Three-Hop Distance Estimation in Social Graphs

Pascal Welke
University of Bonn[1]
Bonn, Germany
welke@uni-bonn.de

Alexander Markowetz
Markowetz.de
Bonn, Germany
alexander@markowetz.de

Torsten Suel
New York University
New York, USA
torsten.suel@nyu.edu

Maria Christoforaki
Yelp Inc.[2]
San Francisco, USA
mchristoforaki@gmail.com

***Abstract*—In this paper, we study a 3-hop approach to distance estimation that uses two intermediate landmarks, where each landmark only stores distances to vertices in its local neighborhood and to the other landmarks. We show how to suitably represent and compress the distance data stored for each landmark, for the 2-hop and 3-hop case. Overall, we find that 3-hop methods achieve modest but promising improvement in some cases, while being comparable or slightly worse than 2-hop methods in others. Furthermore, our light compression schemes improve the practical applicability of both the 2-hop and 3-hop methods.**

***Keywords*-shortest paths; distance estimation; landmarks**

I. Introduction

Graphs constitute the central form of knowledge representation in a wide variety of domains; important examples include social networks, road networks, web graphs, and molecular interaction graphs. One important primitive encountered in many graph mining and search applications involves pairwise distances: given two vertices, determine the length of a shortest connecting path. In many applications, traversing the graph at runtime to find a shortest path is too slow, while precomputing distances between all pairs requires too much memory.

In many applications, it suffices to have a good estimate of the distance, since this distance is often just a rough proxy for the similarity or relatedness of two nodes (e.g., in expert search in social networks, or search in web graphs) and may be used as one of many features in ranking. For this reason, many researchers have studied how to estimate pairwise distances, which allows for much faster and more space efficient solutions than the exact case. Most of the approaches are heuristic in nature, and are engineered to work well for specific graphs and query loads. In particular, social graphs tend to have certain properties, such as highly skewed degree distributions that follow a power law, high clustering coefficients, and small diameters. Many of the queries issued in social applications tend to be for pairs of nodes that have smaller than average distance. Thus, methods developed for such graphs may not work well on other types of graphs or query loads, and often do not have good worst-case bounds.

Probably the most commonly studied and used approach for distance estimation is based on the idea of *global landmarks*, which are designated vertices that store their distance from all other nodes in an undirected graph. Given a landmark w, the distance $d(u, v)$ between any nodes u and v can be upper-bounded by $d(u, w)+d(v, w)$ using the triangle inequality; the best approximation is then obtained by taking the minimum over all landmarks.

The precision of this estimate depends on the number of landmarks as well as their selection strategy; the best landmarks are those that are on, or close to, shortest paths between many pairs u and v. If u and v are often close to each other, then it is necessary to have landmarks in many neighborhoods of the graph. Unfortunately, this *two-hop* approach does not scale well as the graph size increases. This is because for larger graphs, we need more and more landmarks to get good estimates, resulting in total space requirements that are super-linear in the number of vertices. Conversely, if we are restricted to space linear in the number of vertices, then this limits us to a fixed number of landmarks independent of graph size, leading to increasingly bad estimates.

In this paper, we study a *three-hop* approach that obtains estimates via two intermediate landmark nodes, and that can support a larger number of landmarks in linear space. Our methodology is based on a technique described by Bast et al. [1] in the context of road networks, where they apply the above method in a multi-layered fashion to obtain *exact shortest paths*, not just distance estimates. In contrast, here we engineer this method to suit the properties of social graphs.

The basic idea is very simple: we choose a set of landmarks, and then precompute and store for each landmark its distance from all other landmarks. In addition, for each node we store its distance to a limited number of carefully selected close-by landmarks. Given two nodes u and v, we can then upper-bound $d(u, v)$ by $d(u, w_1) + d(w_1, w_2) + d(w_2, v)$ where w_1 and w_2 are any landmarks for which $d(u, w_1)$ and $d(v, w_2)$ have been stored. The best estimate is obtained by taking the smallest bound over all possible choices of w_1 and w_2.

This approach raises a number of questions that need to be explored. First, we need suitable schemes to select three-hop landmarks, and to choose for which close-by landmarks each node should store distances. Second, we need to show that three-hop methods actually do better. While three-hop methods can store $O(\sqrt{n})$ landmarks in contrast to $O(1)$ landmarks for the two-hop method, given space linear in the number of vertices n, this does not directly imply better estimates. Three-hop methods might incur larger estimation errors than two-hop methods that use a single application of the triangle inequality, potentially canceling out any benefits due to having

[1] Work done while visiting NYU.
[2] Work done while a graduate student at NYU.

more landmarks. Finally, given the different structure of two-hop and three-hop methods, we need to decide how to best store and encode the distance data. Simply assuming that each distance costs a fixed number of bits (say one 32-bit int) would not result in a fair comparison of the approaches: In practical settings the absolute space consumption of the data structures versus estimation quality counts and each method should use the available space in an efficient manner. This includes compressing the data structures in a way that allows fast answering of arbitrary distance queries to allow for more stored information in a given amount of space.

In the remainder of this paper, we address these questions. We evaluate several strategies for selecting three-hop landmarks, including centrality and degree. We also study how to select which landmarks a vertex should store distances to. While our baseline simply selects the k closest landmarks for each vertex, we show how to select a better set of landmarks such that no landmark obscures (i.e., is on the shortest path to) another landmark. We then study how to reduce the space consumption of the methods, by exploring compression methods for landmark distance tables. The resulting methods benefit both two-hop and three-hop schemes, and enable a fair comparison that gives each method the same amount of space. Overall, our main technical contributions are as follows: (1) We describe a three-hop approach for distance estimation in graphs that provides improvements in precision in many scenarios. (2) We reevaluate existing methods for landmark selection in the case of three-hop methods. (3) We show how to store and compress the main data structures used in two-hop and three-hop schemes. (4) We run extensive experiments comparing two-hop and three-hop methods in terms of estimation accuracy, efficiency, and scalability on a number of data sets and query load, showing both strengths and weaknesses of the approach.

Section II discusses related work, and Section III formally states the problem and describes the two-hop approach. Then Section IV introduces our three-hop approach and Section V discusses compression. The experimental evaluation is provided in Section VI, and finally Section VII concludes.

II. Related Work

There are three classes of previous work that are closely related to our work, exact distance computation, distance estimation in graphs, and graph compression.

A. Exact Distance Computation

The state of the art in point-to-point shortest path (PPSP) computation are *ALT algorithms* [2], [3], [4], [5], which combine a bidirectional version of Dijkstra's algorithm with an A^* search guided by lower bounds. In most common scenarios, this succeeds in reducing the search space for the shortest path to a small subgraph, thus significantly improving efficiency over unidirectional or bidirectional versions of Dijkstra's algorithm. Most of the literature on ALT algorithms [2], [3], [4] has focused on road networks, which tend to have properties such as near planarity, low degree, and the presence of hierarchy, that benefit these algorithms. However, they often do not perform well on social networks, web graphs, or co-authorship networks, which do not exhibit these properties.

The most closely related previous work [1] computes exact distances in road networks using an idea similar to ours: It defines a locality relation for pairs of vertices and uses online shortest path computations for local query pairs. "Long distance" queries are answered using a three-hop data structure that must retrieve the exact distance for any such pair. They give two implementations of this idea that are tailored to road networks and extend it by recursively applying the scheme (resulting in a k-hop technique for $k \geq 2$). In contrast, our method answers *all* distance queries *approximately* using a three-hop data structure, due to the fact that almost all distance queries would be "local" in a social graph, resulting in too many online path computations for the approach in [1].

Other recent work [6], [7] also achieves faster exact distance computation, but at the cost of often very significant storage overheads that depends on the structure of the graph.

B. Distance Estimation

One basic approach uses embedding methods [8], [9], [10], [11], [12], [13], where graphs are mapped to low-dimensional vector spaces with efficiently computable distance functions that approximate distances in the original graph within guaranteed bounds. Another line of work approximates the graph by sparse subgraphs called spanner graphs [14], [15], [16]. Both approaches can provide provable bounds on space requirements and the quality of the approximation. However, most methods are difficult to implement, and no empirical evaluation is given. Our methods here do not provide theoretical guarantees, but they are simple to implement and work well in practice in terms of their approximation error and space consumption on common classes of graphs.

Landmark-Based Methods: The most widely used heuristic approach is based on global landmarks [17], [18], [19], [20], [21], [22], which are subsets of the nodes for which we store distances to all other, or in some cases some selected other, nodes. Given a query, we can then apply the triangle inequality on these precomputed distances to get upper and lower bounds on the exact distance. While most work has focused on upper bounds, there has also been some study of lower bounds [2], [3], [4], [19], [21]. However, lower bounds obtained from landmarks tend to have a much larger error than upper bounds [19]. Landmark-based methods can be divided into two main categories.

Global landmark approaches, which can also be seen as embedding techniques, store distances between each landmark and all other nodes. Although theoretic guarantees on the quality of *most* obtained distance estimates for randomly selected landmarks are possible [22], much accuracy can be gained in practice by selecting landmarks more carefully based on centrality measures [19]. This significantly reduces estimation errors for a given amount of space, but may result in redundancies between different landmarks. Hence [18] suggest an alternative greedy algorithm that considers a graph's

"coverage" by landmarks, and an algorithm that first partitions a graph and then chooses good landmarks for each partition. Recent work shows how to obtain machine-learned estimates based on landmarks [23].

Local landmark approaches store for each landmark only the distances to a subset of close-by nodes. A multilevel sampling approach to select landmarks, where each level only stores distances to nodes that are within a certain distance from the landmarks is presented in [16]. Any graph can be preprocessed in $O(k \cdot |E| \cdot |V|^{1/k})$ expected time to produce landmark distance data of size $O(k \cdot |V|^{1+(1/k)})$, for any constant number of levels $k \geq 1$. Using these sketches, any point-to-point query can be answered in $O(k)$ time to provide a $(2k-1)$-approximation of the correct distance. The work in [16] can be simplified [21], allowing for an easier implementation while retaining theoretical guarantees. In practice the obtained estimation errors are usually much better than the theoretical guarantees [21], but significantly larger than the best global landmark methods using the same space [23].

Finally, a closely related problem is how to compute actual approximate shortest paths, not just estimate their lengths, using landmarks [20], [24]. There is also some loosely related but different work in the database community on query processing for reachability queries in graphs [25], [26], [27], [28].

Our work can be seen as a combination of ideas from the global and local landmark approaches. Our landmarks store distances to all other landmarks, but only to those non-landmark nodes that are close by. While almost all global and local landmark approaches use two hops, we use three hops, following the work in [1].

C. Graph Representation and Compression

We focus on how to best represent and compress distance data in landmark approaches, which started when we realized that a fair comparison to other methods requires an exploration of this issue. To the best of our knowledge, ours is the first work considering how to compress landmark-related data. There is, however, a significant amount of related work on graph compression in the data compression and web search literature, including work on compressing web graphs [29], [30], [31], [32], [33], [34], [35] and social graphs [35], [36], [37], [38]. The proposed techniques exploit vertex similarities, edge locality, and many other structural properties of the graphs for better compression. One very common idea is to store the adjacency list of one vertex as a delta with respect to that of another close-by or similar vertex, and then apply further compression using, e.g., variable length or run-length encodings. We apply similar ideas in our approach.

III. Preliminaries

Throughout this paper, G denotes an undirected graph where V is the set of vertices and E the set of edges. Unless stated otherwise, we assume graphs to be simple, connected, unweighted, and free of self-loops. A path P connecting two nodes v_i and v_j is an acyclic sequence of edges $[(v_i, v_u), (v_u, v_w), (v_w, v_x), \ldots, (v_z, v_j)]$. Its length $l(P) := |P|$ is the number of edges in the sequence. The distance $d(v_i, v_j)$ is the minimum length of any path connecting v_i and v_j. In a weighted graph, every edge (v_i, v_j) has a weight $c(v_i, v_j) \in \mathbb{R}^+$, and the length of a path P is defined as the sum of edge weights.

A. Problem Definition

In this work we address the following problem: Given an undirected graph G, compute a data structure of size $O(|V| + |E|)$ that allows fast approximate answers to distance queries for arbitrary pairs of vertices $v_i, v_j \in V$. In particular, we want space to be linear with a small constant, i.e., not significantly larger than the original graph, and the time for estimations to be constant or at least significantly faster than linear.

B. Two-Hop Distance Estimation using Landmarks

One solution to the above problem are global landmarks, which we shall call *two-hop* landmarks. This approach selects a set of nodes $L \subset V$ as landmarks and stores for each node in V its distance to every node in L, resulting in a storage cost of $O(|V| \cdot |L|)$. To determine these distances during preprocessing, we execute a BFS from each landmark, requiring time $O(|L| \cdot (|V| + |E|))$. We will describe below how to select the landmarks.

By the triangle inequality, the distance between any nodes v_i and v_j is upper-bounded by $\min_{l \in L} d(v_i, l) + d(l, v_j)$, and also lower-bounded by $\max_{l \in L} |d(v_i, l) - d(l, v_j)|$. We focus here on upper bounds, which tend to be much closer to the real answer than lower bounds [19]. Given a query, we can thus return an upper bound in time $O(|L|)$, by performing two lookups for each landmark.

C. Landmark Selection

Both the two- and three-hop approach require a set of global landmarks L. Optimal selection of landmarks for two-hop methods is known to be NP-hard, and various greedy heuristics have been proposed [19]. These algorithms usually impose an ordering on the set of nodes, and then select the top-$|L|$ nodes as landmarks. We consider the following methods: **Random Order**; **Degree**, which is believed to indicate an improved chance to be on many shortest paths; **Closeness**, measuring the inverse of the distances from the particular node to all other node; **Betweenness**, the number of shortest paths passing through a vertex; and **Greedy Coverage**, which successively selects landmarks that connect to the largest fraction of the remaining graph and then removes them and their neighbors [18]. For the two-hop approach, Betweenness clearly outperforms all other methods, whereas Random Order is very weak [19]. However, Betweenness as well as Closeness are expensive to compute, a drawback that can be remedied by sampling. Section VI experimentally evaluates their performance for three-hop landmark selection.

IV. The Three-Hop Approach

We now introduce our three-hop approach to distance estimation. Recall that the two-hop approach stores distances from

every node to *every* landmark. However, many of these stored distances never contribute by being part of the minimal upper bound of any distance estimate. Our three-hop approach tries to remove these unnecessary stored distances. This is of course also the motivation for the local two-hop approach in [21], [16], but as shown in [23] this does not come close to the best global schemes. Our approach stores distances between every pair of landmarks, while every non-landmark node only stores distances to a small number of close-by landmarks. This decreases space consumption, while trying to preserve accuracy.

To illustrate the idea, we start with a simplified version. First, we choose a set of landmarks L. For each vertex v_i we store the identifier of the nearest landmark l_i, and the distance $d(v_i, l_i)$. In addition, we maintain distances between any two landmarks in a $|L| \times |L|$ matrix. At query time, we upper-bound $d(v_i, v_j)$ by $d(v_i, l_i) + d(l_i, l_j) + d(l_j, v_j)$.

The tightness of the bound depends on the position of landmarks l_i and l_j; the closer they are to a shortest path, the smaller the error. If both landmarks reside on the same shortest path, then we obtain the exact distance. Of course, even if l_i lies on or close to a shortest path to v_j, it may be far away from any shortest path to some other vertex v'_j. We can hence improve this method by choosing more landmarks for each node. Thus, for each node v_i, we pick a small set $v_i.L$ of at most k landmarks from among all landmarks. As we will show, even $k = 5$ improves significantly over $k = 1$.

Our first approach, called **Next**, selects the k nearest landmarks for each vertex. To efficiently compute these landmarks, we initiate a BFS from each landmark, and maintain a priority queue of the k nearest landmarks at each vertex. While this solution performs reasonably well, there are situations where it falls short.

Given a vertex v_i, we denote a landmark l as *obscured*, iff there exists a shortest path from v_i to l containing another landmark. Consider $v_i, v_j \in V(G)$ and $l_1, l_2, l_3 \in L$. Suppose we associate l_1 and l_2 with v_i and l_2 is obscured by l_1. Then we get $d(v_i, l_2) + d(l_2, l_3) + d(l_3, v_j) = d(v_i, l_1) + d(l_1, l_2) + d(l_2, l_3) + d(l_3, v_j) \geq d(v_i, l_1) + d(l_1, l_3) + d(l_3, v_j)$ for all $v_j \in V$ and $l_3 \in v_j.L$. This implies that storing l_2 at v_i is a waste of space. Instead, we should add another landmark that is further away but not obscured, or store less than k landmarks at v_i, if there are no more unobscured landmarks. Our second association algorithm, **RedBlue**, thus assigns the closest $\leq k$ unobscured landmarks to each vertex. It can be implemented by an extended BFS that maintains two queues (one red for obscured vertices, one blue for unobscured) at no additional preprocessing cost. A similar algorithm has been described in [1].

Formally, the overall algorithm now works as follows: First, we select a set of landmarks $L \subseteq V$. Then, for every vertex v_i, we select a set $v_i.L$ of $\leq k$ landmarks from L using Next or RedBlue, and store their distances from v_i. We also store distances between all landmarks in L in a matrix. Given a query (v_i, v_j), we obtain $\leq k^2$ upper bounds $d = d(v_i, l_i) + d(l_i, l_j) + d(l_j, v_j)$, one for each $(l_i, l_j) \in v_i.L \times v_j.L$ and select the best one.

The approach analysis is as follows: Landmark selection time depends of course on the methodology. Simple approaches, such as **Degree**, run in $O(|V| + |E|)$ time, while more complicated approaches take much more time but can usually be engineered to run fast enough in practice. Distances between landmarks in G are computed in $O(|L| \cdot (|V| + |E|))$ by running BFS from each landmark. Selection of the sets $v_i.L$ for each v_i can be done during this computation, with an additional factor in the worst case of $\log k$ if using a minimum heap at each node, and a constant factor if using linear time selection instead. (We implemented the heap version as k is small and the worst case unlikely.) At each node, we store at most k landmarks and distances, requiring $O(|L|^2 + |V| \cdot k)$ space. For constant k and $|L| = O(\sqrt{|V|})$, this becomes $O(|V|)$. Query time is $O(k^2)$ assuming constant time access to the distance values.

The three-hop approach can thus asymptotically afford to keep significantly more landmarks than the two-hop approach, though we need to see how this affects precision. A fair comparison of the two requires experimentation that varies $|L|$, k, and the algorithms for landmark selection and for assignment of landmarks to nodes (in the case of three-hop). Experiments also need to make sure to suitably represent, and if needed compress, the different structures, instead of just fixing the total number of distances stored.

V. Compressing Landmark Data Structures

Although similar, the two-hop and the three-hop approach store different kinds of data. Hence, a fair practical comparison of the two methods should assign both methods the same amount of memory. We now describe lightly compressed data structures for both methods to address this issue.

A. Two-Hop Methods

For the two-hop approach we need to represent and compress a $|V| \times |L|$ array of integers. Each query accesses the two complete rows of the involved vertices. Hence, we can employ compression techniques that allow us to decompress complete rows quickly. A naïve approach would employ a 32-bit integer for each cell. However, even in very large graphs, distances usually do not exceed a very modest value (due to the small world phenomenon). Let maxdist denote the maximum distance value in the matrix. A baseline (BL) approach thus employs a fixed-size encoding of $\lceil \log_2 \text{maxdist} \rceil$ bits per entry.

We can do better with two additional observations. First, even if maxdist is, say, 15, most entries in the matrix are significantly smaller. Thus, we can use variable-length encoding schemes to save additional space. There are many such compression schemes [39]; we use Rice coding and choose its parameter globally for the entire graph. We call this scheme single row compression (SRC).

Space can be reduced further using neighbor-list compression (NLC), an idea previously used in the context of web graph compression [31], [32]. So far, we have compressed each row of the array individually. In NLC, we represent a

row by storing the difference to a neighboring node. Consider two neighboring vertices v and w. For any landmark $l \in L$, the corresponding distances to v and w can differ by at most one, i.e., $d(v,l) - d(w,l) \in \{-1, 0, 1\}$. We can thus represent a row by (i) a pointer to another row, and (ii) a list of L values $\in \{-1, 0, 1\}$, indicating the differences to that row. These values are best represented using a simple Huffman code, with 0 stored as 0, -1 as 10, and $+1$ as 11.

Of course, this requires some lists to be stored by themselves, without NLC. In general, for each row, we need to (i) decide whether to express it in absolute terms, or relative to a neighbor using NLC, and (ii) if so, choose the right neighbor. We have to ensure that any reference chains remain acyclic and of reasonable length. This applies equally to two-hop and three-hop methods, and we describe details below.

B. Three-Hop Methods

We first discuss how to store the symmetric matrix of all distances between landmarks. We thus maintain only the upper triangle as an array of $\binom{L}{2}$ entries. This allows accessing any landmark to landmark distance in constant time.

As in the two-hop case, distances in this matrix are quite small. We thus resort to a fixed-length encoding of suitable size and do not further compress this array. There are two reasons. First, the query algorithm does not access this array in a row-wise fashion, but almost at random. Second, this array is usually smaller than the array storing distances between ordinary nodes and landmarks, but is more frequently accesses; thus, speed is more important than compressed size.

Distances Between Nodes and Landmarks: For each vertex v, we need to store distances to at most k landmarks, plus the information identifying these landmarks. Thus, for each vertex v, the corresponding landmarks may be different, and we switch from a matrix representation to lists of (i,d)-pairs where $i \in \{0 \dots |L|-1\}$ is the ID of a landmark, and d is the distance from it. These entries could be compressed individually, or by expressing records of one vertex relatively to those of a neighbor since neighbors often share many landmarks. However, this requires somewhat different encoding schemes.

In a first step, we replace the naïve implementation with a tight encoding of fixed size. Note that the ranges of the distance values and landmark-IDs are quite different. The former is rather small ($\leq diameter$), while the latter is larger (up to $\sqrt{|V|}$). Thus, in a baseline approach (BL), we could store landmark IDs in $\lceil \log_2(|L|) \rceil$ bits and distances in $\lceil \log_2(maxdist) \rceil$ bits.

In the next step, single row compression (SRC) further compacts individual entries. As in two-hop above, distance values are encoded using Rice coding with a globally chosen parameter. In this case, the benefits of Rice encoding are even more pronounced as the k landmarks are chosen to be close by. In contrast, the larger landmark IDs do not benefit as much from this step. However, if we sort entries by landmark IDs, then we can encode the gaps between landmark ids using variable length codes. At query time, we reconstruct landmark IDs by summing up the gaps.

The list of records for neighboring nodes can differ in two ways. The nodes could be associated with different landmarks; however, the algorithms in Section IV emphasize close-by landmarks and thus neighboring nodes are frequently associated with the same landmarks. Second, even if landmarks are the same, distances could differ by at most 1, as in the two-hop case. In general, one can expect to identify at least one neighbor whose information is very similar (almost the same landmarks and distances).

As in the two-hop case, we can again express records in terms of the differences to a neighboring node. in this case, each record can differ from its counterpart at the neighbor in one of four ways: (i) the distance at v_i and v_j is equal, (ii) it deviates by $+1$, (iii) it deviates by -1, and (iv) the particular landmark is not associated with the neighbor. We thus store an array of two-bit entries encoding this information, and for any entries of type (iv) we store the ID and distance to the correct landmark.

C. Finding Reference Nodes

For both two-hop and three-hop methods, we need to decide when to use NLC, and if so, which neighbor to choose as reference. We have three requirements. Reference chains need to be acyclic to be decodable. Second, reference chains must be constrained to a reasonable length for efficient decoding; for this, we introduce a parameter $maxpath$. Third, we would like to get the best compression possible. We devise a simple heuristic that ensures the first two requirements and greedily approximates the third one.

The algorithm iterates over all vertices, considering only neighbors that have already been compressed and decides if the vertex is encoded absolutely, or relatively to one of these neighbors. This step employs a gain function described below. Connecting only to neighbors that have been processed ensures there are no cycles. To restrict the length of reference chains, we only connect to neighbors if the resulting chain length cl is less than $maxpath$, as suggested in [31], [32].

The gain function expressing the benefit of a relative encoding is defined as follows. Let $s_{SRC}(v_i)$ denote the number of bits required for storing the information at some vertex v_i using SRC. Similarly, let $s_{NLC}(v_i, v_j)$ denote number of bits for a NLC encoding based on some already processed neighbor v_j. If $diff(v_i, v_j) = s_{SRC}(v_i) - s_{NLC}(v_i, v_j)$ is larger than zero, we would benefit from a relative representation, and should thus encode v_i relatively to the neighbor v_j that maximizes this difference. In practice, absolute representation leaves more options (with regards to $maxpath$) for nodes to be processed later, and thus we introduce a threshold $minGain$ for the benefit, below which we always use absolute encoding. Similarly, assume two vertices v_{j1}, and v_{j2} such that $diff(v_i, v_{j1}) = diff(v_i, v_{j2}) > minGain$, but $v_{j1}.cl > v_{j2}.cl$. Again, one could equally well decide to encode v_i relatively to v_{j1} or v_{j2}, but we should actually favor v_{j2} as this leaves more flexibility for processing later nodes. We generalize this by introducing a penalty factor for the resulting chain length $v_i.cl$ of v_i. The resulting gain function

combining benefits and penalty is $gain(v_i, v_j) = \frac{diff(v_i,v_j)}{v_j.cl+1}$. While this heuristic could possibly be improved, it proved sufficient in our experiments.

VI. Experiments

We conducted a series of experiments to evaluate the effectiveness of three-hop landmarks compared to the two-hop landmarks. We show on three large real world graphs that for small distances and for a small space budget three-hop landmarks outperform two-hop landmarks. Moreover, we show that our proposed compression methods reduce the space consumption significantly for both landmark types.

A. Experimental Setup

For our experimental evaluation we use three real world graphs obtained from snap.stanford.edu. All graphs contain a large connected component consisting of well over 90% of all vertices. We preprocess all graphs to obtain the largest connected components. The first graph, ***Astro-phys***, is a collaboration network where each node represents a scientist and there is an edge if two scientists coauthored a paper. It has 17.9K vertices, 197K edges and a diameter of 14. ***Loc-gowalla*** and ***Orkut***, are online social networks. Loc-gowalla consists of 197K vertices, 950K edges and has diameter 16, while Orkut has 3M vertices, 117M edges and a diameter of 10.

For the evaluation of the distance estimation accuracy we generated a set of 50K queries for each of the three graphs. For each query we selected one node v_i at random, and a second node v_j of distance d (from the d^{th} shell of a BFS around v_i). Our assumption is that we are constrained to store only a constant number of bits per vertex to answer (approximate) distance queries. Therefore the vertex adjacency lists cannot be stored in memory, and we also need to estimate distances between neighboring nodes. We focus on distances $d \in \{1 \dots 7\}$. Larger distances are not only extremely rare but are also not as important to estimate as accurately as smaller distances. To measure the overall quality of each method we use a single query load of mixed distances for some of the experiments. In context of social search, distance queries between close vertices are expected to be much more frequent; i.e., the people that social network users search for are usually within their close vicinity. Moreover, the error is much more important to be smaller for small distance queries than for larger. We therefore use a power law distribution over the distances and introduce Q_{mix} based on that distribution.

All algorithms were implemented in Java using fastutil [32] for compression and specialized data structures for integers. The source code is available upon request from the authors.

B. Parameter Selection

To make a fair comparison between the 2-hop and 3-hop approaches we conducted experiments to select the optimal parameters for each method. For 2-hop we found that selecting the landmarks with the highest betweenness gives the best distance estimates on average, reproducing the results in [19]. The same holds for global landmark selection in 3-hop.

For the evaluation of the 3-hop approach we need to choose the size of the global landmark set L given a fixed space budget. A larger set L makes the assigned id range of the global landmarks larger and therefore the landmark id lists of each vertex harder to compress. Therefore the estimation quality reaches a peak as $|L|$ grows and starts decaying for a given space budget. We experimented with a wide range of values up to $\sqrt{|V|}$ for each graph. We found that the best values for $|L|$ are 120, 300, and 500 for Astro-phys, Loc-gowalla, and Orkut respectively, independent of the space budget or number of local landmarks per vertex.

For both 2 and 3-hop landmark methods NLC is most commonly the optimal compression method. Because it reduces the space consumption more than BL and SRC, NLC gives lower estimate errors for the same space consumption. Also, for 3-hop RedBlue provides tighter distance bounds than Next for the same space budget. The above two parameters are discussed in detail in the following subsections.

C. Two-Hop vs Three-Hop

We compare both methods using the parameters from Section VI-B to show that given a restricted space budget, 3-hop outperforms 2-hop for all graphs. Figure 1 presents the comparison between the two methods for all three graphs for the query load Q_{mix} for different individual distances.

We observe that on average, 3-hop outperforms 2-hop in all three graphs. We also observe that the average error is always larger both for smaller distances and for smaller space budgets. However, the behavior of the two methods is not the same for all three graphs. For Astro-phys the benefit of using 3-hop is significantly larger than in the case of Loc-gowalla and Orkut. In fact, for Orkut 2-hop slightly outperforms 3-hop initially. The error lines of the two methods intersect close to 15 bits per vertex and then 3-hop gives slightly better distance estimates for the same space budget. For distances 1 to 3 on Astro-phys and Loc-gowalla the 3-hop error is smaller than the 2-hop error for any space consumption but the smaller the space consumption the greater the benefit of using our 3-hop method. For larger distance there is no benefit in using 3-hop. For Orkut the benefit of 3-hop is only for distances 1 and 2, and we observe a significant disadvantage in using 3-hop for distances larger than 4.

We can draw two basic conclusions from these plots. First, the smaller the distance we aim to estimate, the better 3-hop method works than 2-hop. Due to the higher number of landmarks for 3-hop, vertices are more likely to have a close-by landmark. Due to RedBlue, local landmarks for short distance queries tend to overlap and hence be more exact. In comparison, the smaller number of landmarks in 2-hop results in higher average errors for such queries, Second, 3-hop leads to a greater benefit when only limited space is available. This is particularity beneficial for very large graphs since even a small amount of bits per vertex introduces a very significant space consumption.

Regarding query processing, we do not observe a significant difference in speed. This is because the 2-hop methods have

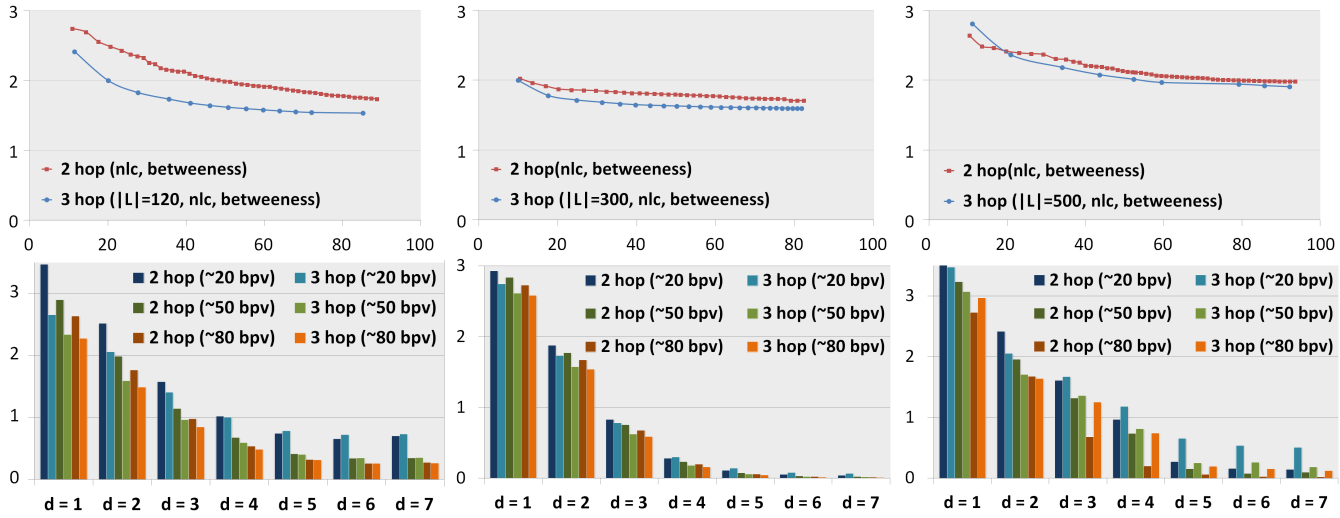

Fig. 1. **Average absolute error of 2-Hop and 3-Hop** (y-axis). Top: For queries Q_{mix} when increasing the space availability (x-axis); i.e. the number of bits that the methods store for each vertex. Bottom: For different distance queries (x-axis) and three fixed space budgets per vertex. For Astro-phys, Loc-gowalla, and Orkut (left to right).

linear query time in the number of global landmarks $|L|$ used, whereas 3-hop has quadratic query time in the number of local landmarks k which can be chosen much smaller.

D. Next vs RedBlue

Figure 2 compares the two local landmark selection methods proposed for 3-hop with respect to their average estimation error and their space consumption for different numbers of k. We experiment with $k = \{5, 10, 15, 20\}$. As can be seen, RedBlue always produces slightly smaller average errors. Most of the time, it decreases the required space per vertex dramatically. In rare cases ($k = 10$ in Orkut, and $k = 15$ in Astro-phys), however, the space required for RedBlue is higher than that of Next: RedBlue may choose different or fewer landmarks per vertex and may yield a smaller number of local landmarks and a smaller error for a fixed graph, k and L. However, it might happen that the resulting landmark id lists cannot be compressed as well, due to larger gaps. Finally, we note that the benefit of using RedBlue is larger for larger values of k. This behavior is consistent in all graphs for all k.

E. Compression Schemes

We applied the compression schemes from Section V on all three graphs for both 2-hop and 3-hop methods to compare their performance. The top plots of Figure 3 show the compression schemes for the 2-hop method. The three graphs behave very similar for all sizes of landmark sets that are used in the experiment. BL as expected consumes the most space for the same number of landmarks and NLC the least. SRC has a behavior rather similar to NLC. The bottom plots show the corresponding space consumption in the 3 compression schemes for the 3-hop method, where we fixed the total number of landmarks to be 120, 300, and 500, respectively. We observe a much greater benefit of the two compression schemes in the 3-hop case: The plots show that NLC is almost twice as good as BL, whereas SRC lies somewhere in between the numbers for 2-hop, and is almost as good for 3-hop.

VII. Conclusions

In this work, we studied the problem of efficient and accurate pairwise distance estimation in social graphs. We proposed a three-hop approach with a landmark selection framework that first selects a set of global landmarks and then assigns a small subset of them to each vertex. Moreover, we describe new compression schemes that apply both to previous two-hop methods and our new three-hop method.

We draw two main conclusions from our experimental evaluation on several real world graphs. First, our three-hop approach outperforms existing two-hop methods for queries of small distances, but does not improve accuracy in some other cases. Second, applying suitable compression schemes on distance tables can significantly decrease space requirements for two-hop as well as three-hop approaches, and is in fact essential when comparing the practical applicability of the different approaches in a fair way. **Acknowledgement** This research was supported by NSF Grant IIS-1117829 “Efficient Query Processing in Large Search Engines”.

References

[1] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes, “In transit to constant time shortest-path queries in road networks,” *ALENEX*, 2007.
[2] A. V. Goldberg, H. Kaplan, and R. F. F. Werneck, “Reach for a*: Effcient point-to-point shortest path algorithms,” Microsoft Research, Tech. Rep. MSR-TR-2005-132, 2005.
[3] A. V. Goldberg and C. Harrelson, “Computing the shortest path: *A** search meets graph theory,” in *SODA*, 2005.
[4] A. V. Goldberg, “Point-to-point shortest path algorithms with preprocessing,” in *SOFSEM (1)*, 2007.
[5] Y. Xiao, W. Wu, J. Pei, W. Wang, and Z. He, “Efficiently indexing shortest paths by exploiting symmetry in graphs,” in *EDBT*, 2009.
[6] A. W. Fu, H. Wu, J. Cheng, and R. C. Wong, “IS-LABEL: an independent-set based labeling scheme for point-to-point distance querying,” *PVLDB*, vol. 6, no. 6, 2013.
[7] T. Akiba, Y. Iwata, and Y. Yoshida, “Fast exact shortest-path distance queries on large networks by pruned landmark labeling,” in *SIGMOD*, 2013.

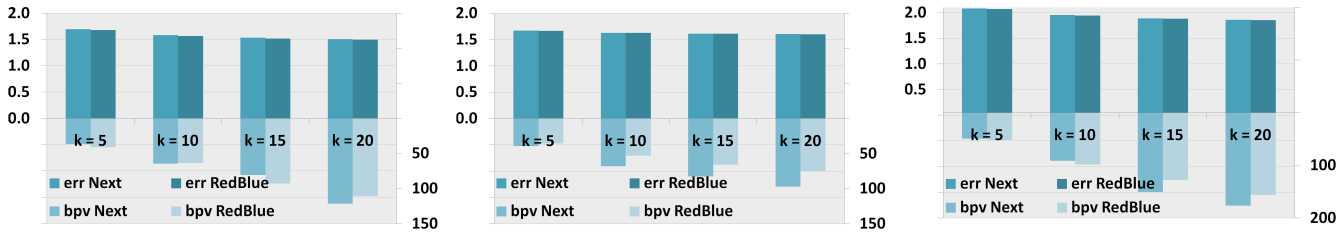

Fig. 2. **Next vs. RedBlue for different number of local landmarks k:** Upper bars show average absolute error of Next (lighter) and RedBlue (darker) on queries Q_{mix} for graphs Astro-phys, Loc-gowalla, and Orkut (left to right). Lower bars show the number of bits per vertex required for both methods.

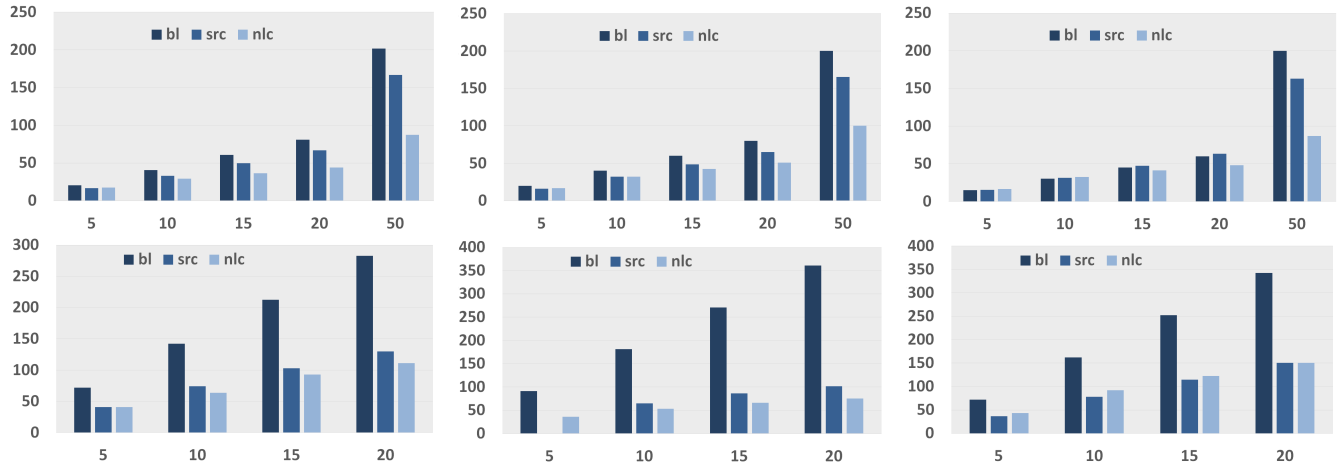

Fig. 3. Compression schemes for 2-hop (top) and 3-hop (bottom) for Astro-phys, Loc-gowalla, and Orkut (left to right). The x-axis denotes the number of landmarks (2-hop) resp. k (3-hop), the y-axis the number of bits per vertex.

[8] J. Bourgain, "On lipschitz embedding of finite metric spaces in hilbert space," *Israel J. Math.*, vol. 52, no. 1, 1985.

[9] J. Matousek, "On the distortion required for embedding finite metric spaces into normed spaces," *Israel J. Math.*, vol. 93, no. 1, 1996.

[10] Y. Bartal, "On approximating arbitrary metrices by tree metrics," in *STOC*, 1998.

[11] J. Fakcharoenphol, S. Rao, and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," *J. Comput. Syst. Sci.*, vol. 69, no. 3, 2004.

[12] T.-H. H. Chan, K. Dhamdhere, A. Gupta, J. M. Kleinberg, and A. Slivkins, "Metric embeddings with relaxed guarantees," *SIAM J. Comput.*, vol. 38, no. 6, 2009.

[13] X. Zhao, A. Sala, H. Zheng, and B. Y. Zhao, "Efficient shortest paths on massive social graphs," in *CollaborateCom*, 2011.

[14] S. Baswana, "Streaming algorithm for graph spanners - single pass and constant processing time per edge," *Inf. Process. Lett.*, vol. 106, no. 3, 2008.

[15] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, "Graph distances in the streaming model: the value of space," in *SODA*, 2005.

[16] M. Thorup and U. Zwick, "Approximate distance oracles," *J. ACM*, vol. 52, no. 1, 2005.

[17] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. de Castro Reis, and B. A. Ribeiro-Neto, "Efficient search ranking in social networks," in *CIKM*, 2007.

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

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

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

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

[22] J. M. Kleinberg, A. Slivkins, and T. Wexler, "Triangulation and embedding using small sets of beacons," *J. ACM*, vol. 56, no. 6, 2009.

[23] M. Christoforaki and T. Suel, "Estimating pairwise distances in large graphs," in *BigData*, 2014.

[24] K. Tretyakov, A. Armas-Cervantes, L. García-Bañuelos, J. Vilo, and M. Dumas, "Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs," in *CIKM*, 2011.

[25] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry, "3-hop: a high-compression indexing scheme for reachability query," in *SIGMOD*, 2009.

[26] R. Jin, Y. Xiang, N. Ruan, and H. Wang, "Efficiently answering reachability queries on very large directed graphs," in *SIGMOD*, 2008.

[27] S. Trißl and U. Leser, "Fast and practical indexing and querying of very large graphs," in *SIGMOD*, 2007.

[28] R. Schenkel, A. Theobald, and G. Weikum, "Hopi: An efficient connection index for complex xml document collections," in *EDBT*, 2004.

[29] M. Adler and M. Mitzenmacher, "Towards compressing web graphs," in *DCC*, 2001.

[30] T. Suel and J. Yuan, "Compressing the graph structure of the web," in *DCC*, 2001.

[31] K. H. Randall, R. Stata, J. L. Wiener, and R. Wickremesinghe, "The link database: Fast access to graphs of the web," in *DCC*, 2002.

[32] P. Boldi and S. Vigna, "The webgraph framework I: compression techniques," in *WWW*, 2004.

[33] F. Claude and G. Navarro, "A fast and compact web graph representation," in *SPIRE*, 2007.

[34] G. Buehrer and K. Chellapilla, "A scalable pattern mining approach to web graph compression with communities," in *WSDM*, 2008.

[35] P. Boldi, M. Santini, and S. Vigna, "Permuting web graphs," in *WAW*, 2009.

[36] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *KDD*, 2009.

[37] F. Claude and S. Ladra, "Practical representations for web and social graphs," in *CIKM*, 2011.

[38] C. Hernández and G. Navarro, "Compressed representation of web and social networks via dense subgraphs," in *SPIRE*, 2012.

[39] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Computing Surveys*, vol. 38, no. 2, 2006.