0.050	0.062
kFE	0.000	0.050	0.050	0.055	0.068

Table 4: Porto: Recall@K % (out-of-K) for certified kNN.

K	100	200	300	400	500
TriScheme	26.962	22.624	20.561	19.226	18.340
SGEL	**33.109**	**27.699**	**25.243**	**23.625**	**22.745**
kLE	26.426	22.370	20.501	19.398	18.608
kFE	26.575	22.532	20.768	19.675	18.955

Table 5: London: Recall@K % (in-K) for certified kNN.

K	5	10	15	20
SPLUB	**6.000**	**7.000**	**8.400**	**10.700**
GEL	**2.800**	**3.400**	**5.333**	**6.400**
SGEL	**2.200**	**3.200**	**5.067**	**6.111**
kLE	0.400	0.600	0.933	1.300
kFE	0.800	1.200	1.867	2.500
TriScheme	0.400	0.200	0.133	0.400

Table 6: London: Recall@K % (out-of-K) for certified kNN.

K	5	10	15	20
SPLUB	**71.902**	**64.209**	**59.472**	**55.460**
GEL	**66.110**	**58.626**	**53.730**	**50.123**
SGEL	**64.577**	**57.288**	**52.626**	**48.982**
kLE	41.669	34.687	30.724	27.828
kFE	52.196	45.055	40.871	37.595
TriScheme	48.405	41.877	38.380	35.619

Table 7: SIFT kNN certification results.

top K = 150

Method	Out-of-K Recall%
TriScheme	0.366
SGEL	**0.714**

Recall@K % (out-of-K)

top K = 49,000

Method	In-Recall@K%
TriScheme	0.082
SGEL	**0.134**

Recall@K % (in-K)

Table 8: ERC% for Prim's MST.

Method	ERC%		
	London	NYC	Porto
SPLUB	**54.20**	**47.66**	–
GEL	**53.17**	**47.42**	–
SGEL	**52.35**	**47.25**	**68.17**
kLE	46.94	42.73	43.79
kFE	46.99	41.59	44.27
TriScheme	34.79	27.60	51.48

Table 9: ERC% of PAM.

Method	ERC%		
	London	NYC	Porto
SPLUB	**43.83**	**41.12**	–
GEL	**43.09**	**41.03**	–
SGEL	**42.26**	**40.89**	**21.89**
kLE	38.34	37.99	8.94
kFE	38.66	37.66	9.58
TriScheme	25.29	22.40	16.36

holds: SPLUB, GEL, and SGEL resolves about 41% of edges, while TriSearch resolves 22.40%.

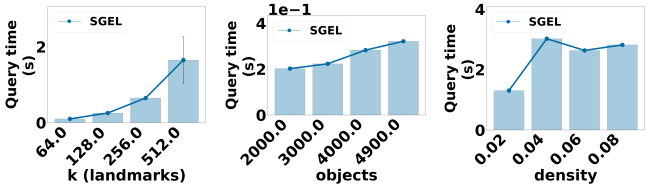
For Porto, we use density 0.02, $k = 256$, and 300 samples. SPLUB and GEL were not computed in this large-graph run due to time outs. Table 9 reports the Porto PAM results for cluster size $k_c = 10$. SGEL gives the strongest downstream pruning, saving 21.83% of exact-distance oracle calls, compared with 16.36% for TriSearch, 9.58% for kFE, and 8.94% for kLE. This confirms that SGEL provides the strongest scalable lower bounds for the PAM as well.

5.3 Scalability Evaluation

Having established the qualitative superiority of our proposed GEL and SGEL, we perform scalability evaluation along two key dimensions. (i) **query processing time**. (ii) **preprocessing cost** (time and memory). We vary key parameters.

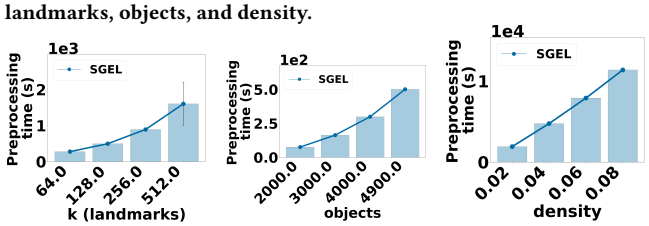
5.3.1 Query Processing Time. We report average query processing time and present the results in Figure 6. Figure 6a presents the query processing time by varying the number of landmarks. Whereas, Figures 6b and 6c present the query processing time by varying number of objects, and the density of the graph, respectively. These results demonstrate that query time scales smoothly with the number of objects and is relatively stable across densities, but increases substantially with the number of landmarks k (Figure 6). Thus, k provides an explicit quality-time knob: increasing k can improve resolvability/certification quality but increases per-query cost, and practical operating points use moderate k.

5.3.2 Preprocessing Time. We report average preprocessing time and present the results in Figure 7. Figure 7a presents the preprocessing processing time by varying number of landmarks. Whereas,



(a) Varying landmarks (b) Varying objects (c) Varying density

Figure 6: Taxi Porto: query time for SGEL under varying number of landmarks, objects, and density.



(a) Varying landmarks (b) Varying objects (c) Varying density

Figure 7: Taxi Porto: preprocessing time for SGEL under varying number of landmarks, objects, and density.

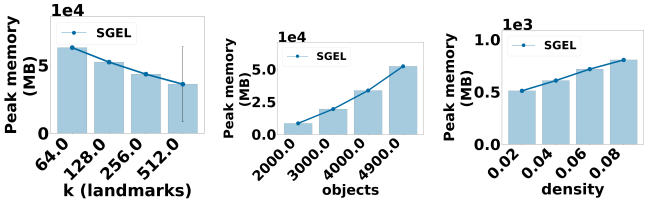
Figures 7band 7c present the preprocessing time by varying number of objects, and the density of the graph, respectively. These results confirm the expected behavior; more importantly, they reveal that preprocessing time increases sharply with k, establishing k as the primary quality–cost trade-off parameter.

5.3.3 Preprocessing Peak Memory. We report peak memory requirement in preprocessing and present the results in Figure 8. Figure 8a presents the preprocessing peak memory by varying number of landmarks. Interestingly, on Porto the peak memory *decreases* as k increases (Figure 8a), indicating that, for this workload, larger landmark sets are associated with smaller number of SP trees that needs to be maintained as part of the optimization process. Whereas, Figures 8b and 8c present the peak memory requirement by varying number of objects, and the density of the graph, respectively. These results demonstrate peak memory grows moderately with density, while it grows sharply with the number of objects, reflecting the large state required for large graphs.

5.3.4 Large Scale Dataset. Our running time analyses of the algorithms for the SIFT dataset are reported in Table 10. For SGEL, we vary landmark counts $k = 128/256/512$. The preprocessing time increases from 1,770s at $k = 128$ to 4,928.8s at $k = 512$, while the total query time remains practical and decreases slightly from 2.025ms at $k = 128$ to 1.972ms at $k = 256$ and 1.866ms at $k = 512$, during

Table 10: SGEL scalability on SIFT for varying landmark counts.

Metric	$k = 128$	$k = 256$	$k = 512$
Preprocessing time (s)	1770	3444.7	4928.8
Total Query time (ms)	2.025	1.972	1.866
ELM size (MB)	323	323	323



(a) Varying landmarks (b) Varying objects (c) Varying density

Figure 8: Taxi Porto: peak memory for SGEL under varying number of landmarks, objects, and density.

Table 11: GEL vs. SGEL preprocessing time and EAF tradeoff.

London			NYC		
k	Speedup	$\frac{\text{GEL RLB}}{\text{SGEL RLB}}$	k	Speedup	$\frac{\text{GEL RLB}}{\text{SGEL RLB}}$
4	14.788	1.017	4	6.614	1.007
8	17.956	1.000	8	8.859	1.006
16	21.088	1.005	16	10.717	1.004
32	21.185	1.004	32	12.130	1.004
64	25.123	1.003	64	12.826	1.002

which the Edge Landmark memory remains consistent at 323MB. These results demonstrate SGEL scales.

5.4 Preprocessing Time vs Quality Tradeoff

Table 11 shows that SGEL substantially reduces preprocessing time while preserving lower-bound quality close to GEL. On London, SGEL achieves a preprocessing speedup ranging from 14.788× at $k = 4$ to 25.123× at $k = 64$. On NYC, the speedup ranges from 6.614× at $k = 4$ to 12.826× at $k = 64$. Thus, the preprocessing advantage of SGEL becomes more pronounced as the landmark budget increases.

At the same time, the quality tradeoff remains very close to 1 across all settings. For London, the quality-tradeoff ratio ranges from 1.000 to 1.017, indicating that GEL has only a small EAF advantage over SGEL. At $k = 64$, the ratio is 1.003, meaning GEL's EAF is only about 0.26% higher than SGEL's, while SGEL is 25.123× faster in preprocessing. For NYC, the quality-tradeoff ratio ranges from 1.002 to 1.007, showing that SGEL and GEL are essentially equivalent in quality, with SGEL even slightly outperforming GEL for some smaller landmark settings. Overall, these results demonstrate that SGEL provides a favorable preprocessing-quality tradeoff: it closely matches GEL's bound quality while substantially reducing preprocessing cost.

5.5 Component Wise Analyses

We evaluate the main components of our optimization framework: edge-landmark selection, sampling-based landmark selection, SP-tree-based tightening, and LCA/RMQ-based query acceleration. Tables 12, 13, and 14 report the effect of these components on EAF, preprocessing time, and per-query time.

Table 12 shows that the optimized methods consistently improve EAF over the edge-folding-only variant. On London, GEL ℓ_1 improves from 0.793 to 0.988, and GEL RLB improves from 0.789 to 0.990. SGEL exhibits the same trend. The effect is largest on Porto, where SGEL ℓ_1 improves from 0.214 to 0.834, and SGEL RLB improves from 0.236 to 0.832.

Table 12: Optimized and unoptimized EAF across datasets.

Method	EAF					
	Optimized			Unoptimized		
	London	NYC	Porto	London	NYC	Porto
GEL ℓ_1	0.988	0.993	–	0.793	0.846	–
GEL RLB	0.990	0.995	–	0.789	0.847	–
SGEL ℓ_1	0.980	0.988	0.834	0.754	0.844	0.214
SGEL RLB	0.981	0.988	0.832	0.799	0.846	0.236

Table 13 shows that landmark selection dominates preprocessing time. For example, on London, GEL ℓ_1 spends 56.061 seconds in landmark selection out of 56.093 seconds total, while GEL RLB spends 61.851 seconds out of 61.875 seconds total. On NYC, the same pattern holds. The SGEL sampling optimization greatly reduces this cost while preserving quality. On London, SGEL ℓ_1 reduces preprocessing from 56.093 seconds to 3.601 seconds, a 15.6× speedup, while EAF decreases only from 0.988 to 0.980. Porto further confirms that landmark selection is the main contributor to total preprocessing time, while SP-tree construction and LCA/RMQ preprocessing contribute much smaller overhead.

Table 14 reports per-query time in microseconds. The query breakdown shows that LCA/RMQ lookup is the dominant query component, but the total query latency remains microsecond-level on London/NYC and below 0.4 ms/query on Porto.

Overall, this study shows that SP-tree tightening improves EAF quality, SGEL substantially reduces preprocessing time with only a small loss in quality, and LCA/RMQ preprocessing keeps query processing efficient.

5.6 Summary of results

Overall, the results demonstrate that SGEL provides the best balance between lower-bound quality and scalability. Across datasets, it consistently achieves the highest edge resolvability outperforming competing methods and, on large datasets such as Porto and SIFT, remaining the only practical approach that reliably resolve edges. Despite its significantly lower preprocessing cost, SGEL closely approximates GEL, achieving empirical approximation factor near 1 while providing speedups of up to 25×. Component-wise analysis shows that preprocessing is dominated by landmark selection, whereas query processing is primarily spent on LCA/RMQ and SP-tree computations. Query time scales linearly with the landmark budget k, making it the primary quality-cost tradeoff parameter. Finally, experiments on downstream tasks and high-dimensional datasets further confirm the effectiveness and robustness of SGEL.

6 RELATED WORK

6.1 Computing Upper and Lower Bounds

There is a large volume of work in computing upper and lower bounds of an unknown edge (exactly or approximately) [11, 19, 20, 22–24, 40, 44]. While the problem of estimating the upper bound is

Table 13: Preprocessing time breakdown across datasets.

Method	Preprocessing time (s)														
	London ($k = 64$)					NYC ($k = 64$)					Porto ($k = 256$)				
	Landmark selection	SPT	LCA/RMQ	Other	Total	Landmark selection	SPT	LCA/RMQ	Other	Total	Landmark selection	SPT	LCA/RMQ	Other	Total
GEL ℓ_1	56.061	0.015	0.017	0.000	56.093	14.618	0.007	0.009	0.000	14.634	–	–	–	–	–
GEL RLB	61.851	0.014	0.010	0.000	61.875	19.977	0.006	0.006	0.000	19.989	–	–	–	–	–
SGEL ℓ_1	3.573	0.015	0.013	0.000	3.601	1.529	0.007	0.008	0.015	1.559	12410.231	51.491	3.905	0.000	12465.627
SGEL RLB	3.596	0.013	0.011	0.000	3.620	1.548	0.006	0.006	0.000	1.560	12262.879	50.395	3.043	0.000	12316.317

Table 14: Query time breakdown across datasets in μs/query.

Method	Query time (μs)														
	London ($k = 64$)					NYC ($k = 64$)					Porto ($k = 256$)				
	Edge-folding	LCA/RMQ	SPT	Other	Total	Edge-folding	LCA/RMQ	SPT	Other	Total	Edge-folding	LCA/RMQ	SPT	Other	Total
GEL ℓ_1	0.556	13.689	1.058	0.134	15.437	0.457	10.899	0.883	0.124	12.363	–	–	–	–	–
GEL RLB	0.579	15.152	1.141	0.133	17.005	0.459	11.124	0.918	0.124	12.625	–	–	–	–	–
SGEL ℓ_1	0.558	14.243	1.112	0.133	16.046	0.449	10.725	0.906	0.124	12.204	16.204	308.150	35.017	0.878	360.249
SGEL RLB	0.560	13.901	1.099	0.132	15.692	0.447	10.697	0.870	0.124	12.138	15.298	302.162	32.668	0.891	351.019

a direct extension of finding the *shortest path between the pair of points*, these results do not extend on lower bound computation.

Sasha and Wang [48] proposed ADM, a dynamic programming based approach to compute the tight all pairs lower and upper bounds in a given a metric space, running in $O(n^3)$ time.

There are pivot-based methods proposed to accelerate the *metric space similarity search*. These algorithms start with a set of pivots(or node Landmarks or Landmarks) and later compute distances between the Landmarks and all other objects. The choice of pivots are important and the work by Zhu et al [10] experimentally evaluates the different choices and their practical implication in similarity search. Some of the examples include, AESA [36], LAESA [43] and TLAESA [35] and variants of TLAESA [34, 42].

In a recent work, Augustine et. al. [4] proposed a query answering interface which provides both upper and lower bounds for distances. Their algorithm Tri Scheme is inspired by LAESA [43] which readily adapts to dynamic changes in the graph. Tri Scheme runs with an expected lookup time of $O(m/n)$. The work also discusses SPLUB which produces the exact tightest lower bound in $O(m + n \log n)$ with no pre-processing cost.

To the best of our knowledge, no related work studies the lower bound distance query problem in the absence of a distance oracle.

6.2 Approximate Distances and Oracles

Approximate distance oracles are oracles that can produce the shortest path distance between any pair of points in a graph. This is achieved by initially analyzing the graph using a pre-processing step, and constructing a data-structure which can be further queried for obtaining pairwise distances(shortest path) in constant time.

The seminal work in this area is by Thorup-Zwick [47], which allows a constant query time with an expected preprocessing time of $O(kmn^{(1/k)})$ to compute a data structure of size $O(kn^{(1+1/k)})$ where k a positive integer, n is the number of nodes in the graph and m is the number of edges in it. The shortest paths produced by this algorithm have a maximum error factor of $O(2k - 1)$. Other, principal works in this area include [13, 26, 41, 50], and they achieve this using sampling the points either probabilistically or greedily to ensure that the data-structure maintains points that ensure the shortest path distances are within a given error-factor.

Graph Spanners (or Spanner), in contrast, are a subgraph to the original graph which preserve the shortest path length between every pair of points with some dilation or error. The dilation or error is bounded by a multiplicative or an additive called a *stretch-factor*. Finding sparse spanners which can ensure the minimal dilation is a class of problems that are closely associated to estimating Approximate Distances. Classical results in this area are by Peleg and Schäffer [38], Althöfer et al. [3], and Kortsarz and Peleg [30]. Optimizing on getting the sparsest spanner with a constant *stretch-factor* has been explored in [1, 8, 15, 25, 41].

The error bounds on **ALBQ** *are seldom addressed in related works, which is our main focus.*

7 CONCLUSION

We study the problem of answering lower bound distance queries in partially observed metric space graphs without access to a distance oracle. We formalize the approximate lower bound queries (ALBQ) problems and propose a parameterized framework that enables a principled trade-off between preprocessing cost, query time, and bound tightness. Our introduction of edge landmarks, along with submodularity-based greedy selection (GEL) and its sampling-based variant (SGEL), provides strong approximation guarantees while achieving $O(k)$ (effectively constant) query time for bounded k. Additional structural optimizations using shortest-path trees further tighten bounds without increasing query overhead. Extensive experiments demonstrate that our methods significantly improve bound quality and edge resolution capability, leading to measurable gains in three different downstream tasks.

ACKNOWLEDGMENTS

The work of NJIT researchers are supported by the following funding agencies: (1) National Science Foundation award number(s): 1942913 (2) Office of Naval Research award number(s): N000141812838, N000142112966, N000142412466. The work of Suraj Shetiya was supported by ANRF under Grant ANRF/ECRG/2024/004976/ENS.

REFERENCES

[1] Amir Abboud, Greg Bodwin, and Seth Pettie. 2018. A hierarchy of lower bounds for sublinear additive spanners. *SIAM J. Comput.* 47, 6 (2018), 2203–2236.

[2] Irwin E Alber, Ziyou Xiong, Nancy Yeager, Morton Farber, and William M Pottenger. 2001. Fast retrieval of multi-and hyperspectral images using relevance feedback. In *IGARSS 2001. Scanning the Present and Resolving the Future. Proceedings. IEEE 2001 International Geoscience and Remote Sensing Symposium (Cat. No. 01CH37217)*, Vol. 3. IEEE, 1149–1151.

[3] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. 1993. On sparse spanners of weighted graphs. *Discrete & Computational Geometry* 9, 1 (1993), 81–100.

[4] Jees Augustine, Suraj Shetiya, Mohammadreza Esfandiari, Senjuti Basu Roy, and Gautam Das. 2021. A generalized approach for reducing expensive distance calls for a broad class of proximity problems. In *Proceedings of the 2021 International Conference on Management of Data*. 142–154.

[5] Michael A Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. 2005. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms* 57, 2 (2005), 75–94.

[6] Andrew P Berman and Linda G Shapiro. 1998. Triangle-inequality-based pruning algorithms with triangle tries. In *Storage and Retrieval for Image and Video Databases VII*, Vol. 3656. SPIE, 356–365.

[7] Andrew P Berman and Linda G Shapiro. 1999. A flexible image database system for content-based retrieval. *Computer Vision and Image Understanding* 75, 1-2 (1999), 175–195.

[8] Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. 2013. Approximation algorithms for spanner problems and directed steiner forest. *Information and Computation* 222 (2013), 93–107.

[9] Swastik Biswas, Sohrab Namazi Nia, Jees Augustine, Suraj Shetiya, Senjuti Basu Roy, and Gautam Das. 2026. Technical Report. https://github.com/Swastik1993/metric-space-lb.

[10] Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. 2003. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters* 24, 14 (2003), 2357–2366.

[11] Edward PF Chan and Heechul Lim. 2007. Optimization and evaluation of shortest path queries. *The VLDB journal* 16 (2007), 343–369.

[12] Bernard Chazelle. 1984. Computing on a free tree via complexity-preserving mappings. *Algorithmica* 2 (1984), 337–361. https://api.semanticscholar.org/CorpusID:11701214

[13] Shiri Chechik. 2014. Approximate distance oracles with constant query time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. 654–663.

[14] S-S Cheung and Avideh Zakhor. 2005. Fast similarity search and clustering of video sequences on the world-wide-web. *IEEE Transactions on Multimedia* 7, 3 (2005), 524–537.

[15] Eden Chlamtáč and Michael Dinitz. 2016. Lowest-Degree k-Spanner: Approximation and Hardness. *Theory of Computing* 12, 1 (2016), 1–29.

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

[17] AFW Coulson, JF Collins, and A Lyall. 1987. Protein and nucleic acid sequence database searching: a suitable case for parallel processing. *Comput. J.* 30, 5 (1987), 420–424.

[18] Erik D. Demaine, Gad M. Landau, and Oren Weimann. 2009. On Cartesian Trees and Range Minimum Queries. *Algorithmica* 68 (2009), 610 – 625. https://api.semanticscholar.org/CorpusID:54285442

[19] Edsger W Dijkstra. 2022. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*. 287–290.

[20] Dujian Ding, Sihem Amer-Yahia, and Laks Lakshmanan. 2022. On Efficient Approximate Queries over Machine Learning Models. *Proc. VLDB Endow.* 16, 4 (dec 2022), 918–931. https://doi.org/10.14778/3574245.3574273

[21] Michael R Garey and David S Johnson. 1979. Computers and intractability. *A Guide to the* (1979).

[22] Andrew V Goldberg and Chris Harrelson. 2005. Computing the shortest path: A search meets graph theory.. In *SODA*, Vol. 5. 156–165.

[23] Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. 2006. Reach for A*: Shortest Path Algorithms with Preprocessing.. In *The shortest path problem*. Citeseer, 93–139.

[24] Bruce Golden. 1976. Shortest-path algorithms: A comparison. *Operations Research* 24, 6 (1976), 1164–1168.

[25] Lee-Ad Gottlieb and Liam Roditty. 2008. An optimal dynamic spanner for doubling metric spaces. In *European Symposium on Algorithms*. Springer, 478–489.

[26] Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. 2008. Approximate distance oracles for geometric spanners. *ACM Transactions on Algorithms (TALG)* 4, 1 (2008), 1–34.

[27] STEVEN HEILMAN. 2010. THE RADON AND FOURIER TRANSFORMS: THE MATHEMATICS OF X-RAYS AND CT-SCANS. (2010).

[28] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. 1993. Comparing images using the Hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence* 15, 9 (1993), 850–863.

[29] Leonard Kaufman and Peter Rousseeuw. 1987. *Clustering by means of medoids*. North-Holland.

[30] Guy Kortsarz and David Peleg. 1994. Generating sparse 2-spanners. *Journal of Algorithms* 17, 2 (1994), 222–236.

[31] Andreas Krause and Daniel Golovin. 2014. Submodular function maximization. *Tractability* 3, 71-104 (2014), 3.

[32] Tian Liu, Weiyu Xu, Pascal Spincemaille, A Salman Avestimehr, and Yi Wang. 2012. Accuracy of the morphology enabled dipole inversion (MEDI) algorithm for quantitative susceptibility mapping in MRI. *IEEE transactions on medical imaging* 31, 3 (2012), 816–824.

[33] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[34] Luisa Micó and Jose Oncina. 2014. Dynamic Insertions in TLAESA Fast NN Search Algorithm. In *2014 22nd International Conference on Pattern Recognition*. IEEE, 3828–3833.

[35] Luisa Micó, José Oncina, and Rafael C Carrasco. 1996. A fast branch & bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters* 17, 7 (1996), 731–739.

[36] María Luisa Micó, José Oncina, and Enrique Vidal. 1994. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters* 15, 1 (1994), 9–17.

[37] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. 1978. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.* 14, 1 (1978), 265–294. https://doi.org/10.1007/BF01588971

[38] David Peleg and Alejandro A Schäffer. 1989. Graph spanners. *Journal of graph theory* 13, 1 (1989), 99–116.

[39] Robert Clay Prim. 1957. Shortest connection networks and some generalizations. *The Bell System Technical Journal* 36, 6 (1957), 1389–1401.

[40] Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu. 2012. Approximate shortest distance computing: A query-dependent local landmark scheme. *IEEE Transactions on Knowledge and Data Engineering* 26, 1 (2012), 55–68.

[41] Liam Roditty, Mikkel Thorup, and Uri Zwick. 2005. Deterministic constructions of approximate distance oracles and spanners. In *International Colloquium on Automata, Languages, and Programming*. Springer, 261–272.

[42] SH Rodriguez, JF Martinez-Trinidad, and JA Carrasco-Ochoa. [n.d.]. On the selection of base prototypes for LAESA and TLAESA classifiers. In *2008 19th International Conference on Pattern Recognition*.

[43] Enrique Vidal Ruiz. 1986. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters* 4, 3 (1986), 145–157.

[44] Christian Sommer. 2014. Shortest-path queries in static networks. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 1–31.

[45] Suvrit Sra, Joel Tropp, and Inderjit Dhillon. 2004. Triangle fixing algorithms for the metric nearness problem. *Advances in Neural Information Processing Systems* 17 (2004).

[46] Larissa Stanberry, Rajesh Nandy, and Dietmar Cordes. 2003. Cluster analysis of fMRI data using dendrogram sharpening. *Human brain mapping* 20, 4 (2003), 201–219.

[47] Mikkel Thorup and Uri Zwick. 2005. Approximate distance oracles. *Journal of the ACM (JACM)* 52, 1 (2005), 1–24.

[48] Jason Tsong-Li Wang and Dennis E Shasha. 1990. Query Processing for Distance Metrics.. In *VLDB*, Vol. 90. 602–613.

[49] Bernard M Waxman. 1988. Routing of multipoint connections. *IEEE journal on selected areas in communications* 6, 9 (1988), 1617–1622.

[50] Yanling Yang, Kaizhong Zhang, Xiong Wang, Jason TL Wang, and Dennis Shasha. 1998. An approximate oracle for distance in metric spaces. In *Combinatorial Pattern Matching: 9th Annual Symposium, CPM 98 Piscataway, New Jersey, USA, July 20–22 1998 Proceedings 9*. Springer, 104–117.

[51] Changchuan Yin, Ying Chen, and Stephen S-T Yau. 2014. A measure of DNA sequence similarity by Fourier Transform with applications on hierarchical clustering. *Journal of theoretical biology* 359 (2014), 18–28.