2012 IEEE 28th International Conference on Data Engineering

Approximate Shortest Distance Computing: A Query-Dependent Local Landmark Scheme

Miao Qiao, Hong Cheng, Lijun Chang, Jeffrey Xu Yu

Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong
{mqiao, hcheng, ljchang, yu}@se.cuhk.edu.hk

***Abstract*—Shortest distance query between two nodes is a fundamental operation in large-scale networks. Most existing methods in the literature take a landmark embedding approach, which selects a set of graph nodes as landmarks and computes the shortest distances from each landmark to all nodes as an embedding. To handle a shortest distance query between two nodes, the precomputed distances from the landmarks to the query nodes are used to compute an approximate shortest distance based on the triangle inequality.**

In this paper, we analyze the factors that affect the accuracy of the distance estimation in the landmark embedding approach. In particular we find that a globally selected, query-independent landmark set plus the triangulation based distance estimation introduces a large relative error, especially for nearby query nodes. To address this issue, we propose a query-dependent local landmark scheme, which identifies a local landmark close to the specific query nodes and provides a more accurate distance estimation than the traditional global landmark approach. Specifically, a local landmark is defined as the least common ancestor of the two query nodes in the shortest path tree rooted at a global landmark. We propose efficient local landmark indexing and retrieval techniques, which are crucial to achieve low offline indexing complexity and online query complexity. Two optimization techniques on graph compression and graph online search are also proposed, with the goal to further reduce index size and improve query accuracy. Our experimental results on large-scale social networks and road networks demonstrate that the local landmark scheme reduces the shortest distance estimation error significantly when compared with global landmark embedding.

I. Introduction

As the size of graphs that emerge nowadays from various application domains is dramatically increasing, the number of nodes may reach the scale of hundreds of millions or even more. Due to the massive size, even simple graph queries become challenging tasks. One of them, the shortest distance query, has been extensively studied during the last four decades. Querying shortest paths or shortest distances between nodes in a large graph has important applications in many domains including road networks, social networks, communication networks, sensor networks, biological networks, the Internet, and so on. For example, in road networks, the goal is to find shortest routes between locations; in social networks, the goal is to find the closest social relationships such as friendship or collaboration between users; while in the Internet environment, the goal is to find the nearest server in order to improve access latency for clients. Although classical algorithms like breadth-first search (BFS), Dijkstra's algorithm [1], and A^* search algorithms [2], [3], [4] can compute the exact shortest paths in a network, the massive size of the modern information networks and the online nature of such queries make it infeasible to apply the classical algorithms online. On the other hand, it is space inefficient to precompute the shortest paths between all pairs of nodes and store them on disk, as it requires $O(n^3)$ space to store the shortest paths and $O(n^2)$ space to store the distances for a graph with n nodes.

Recently, there have been many different methods [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] for estimating the shortest distance between two nodes in a graph based on graph embedding techniques. A commonly used embedding technique is *landmark embedding*, where a set of graph nodes is selected as *landmarks* [6], [12], [14] (also called *reference nodes* [11], [15], *beacons* [8], or *tracers* [5]) and the shortest distances from a landmark to all the other nodes in a graph are precomputed. Such precomputed distances can be used online to provide an approximate distance between two graph nodes based on the triangle inequality.

In this paper, we revisit the landmark embedding approach. According to the findings in the literature [5], [12], the problem of selecting the optimal landmark set is NP-hard, by a reduction from the classical NP-hard problems such as set cover or minimum K-center [16]. As a result, the existing studies use random selection or graph measure based heuristics such as degree, betweenness centrality, closeness centrality, coverage, etc. Despite the various heuristics which try to optimize landmark selection, all the existing methods follow the triangulation based distance estimation, which estimates the shortest distance between a pair of query nodes as the sum of their distances to a landmark. As the landmark selection step is query independent, the landmark set provides a single global view for all possible queries which could be diameter apart or close by. Thus it is hard to achieve uniformly good performance on all queries. As a consequence, the landmark embedding approach may introduce a large relative error, especially when the landmark set is distant from both nodes in a query but the two nodes themselves are nearby. For example, in a US road network with 24 million nodes and 58 million edges, the landmark embedding technique leads to a relative error of 68 in the worst case, i.e., the estimated distance is 69 times of the actual shortest distance.

This observation motivates us to find a query-dependent "local landmark" which is close to both query nodes for a more accurate distance estimation. In contrast, the original

1084-4627/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDE.2012.53

landmarks are called "global landmarks". In this paper, we propose a query-dependent local landmark scheme, which identifies a local landmark specific to a pair of query nodes. Then the distance between the two query nodes is estimated as the sum of their shortest distances to the local landmark, which is much closer than the global one. The query-dependent local landmark scheme is expected to reduce the distance estimation error in principle, compared with the traditional global landmark embedding.

Challenges. The key challenges of the query-dependent local landmark scheme lie in the following aspects. First, efficient local landmark indexing and retrieval techniques are needed. We cannot afford expensive online computation to find a query-specific local landmark, as it would significantly increase the query processing time. Second, the shortest distance from a query node to a local landmark needs to be efficiently computed. This distance should not be computed from scratch at the query time. These two factors are crucial to achieve efficient online query processing. Third, the embedding index should be compact. The estimation accuracy improvement and the query processing efficiency should not be achieved at the expense of an increase in the offline indexing complexity. For example, precomputing and storing local landmarks for all possible query pairs has $O(n^2)$ space complexity, which is too much for large-scale networks.

Bearing these goals in mind, we propose a *shortest path tree* based local landmark scheme, where the shortest path trees rooted at global landmarks help to select the query-dependent local landmarks between two query nodes. Specifically, a local landmark is defined as the *least common ancestor* (LCA) of the two query nodes in a shortest path tree rooted at a global landmark. Our analysis and experimental results show that our proposed local landmark scheme can significantly improve the distance estimation accuracy of the traditional global landmark embedding and the state-of-the-art embedding techniques, without increasing the embedding or query complexity.

Our main contributions are summarized as follows.

- In the traditional landmark embedding, we find that the query-independent global landmark selection introduces a large relative error, especially for nearby query nodes which are distant from the global landmarks. In light of this, we propose a *query-dependent local landmark scheme* which finds a local landmark close to both query nodes to improve the distance estimation accuracy. The local landmark scheme proves to be a robust embedding solution that substantially reduces the dependency of query performance on the global landmark selection strategy.
- Given a query node pair, the proposed local landmark scheme finds a local landmark, which is defined as the *least common ancestor* of the two query nodes in the shortest path tree rooted at a global landmark. An $O(1)$ time algorithm for finding LCAs online is introduced. We show that the shortest path tree based local landmark scheme can significantly improve the distance estimation accuracy, without increasing the offline embedding or the online query complexity. Optimization techniques including graph compression and local search are also proposed with the goal to further reduce index size and improve query accuracy.
- We performed extensive experiments on large-scale social networks and road networks. Experimental results show that our shortest path tree based local landmark scheme significantly reduces the average relative error to the scale of $0-10^{-3}$, which is orders of magnitude better than the global landmark approach.

II. Preliminary Concepts

Consider an edge weighted graph $G = (V, E, w)$, where V is a set of vertices, E is a set of edges, and $w : E \mapsto \mathbb{R}^+$ is a weighting function mapping an edge $(u, v) \in E$ to a positive real number $w(u, v) > 0$, which measures the length of (u, v). We denote $n = |V|$ and $m = |E|$. For a pair of vertices $a, b \in V$, we use $\delta(a, b)$ to denote the shortest distance between a and b, and $SP(a, b) = (a, v_1, v_2, \ldots, v_{l-1}, b)$ to denote the corresponding shortest path, where $\{a, v_1, \ldots, v_{l-1}, b\} \subseteq V$ and $\{(a, v_1), (v_1, v_2), \ldots, (v_{l-1}, b)\} \subseteq E$. In this paper, we limit our study to undirected graphs.

A. Landmark Embedding

Given a pair of query nodes (a, b), to efficiently estimate an approximate shortest distance between a and b, a commonly adopted approach is *landmark embedding*. Consider a set of nodes $S = \{l_1, \ldots, l_k\} \subseteq V$ which are called *landmarks*. For each $l_i \in S$, we compute the shortest distances to all nodes in V. Then for every node $v \in V$, we can use a k-dimensional vector

$$\overrightarrow{D}(v) = \langle \delta(l_1, v), \delta(l_2, v), \ldots, \delta(l_k, v) \rangle$$

to represent its distances to the k landmarks. This is called landmark embedding, which can be used to compute an approximate shortest distance between nodes a and b based on the triangle inequality as

$$\widetilde{\delta}(a, b) = \min_{l_i \in S} \{\delta(l_i, a) + \delta(l_i, b)\} \quad (1)$$

This general embedding approach has been widely used in most existing methods in the literature.

B. Landmark Selection

In the landmark embedding approach, a key question is how to select the landmark set S from V, as the landmark selection can heavily influence the estimation accuracy of shortest distance queries. However, selecting the optimal set of landmarks has been proven to be very hard. [12] formulates the landmark selection problem based on betweenness centrality and shows that the set cover problem is polynomial-time reducible to it, thus proves the latter is NP-hard. Another study [5] formulates the landmark selection problem as the minimum K-center problem [16] which is NP-complete. Due to the hardness of the landmark selection problem, previous studies (e.g., [5], [12], [15]) proposed various heuristics, including random selection, degree based, centrality based, and coverage

based selection heuristics. But the performance of different heuristics heavily depends on the graph properties, e.g., degree distribution, diameter, etc. There is no heuristic that excels in all kinds of graphs.

C. Factors on Embedding Performance

Here we briefly discuss the factors that affect the performance of landmark embedding.

A globally selected query-independent landmark set: Most existing methods select a single set of global landmarks which are independent of queries. Such a query-independent landmark set provides a single global view for all possible queries which could be diameter apart or close by, thus it cannot achieve uniformly good performance on all queries. In particular, the landmark set can only provide a very rough distance estimation for a query, especially when it is distant from both query nodes, and the two query nodes are close by, as illustrated in Figure 1.

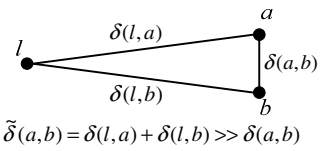


Fig. 1. Distance Estimation with a Landmark

The number of landmarks k: In general, increasing the number of landmarks k will improve the performance of landmark embedding. An extreme case is $k = |V|$ which leads to zero estimation error. This actually corresponds to computing all pair shortest paths as an embedding. As a side effect, increasing k will cause an increase of the query processing time and the index size, as the query complexity is $O(k)$ and the index space complexity is $O(kn)$. Thus, increasing k is not an efficient or scalable solution to improve the embedding performance.

III. Local Landmark Scheme

In this section, we aim to improve the performance of landmark embedding by proposing a *local landmark scheme*, a novel framework for estimating the shortest distance with the aid of a small number of query-dependent local landmarks. To distinguish, we call the landmarks in S as *global landmarks*. Specifically, given a query node pair (a, b) and a global landmark set S, the idea is to identify a query-specific local landmark, which is closer to the shortest path $SP(a, b)$ than any global landmark, thus providing a more accurate distance estimation. Let us see an example before giving our problem statement.

Example 1: Figure 2 shows an example graph with the global landmark set $S = \{l_1, l_2, l_3\}$. In this graph, a solid line between two nodes represents an edge, while a dashed line between two nodes represents a path which can have zero or more intermediate connecting nodes.

For a pair of query nodes (a, b), the path in bold (a, e, f, g, b) is the shortest path between a and b. In addition, the shortest paths from the global landmark l_1 to a and b are $(l_1, \ldots, c, e, a)$ and $(l_1, \ldots, c, d, g, b)$, respectively. Based on l_1, the estimated shortest distance between a and b is

$$\widetilde{\delta}(a, b) = \delta(l_1, a) + \delta(l_1, b) \tag{2}$$

But if we have the shortest distances $\delta(c, a)$ and $\delta(c, b)$, we can have a more accurate distance estimation based on c than that based on l_1:

$$\widetilde{\delta}^L(a, b) = \delta(c, a) + \delta(c, b) \tag{3}$$

as compared with $\widetilde{\delta}^L(a, b)$, $\widetilde{\delta}(a, b)$ in Eq.(2) counts the shortest distance $\delta(l_1, c)$ twice extra, i.e., $\widetilde{\delta}(a, b) = \widetilde{\delta}^L(a, b) + 2\delta(l_1, c)$.

As opposed to the concept of global landmark, we call node c a local landmark with respect to the query nodes a, b. Similarly, nodes d and h are local landmarks, as opposed to the global landmarks l_2 and l_3.

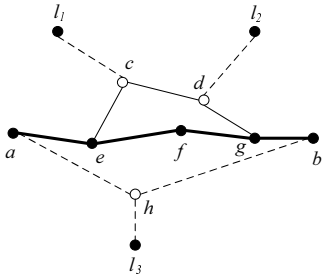


Fig. 2. Local Landmarks

Problem Statement: In this paper, we study the problem of finding query-dependent local landmarks for shortest distance queries in large graphs. Given an arbitrary query node pair (a, b) and a global landmark set S, our goal is to identify a query-specific local landmark which is closer to the true shortest path $SP(a, b)$ than any global landmark in a graph G. The approximate shortest distance between a and b is computed as the sum of their distances to the local landmark.

A. Query-Dependent Local Landmarks

We first define an abstract query-dependent local landmark function, which returns a local landmark given a query and a set of global landmarks S.

Definition 1 (Query-Dependent Local Landmark): Given a global landmark set S and a query (a, b), a query-dependent local landmark function is

$$L_{ab}(S) : V^k \mapsto V$$

which maps S to a vertex in V called a local landmark.

The query-dependent local landmark is a function of the query (a, b) and the global landmark set S. The dependency on the global landmark set S is necessary, because the set S and the embedded distances contain useful information of the graph, which can help in identifying a high-quality local landmark close to the query nodes.

With the local landmark function, we can estimate a shortest distance of a query (a, b) as

$$\widetilde{\delta}^L(a, b) = \delta(L_{ab}(S), a) + \delta(L_{ab}(S), b) \tag{4}$$

To use the local landmark to process a distance query practically as specified in Eq.(4), we need to design the query-dependent local landmark function such that $\forall a, b \in V$:

1) The query-dependent local landmark $L_{ab}(S)$ can be efficiently retrieved to answer a query online;
2) The shortest distances $\delta(L_{ab}(S), a)$, $\delta(L_{ab}(S), b)$ can be efficiently computed to answer a query online; and
3) The embedding index for the local landmark scheme should be compact.

IV. Shortest Path Tree Based Local Landmark

In the landmark embedding approach, for each global landmark $l \in S$, the shortest distances from l to all vertices in V are precomputed for the embedding purpose. To preserve more delicate information, we can further consider the *shortest path tree* (SPT) rooted at each global landmark $l \in S$.

Definition 2 (Shortest Path Tree): Given a graph $G = (V, E, w)$, the shortest path tree rooted at a vertex $r \in V$ is a spanning tree of G, such that the path from the root r to each node $v \in V$ is a shortest path between r and v, and the path length is the shortest distance between r and v.

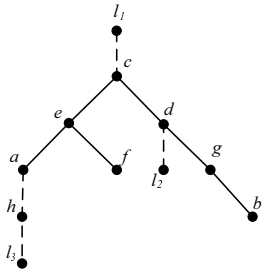


Fig. 3. Shortest Path Tree Rooted at l_1

Figure 3 shows a shortest path tree rooted at the global landmark l_1 according to the graph in Figure 2. Note that the l_1-rooted SPT that can be derived from Figure 2 may not be unique. Here we show one example. A shortest path tree not only contains the shortest distance information from the tree root, it also preserves the more delicate structure information on how the two query nodes are connected. Based on the tree structure, we can identify a node, e.g., c, which is closer to nodes a and b than l_1. In addition, the tree has some nice properties which lead to efficient algorithms for online query processing. Based on this intuition, we propose a shortest path tree based local landmark function.

A. SPT Based Local Landmark Function

In Example 1, we find that the distance estimation based on node c, i.e., $\widetilde{\delta}^L(a, b) = \delta(c, a) + \delta(c, b)$ is a tighter upper bound than that based on l_1. If we look at the shortest path tree rooted at l_1 in Figure 3, it is not hard to find that c is the *least common ancestor* (LCA) of a and b.

Definition 3 (Least Common Ancestor): Let T be a rooted tree with n nodes. The least common ancestor of two nodes u and v in T is the node furthest from the root that is an ancestor of both u and v. $LCA_T(u, v)$ returns the least common ancestor of u and v.

In light of this, we propose an SPT based local landmark function, which returns the LCA of the query nodes (a, b) in the shortest path tree rooted at each landmark $l \in S$.

Definition 4 (SPT Based Local Landmark Function): Given a global landmark set S and a query (a, b), the SPT based local landmark function is defined as:

$$L_{ab}(S) = \arg \min_{r \in \{LCA_{T_l}(a,b) | l \in S\}} \{\delta(r, a) + \delta(r, b)\}$$

where $LCA_{T_l}(a, b)$ denotes the least common ancestor of a and b in the shortest path tree T_l rooted at $l \in S$.

We can show that the distance estimation with the SPT based local landmarks is more accurate, or at least the same accurate as that with the global landmark set.

Theorem 1: Given a global landmark set S, $\forall a, b \in V$, we have

$$\delta(a, b) \leq \widetilde{\delta}^L(a, b) \leq \widetilde{\delta}(a, b)$$

.

Proof: First, $\delta(a, b) \leq \widetilde{\delta}^L(a, b)$ holds according to the triangle inequality.

Next, $\forall l \in S$, $r = LCA_{T_l}(a, b)$, we have

$$\delta(l, a) + \delta(l, b) = \delta(r, a) + \delta(r, b) + 2\delta(r, l)$$

As $\delta(r, l) \geq 0$, we have

$$\delta(r, a) + \delta(r, b) \leq \delta(l, a) + \delta(l, b)$$

Consequently,

$$\begin{aligned} \widetilde{\delta}^L(a, b) &= \min_{l \in S} \{\delta(LCA_{T_l}(a, b), a) + \delta(LCA_{T_l}(a, b), b)\} \\ &\leq \min_{l \in S} \{\delta(l, a) + \delta(l, b)\} \\ &= \widetilde{\delta}(a, b) \end{aligned}$$

■

To efficiently calculate $\widetilde{\delta}^L(a, b)$, $a \neq b \in V$, we have:

$$\begin{aligned} \widetilde{\delta}^L(a, b) &= \min_{l \in S} \{\delta(LCA_{T_l}(a, b), a) + \delta(LCA_{T_l}(a, b), b)\} \\ &= \min_{l \in S} \{\delta(l, a) + \delta(l, b) - 2\delta(l, LCA_{T_l}(a, b))\} \end{aligned} \tag{5}$$

where $LCA_{T_l}(a, b)$ is the least common ancestor of a and b in the SPT rooted at $l \in S$. For each SPT rooted at a global landmark $l \in S$, we compute the LCA of a and b and estimate the shortest distance between a and b as the sum of their shortest distances to the LCA. When $LCA_{T_l}(a, b)$, $\forall l \in S$ is known, Eq.(5) can be calculated in $O(|S|)$ time. For each $l \in S$, it involves three lookup operations for the embedded distances $\delta(l, a)$, $\delta(l, b)$ and $\delta(l, LCA_{T_l}(a, b))$.

B. LCA Computation

In this subsection, we introduce the techniques in [17] for efficiently finding the LCA of two nodes in a shortest path tree in $O(1)$ time with an $O(n)$ size index. A closely related problem to LCA is *Range Minimum Query* (RMQ), the solution to which leads to the solution to LCA.

Definition 5 (Range Minimum Query): Let A be an array of length n, namely $A[1, \ldots, n]$. For indices $1 \leq i \leq j \leq n$, $RMQ_A(i, j)$ returns the index of the smallest element in the subarray $A[i, \ldots, j]$.

This linkage between the two problems is based on the following observation. The LCA of nodes a and b is the shallowest node encountered between the visits to a and to b during a depth first search of a tree T. Based on this linkage, we can reduce the LCA problem to RMQ as follows:

1) Perform a depth first search on the tree T and record the node label sequence $trace[1, \ldots, 2n-1]$ in the Euler Tour of the tree. The Euler Tour of T traverses each of the $n-1$ edges in T twice, once in each direction, during a DFS traversal. Therefore, the array $trace$ has a length $2n-1$. For example, for the shortest path tree in Figure 3, $trace = (l_1, \ldots, c, e, a, \ldots, h, \ldots, l_3, \ldots, h, \ldots, a, e, f, e, c, d, \ldots, l_2, \ldots, d, g, b, g, d, c, \ldots, l_1)$.
2) Record the level of the nodes in the tree T in an array $L[1, \ldots, 2n-1]$, where $L[i]$ is the level of the node $trace[i]$. We define the level of the root as 0, and the level of a child node increases that of its parent by 1.
3) For the n nodes in the tree T, record the time stamp of each node in $stamp[1, \ldots, n]$, where $stamp[a] = \min_i\{trace[i] = a\}$, $a \in V$. This is the time when node a is visited for the first time in a DFS traversal.

With the above transformation, without loss of generality, for two nodes a, b, let $stamp[a] < stamp[b]$, then

$$LCA_T(a, b) = trace[\arg \min_{stamp[a] \leq i \leq stamp[b]} L[i]] \quad (6)$$
$$= trace[RMQ_L(stamp[a], stamp[b])]$$

The query $RMQ_L(stamp[a], stamp[b])$ finds the index of the node with the smallest level, i.e., the shallowest node, in the subarray $L[stamp[a], stamp[b]]$. Therefore, by recording the Euler Tour of a shortest path tree in a DFS traversal, plus an RMQ query in the array L, we can compute the LCA between any two nodes in the tree. The DFS traversal and the Euler Tour need to be done only once as a preprocessing operation in $O(n)$ time. Thus the complexity of an LCA query is determined by that of an RMQ query. [17] proposed an efficient RMQ algorithm in $O(1)$ time with an index of size $O(n)$. The RMQ algorithm is presented in Appendix.

C. Error Rate Analysis

The local landmark scheme (LLS) we propose is different from the global landmark scheme (GLS) in that, LLS is query-dependent, whereas GLS is query-independent. In the literature [8], [9], [15], a query-independent global landmark scheme can be possibly bounded, because the landmark set is global and is the same for any queries, even though the bound may not be tight. Due to the nature of the query-dependence, it is difficult to obtain a bound for LLS. However, it is provable that LLS has at least the same bound as GLS if GLS is bounded. As shown in Theorem 1 as well as confirmed in our extensive experimental studies, LLS has a much lower error rate in practice.

D. Complexity Analysis

We analyze the online query complexity and the offline embedding complexity of our local landmark scheme.

Online Query Time Complexity: The online query is done based on Eq.(5), $\widetilde{\delta}^L(a, b) = \min_{l \in S}\{\delta(l, a) + \delta(l, b) - 2\delta(l, LCA_{T_l}(a, b))\}$. For each global landmark $l \in S$, there are three lookup operations to retrieve the embedded distances, each of which takes $O(1)$ time. In addition, there is one LCA query which can be answered in $O(1)$ time by the RMQ algorithm. Note that the $O(1)$ time RMQ algorithm involves a constant number of additions and divisions, a log and a floor operations, a constant number of table lookups and minimum operations. Compared with the global landmark embedding which only needs two lookups for $\delta(l, a)$ and $\delta(l, b)$, our query processing time has a larger constant factor. When considering all global landmarks in S, the online query time complexity is $O(|S|)$.

Empirically, if the depth of an SPT is small, we can trace the LCA of nodes a and b on the SPT by starting from the two nodes and following the child-parent links until the LCA is met. This strategy could be more efficient than RMQ on graphs with a small diameter.

Offline Embedding Space Complexity: The space requirement of our SPT based local landmark scheme can be partitioned into the following three parts:

1) Embedded distances. For each global landmark, we need to store the precomputed shortest distances to every node in the graph. Thus the space complexity for the embedded distances is $O(|S|n)$.
2) Shortest path trees. For each global landmark, we compute the corresponding SPT and then generate the *trace*, *L* and *stamp* arrays with a DFS traversal. A shortest path tree and the above arrays each takes $O(n)$ space. So the total space complexity is $O(|S|n)$.
3) RMQ index tables. To answer an arbitrary LCA query $LCA_T(a, b)$ in a shortest path tree, we need to perform a query $RMQ_L(stamp[a], stamp[b])$ on the tree level array $L[1, \ldots, 2n-1]$. According to the ± 1 RMQ algorithm [17] in Appendix, it builds an index table of size $O(n)$. Considering all the $|S|$ shortest path trees, the total size for the RMQ index tables is $O(|S|n)$.

Combining the above three factors, the offline embedding space complexity of our local landmark scheme is $O(|S|n)$.

Offline Embedding Time Complexity: Given a global landmark set S, the time complexity to compute the single-source shortest paths from a landmark by Dijkstra's algorithm [1] is $O(m + n \log n)$. It can be simplified to $O(n \log n)$ when the graph G is sparse. As a by-product, Dijkstra's algorithm also produces the shortest path trees. In addition, we have to build

the RMQ index table in $O(n)$ time for each global landmark in S. Thus the total embedding time complexity is $O(|S|n \log n)$.

When compared with global landmark embedding, it is not hard to verify that our local landmark scheme has the same online query complexity and offline embedding complexity, although our complexities have slightly larger constant factors.

V. Optimization Techniques

In this section, we propose two additional techniques, *graph compression* and *local search* to further optimize the performance of our local landmark scheme. Graph compression aims to reduce the embedding index size by compressing the graph nodes, and local search performs limited scope online search to improve the distance estimation accuracy.

A. Index Reduction with Graph Compression

As we have shown, the embedding index takes $O(|S|n)$ space, which is a linear function of the graph node number n. Thus we can effectively reduce the embedding index size if the graph nodes can be compressed. Towards this goal, we propose graph compression techniques which reduce some simple local graph structures with low-degree nodes to a representative node. Our compression techniques are lossless, thus do not sacrifice the distance query accuracy.

1) Graph Compression and Index Construction: We first define two types of special graph nodes, i.e., *tree nodes* and *chain nodes*.

Definition 6 (Graph Incident Tree): A tree $T = (V_T, E_T, r)$ with the root r is a graph incident tree on a graph $G = (V, E)$ if (1) $V_T \subset V$, $E_T \subset E$; and (2) for any path $p(u, v)$ between $u \in V_T$ and $v \in V - V_T$, p must go through the tree root r. A graph incident tree is maximal if it is not contained in another graph incident tree. The nodes $V_T - \{r\}$ are called tree nodes and the root r is the entry node of all tree nodes in T.

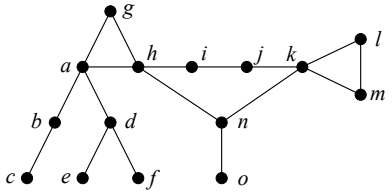


Fig. 4. Graph Compression Example

For example, in Figure 4, the tree with nodes a, b, c, d, e, f is a maximal graph incident tree, where the root a is the *entry node* and nodes b, c, d, e, f are the *tree nodes*. A graph incident tree can be simply discovered by recursively removing graph nodes with degree 1 until the entry node is met (with degree $>$ 1). As the entry node is the only access point for a tree node to connect to the rest of the graph, it is sufficient to keep the entry node as a representative. The graph incident tree is thus compressed to the entry node by removing all the tree nodes. In addition, the distance from each tree node to the entry node is saved in an array. After we remove all tree nodes, we next identify the chain nodes.

Definition 7 (Chain Nodes): Given a graph $G = (V, E)$, a chain node $v \in V$ is a non tree node with $degree(v) = 2$.

If we trace through the two edges incident on a chain node respectively, the two nodes which are first encountered through the chain with a degree greater than 2 are called *end nodes*. The two end nodes may be identical when a cycle exists. For example in Figure 4, nodes i and j are *chain nodes* with *end nodes* h and k. We will remove the chain nodes and the incident edges, and then connect the two end nodes with a new edge, whose length is equal to the length of the chain. The distances from a chain node to both end nodes are saved in an array.

After we remove the tree nodes and chain nodes and their incident edges from a graph $G = (V, E)$, we obtain a compressed graph $G' = (V', E')$. Then the index structures including the embedded distances, shortest path trees and RMQ index tables are constructed on top of G', instead of G. As $|V'| < |V|$, the index size can be effectively reduced.

One point worth noting is that, as a graph incident tree is compressed to a single entry node, the tree structure is lost. When given two query nodes from the same tree, their LCA cannot be identified from the compressed graph G'. For example, for e, f in Figure 4, their LCA d cannot be identified from the compressed graph, as it has been removed as a tree node. In order to handle such queries, we select one global landmark $l \in V$ and build the embedding index including the shortest path tree and RMQ table on the *original graph*; and select the other global landmarks from V' and build the index on the *compressed graph*. For any two tree nodes on the same graph incident tree, e.g., e, f, their LCA is the same in all shortest path trees rooted at any global landmarks. Thus it is sufficient to build the full index on the original graph for one global landmark only. The space complexity is $O(n + (|S| - 1)n')$, which is smaller than $O(|S|n)$ on the original graph.

2) Query Processing on Compressed Graph: Given a query (a, b), if $a, b \in V'$, the local landmark based approximate shortest distance $\widetilde{\delta}^L(a, b)$ can be estimated by Eq.(5) in the same way as in the original graph; otherwise, if at least one of a, b is a tree node or chain node and not in V', then

$$\widetilde{\delta}^L(a, b) = \min_{r_a \in map(a), r_b \in map(b)} \{\delta(a, r_a) + \widetilde{\delta}^L(r_a, r_b) + \delta(b, r_b)\} \quad (7)$$

where $map(a)$ contains the representative nodes for a, defined as follows: (1) if a is a tree node, $map(a)$ contains the corresponding entry node; (2) if a is a chain node, $map(a)$ contains the two end nodes; and (3) if $a \in V'$, $map(a)$ is a itself. The intermediate query $\widetilde{\delta}^L(r_a, r_b)$ can be answered by Eq.(5), as $r_a, r_b \in V'$, according to the definition of $map()$.

There are two special cases to be handled separately.

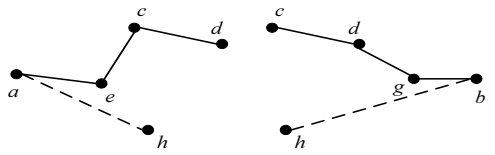
1) If a, b are tree nodes from the *same* graph incident tree, the query $\widetilde{\delta}^L(a, b)$ can be answered with the local landmark from the index on the original graph. For example, for query (e, f) in Figure 4, $\widetilde{\delta}^L(e, f) = \delta(e, d) + \delta(f, d)$, where d is the LCA of e and f.

2) If a, b are two chain nodes with the same end nodes, then $\widetilde{\delta}^L(a,b) = \min\{d, |\delta(a,r) - \delta(b,r)|\}$, where d is estimated by Eq.(7), and r is either one of the two end nodes. For example, for query (i,j) in Figure 4, there are two paths $p_1 = (i,j)$ and $p_2 = (i,h,n,k,j)$ between i and j. The distance $\widetilde{\delta}^L(i,j)$ is estimated by the shorter one among p_1 and p_2.

Our graph compression is lossless. Thus in all the above cases, the estimated distance based on the compressed graph will be the same as that based on the original graph.

B. Improving the Accuracy by Local Search

In this subsection, we propose an online *local search* technique which performs a limited scope local search on the graph and may find a shortcut with a smaller distance than that based on local landmark embedding. Given a query (a,b), for each global landmark $l \in S$, we can find the least common ancestor $LCA_{T_l}(a,b)$ in the shortest path tree T_l rooted at l. The shortest path between a query node and a local landmark $LCA_{T_l}(a,b)$ can also be obtained from the corresponding shortest path tree T_l. If we trace the shortest paths from a to all the LCAs (similarly from b to all the LCAs), we can form two *partial shortest path trees* rooted at a and b respectively, e.g., T_a and T_b in Figure 5 following Example 1. A leaf node in such trees must be an LCA; while it is also possible an LCA is an intermediate node, if it lies on the shortest path from a query node to another LCA, e.g., the intermediate node c in T_a in Figure 5 (a).

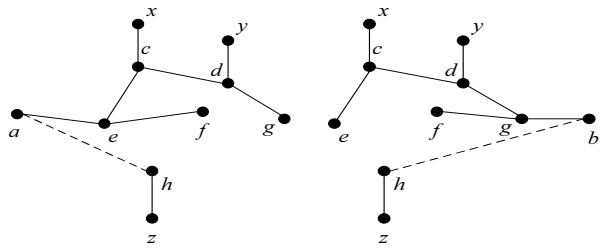


(a) T_a (b) T_b

Fig. 5. Partial Shortest Path Trees

The local search expands a partial shortest path tree T by a width of c, i.e., for each node in T, its neighbors within c hops in the graph are included in the expanded tree T^c. For the two expanded trees T_a^c and T_b^c rooted at the query nodes, the common nodes of T_a^c and T_b^c act as bridges to connect the two query nodes. We will find a path connecting the two query nodes through a bridge with the smallest distance. If the distance is smaller than the estimation $\widetilde{\delta}^L(a,b)$ by the local landmark scheme, we will report this local search distance as a more accurate estimation for the query (a,b). Figure 6 shows the 1-hop expanded trees T_a^1 and T_b^1, where the 1-hop neighbors of every tree node in T_a and T_b are included. Based on the expanded trees there are three paths connecting a and b, i.e., (a,e,c,d,g,b), (a,e,f,g,b), and $(a,\ldots,h,\ldots,b)$. As (a,e,f,g,b) has the shortest distance between a and b, we return the distance as the answer.

Algorithm 1 shows the pseudocode of the local search. Lines 2-3 build two partial shortest path trees rooted at a and b respectively to all the local landmarks. Lines 4-5 expand the two trees to include the neighbors within c hops for each tree node. The notation $dist_{T_a^c}(a,r)$ in line 8 represents the path length from a to r in the expanded tree T_a^c. Note that the local search with tree expansion is done at query time. Compared with the $O(1)$ time query processing based on local landmarks, the local search is more expensive for online query processing.



(a) T_a^1 (b) T_b^1

Fig. 6. 1-Hop Expanded Trees T_a^1 and T_b^1

Algorithm 1 Local Search

Input: A query (a,b) and the expansion width c.
Output: The shortest distance of a path.

1: $LCA \leftarrow \{LCA_{T_l}(a,b) | l \in S\}$
2: $T_a \leftarrow partial\_SPT(a, LCA)$
3: $T_b \leftarrow partial\_SPT(b, LCA)$
4: $T_a^c \leftarrow Tree\_Expansion(T_a, c)$
5: $T_b^c \leftarrow Tree\_Expansion(T_b, c)$
6: $dist \leftarrow \infty$
7: **for** $r \in T_a^c \cap T_b^c$ **do**
8: **if** $dist_{T_a^c}(a,r) + dist_{T_b^c}(b,r) < dist$ **then**
9: $dist \leftarrow dist_{T_a^c}(a,r) + dist_{T_b^c}(b,r)$
10: **end if**
11: **end for**
12: **return** $dist$

VI. Experiments

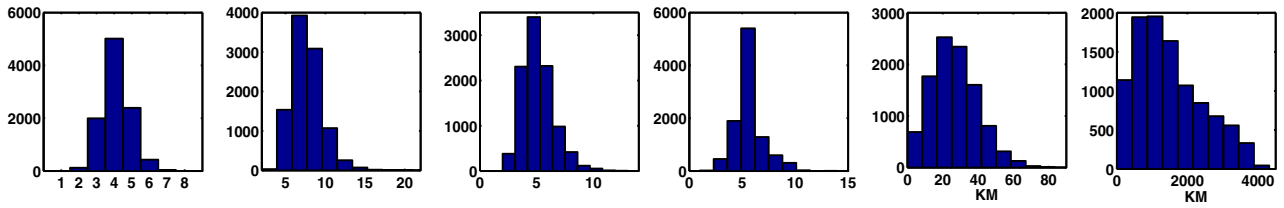
We compare our query-dependent local landmark scheme with global landmark embedding and present extensive experimental results in terms of accuracy, query efficiency and index size on six large networks. All algorithms were implemented in C++ and tested on a Windows server using one 2.67 GHz CPU and 128 GB memory.

A. Dataset Description

Slashdot[1]. Slashdot is a technology-news website introduced in 2002, where users can tag each other via friend and foe links. The data set was crawled in 2008. A node in Slashdot is a user in the community and an edge (u,v) means user u tags user v as his friend.

Google Webgraph[2]. A webgraph released for the Google Programming Contest in 2002. A node is a web page and an edge is a hyperlink between pages.

[1]http://snap.stanford.edu/data/soc-Slashdot0902.html
[2]http://snap.stanford.edu/data/web-Google.html



(a) Slashdot (b) Google (c) Youtube (d) Flickr (e) NYRN (f) USARN

Fig. 7. Shortest Distance Histogram

TABLE I
NETWORK STATISTICS

Dataset	$\|V\|$	$\|E\|$	$\|V'\|$	$\|E'\|$	$\overline{\delta}$
Slashdot	77,360	905,468	36,012	752,478	4.1146
Google	875,713	5,105,039	449,341	4,621,002	7.4607
Youtube	1,157,827	4,945,382	313,479	4,082,046	5.3317
Flickr	1,846,198	22,613,981	493,525	18,470,294	5.5439
NYRN	264,346	733,846	164,843	532,264	27km
USARN	23,947,347	58,333,344	7,911,536	24,882,476	1522km

Youtube. A YouTube video-sharing graph of over 1 million users and 5 million links crawled in 2007 [18].
Flickr. A Flickr photo-sharing graph of 1.8 million users and 22 million links crawled in 2007 [18].
NYRN[3]. The New York City road network is an undirected planar graph with $264,346$ nodes and $733,846$ edges. A node represents an intersection or a road endpoint while an edge weight represents the length of a road segment.
USARN[3]. The USA road network is an undirected planar graph with $23,947,347$ nodes and $58,333,344$ edges.

Table I lists the network statistics. $|V|$ and $|E|$ represent the node and edge numbers in the original graph, while $|V'|$ and $|E'|$ represent the numbers in the compressed graph. As we can see, our proposed graph compression technique effectively reduces the node number by $38\% - 73\%$. Our embedding index is constructed on the compressed graph. We also sample $10,000$ node pairs in each network and show the average shortest distance $\overline{\delta}$. Figure 7 plots the histogram of the shortest distance distribution over the $10,000$ node pairs in each network.

B. Comparison Methods and Metrics

The embedding methods we compare include:
Global Landmark Scheme (GLS). The traditional landmark embedding technique which estimates distance by Eq.(1).
Local Landmark Scheme (LLS). Our local landmark scheme which estimates distance according to the technique in Section V-A2.
Local Search (LS). Algorithm 1 which further improves the estimation accuracy with a local search of a width c. We set $c = 1$ in our experiments.

For the global landmark selection, we use random selection and closeness centrality based selection [12]. As it is expensive to compute the exact centrality scores for all nodes in a graph, following the strategy in [12], we compute approximate centrality scores based on a set of random seeds. Then the k nodes with the smallest shortest distances to the random seed set are selected as landmarks. We use two landmark set sizes $k = 20$ and $k = 50$ in our experiments.
Evaluation Metrics We use the relative error

$$\frac{|\widetilde{\delta}(s,t) - \delta(s,t)|}{\delta(s,t)}$$

to evaluate the quality of an estimated distance for a query (s,t). As it is expensive to exhaustively test all node pairs in a large network, we randomly sample $10,000$ node pairs in each graph as queries and compute the average relative error on the sample set. In addition, we test the query processing time and the embedding index size.

C. Comparison With Global Landmark Scheme

1) Average Relative Error: Table II shows the average relative error (AvgErr) of GLS, LLS and LS with different global landmark sets selected by Random and Centrality on the six networks.

Firstly, LLS reduces the AvgErr of GLS by a large margin in all cases. Under Random landmark selection strategy, the AvgErr of LLS is one order of magnitude smaller than that of GLS on most graphs; while under Centrality, LLS reduces the AvgErr by 40% compared with GLS on average. Furthermore, in most cases the AvgErr of LLS with $k = 20$ landmarks is even lower than that of GLS with $k = 50$ landmarks, and at the same time, LLS ($k = 20$) has a smaller embedding index size than GLS ($k = 50$), see Table IV. This result demonstrates that selecting more global landmarks for GLS (e.g., $k = 50$) and using more index space do not necessarily achieve a better estimation accuracy than LLS ($k = 20$). Thus simply selecting more landmarks for GLS may not be an effective solution, as the main bottleneck of GLS is caused by the query-independent landmark embedding.

Secondly, although the AvgErr of both GLS and LLS monotonically decreases with k, the error reduction of LLS is much more substantial than GLS. Take Random for example, the AvgErr of GLS reduces by 28.1%, 6.3%, 28.3%, 11.1%, 34.9%, 43.6% respectively on the six networks when k increases from 20 to 50, while that of LLS reduces by 48.9%, 55.8%, 38.6%, 45.5%, 58.1%, 69.3%, much more significant than GLS. This fact shows that adding a new global landmark will bring a larger benefit to LLS than GLS.

[3]http://www.dis.uniroma1.it/∼challenge9/download.shtml

TABLE II
AVERAGE RELATIVE ERROR

		$k = 20$						$k = 50$					
		SlashD	Google	Youtube	Flickr	NYRN	USARN	SlashD	Google	Youtube	Flickr	NYRN	USARN
Random	GLS	0.6309	0.5072	0.6346	0.5131	0.1825	0.1121	0.4535	0.4750	0.4549	0.4559	0.1188	0.0632
	LLS	0.1423	0.0321	0.0637	0.0814	0.0246	0.0786	0.0727	0.0142	0.0391	0.0444	0.0103	0.0241
	LS	0.0000	0.0046	0.0009	0.0001	0.0071	0.0090	0.0000	0.0022	0.0003	0.0001	0.0042	0.0030
Centrality	GLS	0.1520	0.0426	0.0595	0.0567	0.6458	1.5599	0.1385	0.0245	0.0461	0.0524	0.6133	0.7422
	LLS	0.1043	0.0290	0.0489	0.0503	0.1536	0.4708	0.0663	0.0140	0.0334	0.0284	0.1533	0.4505
	LS	0.0001	0.0074	0.0010	0.0003	0.1479	0.4703	0.0000	0.0037	0.0005	0.0000	0.1455	0.4483

TABLE III
ONLINE QUERY TIME IN MILLISECONDS

	$k = 20$						$k = 50$					
	SlashD	Google	Youtube	Flickr	NYRN	USARN	SlashD	Google	Youtube	Flickr	NYRN	USARN
GLS	0.002	0.005	0.008	0.009	0.006	0.020	0.005	0.016	0.024	0.027	0.014	0.058
LLS	0.006	0.021	0.015	0.014	0.036	0.067	0.018	0.054	0.032	0.033	0.091	0.196
LS	0.158	2.729	2.818	4.735	0.681	58.289	0.527	3.492	4.178	6.817	1.585	98.221

TABLE IV
INDEX SIZE IN MB

	$k = 20$						$k = 50$					
	SlashD	Google	Youtube	Flickr	NYRN	USARN	SlashD	Google	Youtube	Flickr	NYRN	USARN
GLS	6.2	57.9	90.7	124.7	21.2	1915.8	15.5	144.8	226.8	311.6	52.9	4789.5
LLS	10.4	122.7	103.2	156.1	85.3	4424.6	23.3	284.5	216.1	333.8	203.9	9948.3
LS	16.4	159.7	135.9	303.9	89.6	4623.6	29.4	321.4	248.7	481.6	208.2	10147.3

Thirdly, LLS performs very well on all graphs even with randomly selected global landmarks (i.e., without exploiting graph properties), whereas GLS heavily depends on the landmark selection heuristics. GLS performs reasonably well with Centrality on the social networks, but its AvgErr increases by one order of magnitude with Random in most cases. In contrast, the AvgErr of LLS with Random on the social networks is very close to that with Centrality. Thus LLS provides a robust embedding solution that substantially reduces the dependency of query performance on the global landmark selection strategy.

Finally, LS achieves the best performance in all cases. Its AvgErr is between 0 and the scale of 10^{-3} in most cases. In particular, in social networks the average distance is usually very small, according to the famous rule of "six degrees of separation". Thus by a local search with 1-hop expansion, the expanded trees rooted at both query nodes are very likely to intersect, which helps to find a very short path, or even the shortest path, between the query nodes.

One point worth noting is that, the state-of-the-art techniques for computing shortest paths and shortest distances on road networks have achieved controlled error rate and low complexities, with the aid of coordinates [19] and heuristics such as geometric container [20]. As LLS and LS are designed for general graphs, the performance improvement by our methods is more significant on social network than on road networks.

2) Online Query Time: Table III shows the query time in milliseconds of different methods. According to our analysis in Section IV-D, the online query complexity is $O(1)$ for both GLS and LLS. Even in the largest graph USARN with 24 million nodes, it only costs 0.196 milliseconds for LLS to process one query when $k = 50$. In most cases, LLS is $2 - 4$ times slower than GLS, which is a very small factor.

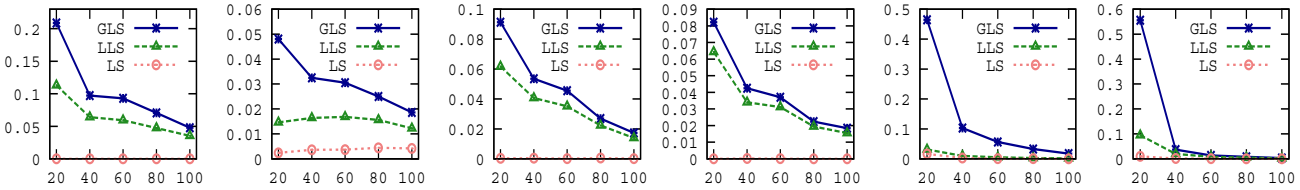
As LS performs online tree expansion and search in query processing, the query time largely depends on the network size. For example, it costs 0.158 milliseconds to process one query in Slashdot, but 58 milliseconds in USARN when $k = 20$, as the node number of USARN is about 310 times larger than that of Slashdot.

3) Index Size: Table IV shows the index size in MB. The index size of LLS is about 2 times that of GLS, as LLS needs to store extra information including shortest path trees and RMQ index tables. LS uses a little extra space compared with LLS, as LS needs to store the original graph in memory for edge expansion and local search. Nevertheless, we can see that our LLS and LS methods use moderate index sizes even for very large networks, e.g., an index of 10 GB (when $k = 50$) on USARN with 24 million nodes. In addition, as Table I shows, our graph compression technique reduces the graph node number by $38\% - 73\%$, which has effectively reduced the index size of our LLS and LS methods. In terms of the index construction time of LLS, it takes less than 1 minute in most cases, while the longest one takes 354.7 seconds on USARN when $k = 50$.

D. Comparison with Other Embedding Methods

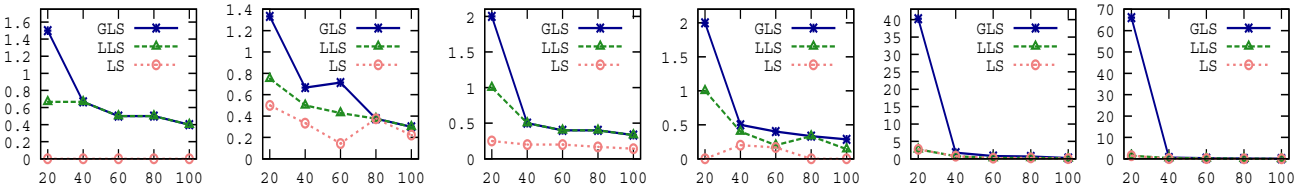
Two closely related studies [13], [14] that have been proposed recently following [9] use $\log n$ sets of global landmarks, each of a different size, to improve the embedding performance. As the TreeSketch algorithm proposed in [14] significantly outperforms [13], we compare our LS method with TreeSketch [14]. We also compare with a global landmark embedding method 2RNE [11] which organizes landmarks in multiple levels.

For the LS method, we use the local landmark embedding



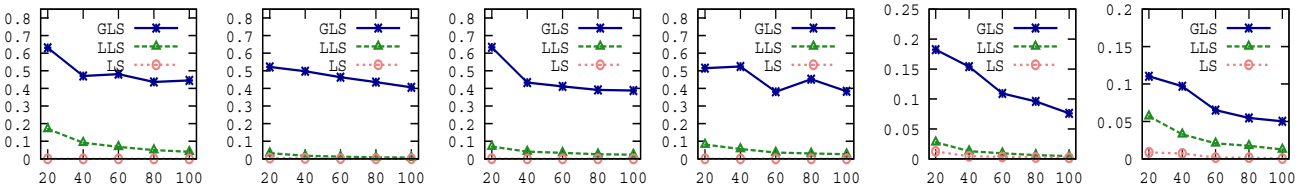
(a) Slashdot (b) Google (c) Youtube (d) Flickr (e) NYRN (f) USARN

Fig. 8. Average Relative Error on Queries with Shortest Distance in Different Ranges



(a) Slashdot (b) Google (c) Youtube (d) Flickr (e) NYRN (f) USARN

Fig. 9. Maximum Relative Error on Queries with Shortest Distance in Different Ranges



(a) Slashdot (b) Google (c) Youtube (d) Flickr (e) NYRN (f) USARN

Fig. 10. Average Relative Error with Different Global Landmark Number k

TABLE V
COMPARISON WITH OTHER METHODS

Dataset	Algorithm	AvgErr	Query Time(ms)	Index Size(MB)
SlashD	2RNE	0.8345	0.001	6.2
	TreeSketch	0.0011	0.176	37.4
	LS	0.0000	0.158	16.4
Google	2RNE	0.5750	0.001	57.9
	TreeSketch	0.0048	3.549	383.7
	LS	0.0046	2.729	159.7
Youtube	2RNE	0.7138	0.001	90.7
	TreeSketch	0.0005	5.317	587.7
	LS	0.0009	2.818	135.9
Flickr	2RNE	0.6233	0.001	124.7
	TreeSketch	0.0001	7.333	959.6
	LS	0.0001	4.735	303.9
NYRN	2RNE	0.4748	0.001	21.2
	TreeSketch	0.0156	1.074	120.5
	LS	0.0071	0.681	89.6
USARN	2RNE	0.4240	0.002	1915.8
	TreeSketch	0.0379	104.769	14555.5
	LS	0.0090	58.289	4623.6

index with $k = 20$ random global landmarks and set the local search width $c = 1$. TreeSketch is a disk-based algorithm. For a fair comparison, we provide a memory-based implementation of TreeSketch. In the memory-based TreeSketch index, for a graph node v and a landmark set S, we need only to store the predecessor of v in the shortest path tree of S, instead of the whole path from v to S as in the disk-based implementation, whose goal is to minimize I/O. This will substantially reduce the index size when compared with the disk version, while it is sufficient to process queries in memory without increasing the query time, thus a more fair comparison. We set the TreeSketch parameter $K = 2$, the same as in [14].

As Table V shows, LS outperforms TreeSketch by a large margin systematically. The average error of LS is much lower than that of TreeSketch in almost all graphs. The query time of LS is 67% that of TreeSketch on average, while the index size is 40% that of TreeSketch on average.

On the other hand, we can see 2RNE is an instantiation of the global landmark approach. Thus its performance with $k = 20$ landmarks is very close to that of GLS reported by us. The query time of 2RNE is within 0.002 milliseconds on all graphs and its index size is 40% that of LS. However, its average error is significantly worse than that of LS.

E. Distance Sensitive Relative Error

We evaluate the performance of GLS, LLS and LS on queries in different shortest distance ranges. For each network, we sort the $10,000$ sample queries in the increasing order of their actual shortest distances and find the 20th, 40th, 60th, 80th and 100th percentiles of the shortest distance. Based on this, we partition the $10,000$ queries into 5 intervals, each

containing 20 percent of the queries. We evaluate the AvgErr in Figure 8 and the maximum relative error (MaxErr) in Figure 9 for queries whose shortest distances fall into the five internals respectively. For each network we adopt the best landmark selection heuristic, i.e., Centrality for social networks and Random for road networks. We set $k = 50$.

We observe that LLS outperforms GLS on both AvgErr and MaxErr in all distance ranges on all networks. The improvement is most significant for queries whose shortest distances are within the 20th percentile. This demonstrates that LLS can provide very accurate estimation for nearby queries while GLS cannot. In particular, we observe that on the two road networks, the improvement of LLS over GLS within the 20th percentile is about 10 times or more on both AvgErr and MaxErr, which is very substantial. This is because the road networks have large diameters. For a close by query, if both query nodes are quite distant from the global landmark set, GLS will provide a very inaccurate distance estimation. In contrast, social networks usually have a fairly small diameter, which guarantees that the global landmark set will not be too far from the query nodes. Therefore, on the social networks, the improvement by LLS on both AvgErr and MaxErr is around 2 times within the 20th percentile.

The performance of LS remains very stable in all distance ranges. The AvgErr of LS is zero or close to zero on most networks. We also observe that, on dense networks with large average degrees, e.g., Slashdot and Flickr, local search with neighbor expansion is particularly effective in finding the (nearly) shortest paths. The AvgErr and MaxErr of LS on Slashdot are zero in all distance ranges.

F. Parameter Sensitivity Tests

In this experiment, we evaluate the average error of GLS, LLS and LS when we vary the number of global landmarks k from 20 to 100. The k global landmarks are selected randomly. Figure 10 shows the results on the six networks. The results consistently show that the AvgErr of the three methods decreases when the global landmark number k increases. LS achieves the smallest AvgErr, ranging from 0 to the scale of 10^{-3}, followed by LLS and then GLS. The AvgErr of LLS is below 0.1 for most cases. For GLS, the AvgErr is up to 0.63 when $k = 20$, and it decreases when k increases.

VII. Related Work

Graph embedding techniques have been widely used to estimate the distance between two nodes in a graph in many applications including road networks [7], [11], social networks and web graphs [10], [12], [13], [14], [15] as well as the Internet [5], [6]. [7] utilizes Linial, London and Robinovich (LLR) embedding technique to estimate the distance between two nodes. Kriegel et al. [11] propose a hierarchical reference node embedding approach which organizes reference nodes in multiple levels for a better scalability. Potamias et al. [12] formulate the reference node selection problem to selecting nodes with high betweenness centrality. [5] proposes an architecture, called IDMaps, which measures and disseminates distance information on the global Internet. [8] defines a notion of slack – a certain fraction of all distances that may be arbitrarily distorted as a performance guarantee based on randomly selected reference nodes. [9] and its follow-up studies [13], [14] provide a $(2k - 1)$-approximate distance estimation with $O(kn^{1+1/k})$ memory for any integer $k \geq 1$. To summarize, the major differences between the above methods lie in the following aspects: (1) landmark selection – some [8], [9], [10], [13], [14] select landmarks randomly, while others [5], [6], [11], [12] use heuristics; (2) landmark organization – some methods organize landmarks in multiple levels [9], [11], [13], [14], while other methods use a flat landmark embedding; and (3) an error bound or not – [8], [9], [13], [14], [15] analyze the error bound of the estimated distances, while most of the other methods have no error bounds or guarantees of the estimated distances.

There have been a lot of studies on computing shortest paths and processing k-nearest neighbor queries in spatial networks. Papadias et al. [21] use the Euclidean distance as a lower bound to prune the search space and guide the network expansion for refinement. Kolahdouzan and Shahabi [22] propose to use first order Voronoi diagram to answer KNN queries in spatial networks. Hu et al. [23] propose an index, called distance signature, which associates approximate distances from one object to all the other objects in the network, for distance computation and query processing. Samet et al. [24] build a shortest path quadtree to support k-nearest neighbor queries in spatial networks. For a spatial network of dimension d, [19] can retrieve an ε-approximation distance estimation in $O(\log n)$ time using an index termed path-distance oracle of size $O(n \cdot \max(s^d, \frac{1}{\epsilon}^d))$. [25] proposes TEDI, an indexing and query processing scheme for the shortest path query based on tree decomposition.

VIII. Conclusions

In this paper, we propose a novel shortest path tree based local landmark scheme, which finds a node close to the query nodes as a query-specific local landmark for a triangulation based shortest distance estimation. Specifically, a local landmark is defined as the LCA of the query nodes in a shortest path tree rooted at a global landmark. Efficient algorithms for indexing and retrieving LCAs are introduced, which achieve low offline indexing complexity and online query complexity. As the query-dependent local landmark is much closer to the query nodes, this strategy significantly reduces the distance estimation error, compared with the global landmark embedding approach. Some optimization techniques are also proposed to further improve the performance. Extensive experimental results on large-scale social networks and road networks demonstrate the effectiveness and efficiency of the proposed local landmark scheme.

Acknowledgment

The work described in this paper was partially supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No.: CUHK

419008, 419109, 411310 and 411211). We thank Andrey Gubichev for providing us the source code of TreeSketch [14].

REFERENCES

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics SSC4*, vol. 4, no. 2, pp. 100–107, 1968.

[3] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A* search meets graph theory," in *SODA*, 2005, pp. 156–165.

[4] A. V. Goldberg, H. Kaplan, and R. F. Werneck, "Reach for A*: Efficient point-to-point shortest path algorithms," in *Workshop on Algorithm Engineering and Experiments*, 2006, pp. 129–143.

[5] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A global internet host distance estimation service," *IEEE/ACM Trans. Networking*, vol. 9, no. 5, pp. 525–540, 2001.

[6] T. S. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *INFOCOM*, 2001, pp. 170–179.

[7] C. Shahabi, M. Kolahdouzan, and M. Sharifzadeh, "A road network embedding technique for k-nearest neighbor search in moving object databases," in *GIS*, 2002, pp. 94–100.

[8] J. Kleinberg, A. Slivkins, and T. Wexler, "Triangulation and embedding using small sets of beacons," in *FOCS*, 2004, pp. 444–453.

[9] M. Thorup and U. Zwick, "Approximate distance oracles," *Journal of the ACM*, vol. 52, no. 1, pp. 1–24, 2005.

[10] M. J. Rattigan, M. Maier, and D. Jensen, "Using structure indices for efficient approximation of network properties," in *KDD*, 2006.

[11] H.-P. Kriegel, P. Kröger, M. Renz, and T. Schmidt, "Hierarchical graph embedding for efficient query processing in very large traffic networks," in *SSDBM*, 2008, pp. 150–167.

[12] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis, "Fast shortest path distance estimation in large networks," in *CIKM*, 2009, pp. 867–876.

[13] A. D. Sarma, S. Gollapudi, M. Najork, and R. Panigrahy, "A sketch-based distance oracle for web-scale graphs," in *WSDM*, 2010.

[14] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum, "Fast and accurate estimation of shortest paths in large graphs," in *CIKM*, 2010.

[15] M. Qiao, H. Cheng, and J. X. Yu, "Querying shortest path distance with bounded errors in large graphs," in *SSDBM*, 2011.

[16] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[17] M. A. Bender and M. Farach-Colton, "The LCA problem revisited," in *LATIN 2000: Theoretical Informatics*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, vol. 1776, pp. 88–94.

[18] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *IMC*, 2007, pp. 29–42.

[19] J. Sankaranarayanan, H. Samet, and H. Alborzi, "Path oracles for spatial networks," *PVLDB*, vol. 2, no. 1, 2009, pp. 1210–1221.

[20] D. Wagner, T. Willhalm, and C. D. Zaroliagis, "Geometric containers for efficient shortest-path computation," *ACM Journal of Experimental Algorithmics*, vol. 10, 2005.

[21] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network database," in *VLDB*, 2003, pp. 802–813.

[22] M. Kolahdouzan and C. Shahabi, "Voronoi-based k nearest neighbor search for spatial network databases," in *VLDB*, 2004, pp. 840–851.

[23] H. Hu, D. L. Lee, and V. C. S. Lee, "Distance indexing on road networks," in *VLDB*, 2006, pp. 894–905.

[24] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *SIGMOD*, 2008, pp. 43–54.

[25] F. Wei, "TEDI: Efficient shortest path query answering on graphs," in *SIGMOD*, 2010, pp. 99–110.

APPENDIX

A. An Algorithm for General RMQ

The Range Minimum Query (RMQ) problem is, given an array A of length n, for indices $1 \leq i \leq j \leq n$, $RMQ_A(i,j)$ returns the index of the smallest element in the subarray $A[i,\ldots,j]$. If we build a table M to store the answers, where the cell $M[i][j]$ records the index of the smallest element in $A[i,\ldots,j]$, $\forall 1 \leq i \leq j \leq n$, we can answer an RMQ query in $O(1)$ time with an index of size $O(n^2)$. Another algorithm can reduce the index size to $O(n \log n)$ as follows.

The algorithm precomputes the RMQ query for each subarray whose length is a power of two. Specifically, for $1 \leq i \leq n$ and $1 \leq j \leq \log n$, we find the minimum element in the subarray starting at i and having length 2^j and store the index in the cell $M[i][j]$, i.e., $M[i][j] = \arg\min_{i \leq k \leq i+2^j-1} A[k]$. The table M has a size of $O(n \log n)$ and a dynamic programming algorithm can fill in M in $O(n \log n)$ time.

To answer a query $RMQ_A(i,j)$, $1 \leq i \leq j \leq n$, let $k = \lfloor \log(j-i) \rfloor$. Then $RMQ_A(i,j)$ can be computed by the index of the minimum of $A[M[i][k]]$ and $A[M[j-2^k+1][k]]$, which has been precomputed and stored in the table M.

B. An Improved Algorithm for ± 1 RMQ

For the RMQ problem there is an improved algorithm with an index size of $O(n)$ for an array A with the ± 1 restriction, which means adjacent elements in A differ by $+1$ or -1.

The algorithm first partitions the array A into blocks of size $\frac{\log n}{2}$. Define an array $A'[1,\ldots,2n/\log n]$ where $A'[i]$ records the minimum element in the i-th block of A. When we run the general RMQ algorithm on the array A', we generate a table M with a size of $O(n)$ in $O(n)$ time. With M, we can find the block which contains the minimum element with an RMQ query on A' in $O(1)$ time.

To answer a query $RMQ_A(i,j)$, $1 \leq i \leq j \leq n$, we first check whether i and j are in the same block or not. If i and j are in the same block, we need to process one block to answer the RMQ query. If i and j are in different blocks, the query $RMQ_A(i,j)$ can be processed in three steps as follows.

1) Compute the min element from i to the end of its block;
2) Compute the min element between i's and j's blocks;
3) Compute the min element from j's block to j.

The second step can be computed in $O(1)$ time with an RMQ query on the array A', which records the minimum elements in each block of A. Therefore, the problem is reduced to finding the minimum element inside a block. In the following we just focus on the inside block query. For a block of size $\frac{\log n}{2}$, we can create a table with a size of $(\frac{\log n}{2})^2 = O(\log^2 n)$ to index the answers for all possible inside block queries. Then the key question is how many tables we need to build to handle the inside block queries? [17] shows that for an array with the special ± 1 property, there are only $O(\sqrt{n})$ kinds of blocks. For identical blocks, the preprocessing can be shared. Therefore, we only need to create $O(\sqrt{n})$ tables each with a size of $O(\log^2 n)$, leading to a total size of $O(\sqrt{n} \log^2 n)$.

To summarize, for the inside block RMQ queries and the RMQ queries on A' with the general RMQ algorithm, the total index size is $O(n)$ and the query time is $O(1)$. Note that the ± 1 RMQ algorithm is applicable to our LCA problem, as the level array $L[1,\ldots,2n-1]$ defined in Section IV-B satisfies the ± 1 property. As a result, an LCA query can be answered by the ± 1 RMQ algorithm in $O(1)$ time with an index size of $O(n)$ constructed in $O(n)$ time.