Toward a Distance Oracle for Billion-Node Graphs

Zichao Qi
Tsinghua University
Beijing, China
qizichao@gmail.com

Yanghua Xiao*
Fudan University
Shanghai, China
shawyh@fudan.edu.cn

Bin Shao
Microsoft Research Asia
Beijing, China
binshao@microsoft.com

Haixun Wang
Microsoft Research Asia
Beijing, China
haixun@gmail.com

ABSTRACT

The emergence of real life graphs with billions of nodes poses significant challenges for managing and querying these graphs. One of the fundamental queries submitted to graphs is the *shortest distance query*. Online BFS (breadth-first search) and offline pre-computing pairwise shortest distances are prohibitive in time or space complexity for billion-node graphs. In this paper, we study the feasibility of building *distance oracles* for billion-node graphs. A distance oracle provides approximate answers to shortest distance queries by using a pre-computed data structure for the graph. Sketch-based distance oracles are good candidates because they assign each vertex a sketch of bounded size, which means they have linear space complexity. However, state-of-the-art sketch-based distance oracles lack efficiency or accuracy when dealing with big graphs. In this paper, we address the scalability and accuracy issues by focusing on optimizing the three key factors that affect the performance of distance oracles: *landmark selection*, *distributed BFS*, and *answer generation*. We conduct extensive experiments on both real networks and synthetic networks to show that we can build distance oracles of affordable cost and efficiently answer shortest distance queries even for billion-node graphs.

1. INTRODUCTION

The emergence of very large graphs, including the World Wide Web [1], the Facebook social network [2], LinkedData [3], and various biological graphs [4], has brought tremendous challenges to data management. A first step toward effectively managing large graphs is to support a class of useful graph operators. In this paper, we are concerned with one of the most useful graph operators: the *shortest distance query*.

Shortest distance queries are important for two reasons. First, *shortest distance queries are indispensable in many graph applications*. For example, in a social network, we are interested in finding the shortest distance between two users. Second, *shortest distance queries are indispensable building blocks for many advanced analytical tasks associated with large graphs*. For example, in social network analysis, a person's importance can be measured by his centrality, which is defined as the maximal shortest distance from the person to any other person. Clearly, centrality is calculated by shortest distance queries. Many other measures, such as betweenness and network diameter, are also based on shortest distances.

1.1 Shortest Distances in Large Graphs

Although much work has been done on computing shortest distances, there is no realistic solutions for efficient shortest distance computation on billion node graphs. For web-scale graphs, it takes hours for the Dijkstra algorithm to find the shortest distance between two nodes [5]. Alternatively, we can pre-compute and store the shortest distances for all pairs of nodes, and use table lookups to answer shortest distance queries at the time of the query. Clearly, this approach requires quadratic space.

In one aspect, the above approaches provide two extreme solutions: The Dijkstra algorithm does not use any pre-computed data structure, but it has large time complexity. In fact, a naive implementation of the Dijkstra algorithm has time complexity $O(|V|^2)$. On the other hand, the table lookup approach has a constant time complexity but a very large space complexity: The pre-computed data structure is of size $O(|V|^2)$. For a web-scale graph where $|V|$ is in the 10^9 range, neither is realistic.

In this paper, we propose using a *distance oracle* to answer shortest distance queries on billion node graphs. A distance oracle is a pre-computed data structure that enables us to find the (approximate) shortest distance between any two vertices in constant time. A distance oracle is feasible for billion node graphs if it satisfies the following criteria:

1. Its pre-computed data structure has a small *space complexity*. For billion-node graphs, we can only afford linear, or sub-linear pre-computed data structures.
2. Its construction has a small *time complexity*. Although distance oracles are created offline, for billion-node graphs, we cannot afford algorithms of quadratic time complexity.
3. It answers shortest distance queries in *constant time*.
4. It answers shortest distance queries with high *accuracy*.

The four criteria are interrelated. In general, the more sophisticated the distance oracle (the more space and the longer time it takes to build the distance oracle), the higher accuracy and the lower time complexity of online query processing. Thus, the key to building a good distance oracle is to seek appropriate tradeoffs. The major objective of our work is to *build a distance oracle that is of linear space complexity, takes constant time for query answering, produces answers of high accuracy, and scales up to billion-node graphs.*

*Correspondence author.This work was supported by the National NSFC (No. 61003001, 61170006, 61171132, 61033010); Specialized Research Fund for the Doctoral Program of Higher Education No. 20100071120032; Shanghai Municipal Science and Technology Commission with Funding No.13511505302; NSF of Jiangsu Province (No. BK2010280).

Proceedings of the VLDB Endowment, Vol. 7, No. 2

1.2 Sketch-based Distance Oracles

We focus on a class of distance oracles that create *a bounded-size sketch for each vertex* as a pre-computed data structure for shortest distance queries. Since the size of the sketch for each vertex is bounded by a constant, sketch-based solutions require linear space only. If the sketch encodes enough useful information, it can produce highly accurate answers in a short time (in most cases a constant time). In the following, we classify sketched-based distance oracles into two categories and review each of them. To simplify the description, we focus on unweighted and undirected simple graphs unless otherwise stated.

Shortest-distance-based sketches. Let $d(u, v)$ denote the shortest distance between vertex u and v in a graph $G(V, E)$. First, we select a set of vertices L called landmarks. Then, we compute the shortest distances between each landmark to each vertex in V. The sketch of a vertex u is simply $\{(w, d(u, w))|w \in L\}$, where $d(u, w)$ is the shortest distance between u and w. For any vertices u and v, their shortest distance can be estimated as the minimal value of $d(u, w) + d(w, v)$ over all $w \in L$.

This distance oracle takes $O(|L|\cdot|V|)$ space and answers queries in $O(|L|)$ time. The accuracy of query answering is primarily determined by landmark selection. In the worst case, if no landmarks exists in a shortest path between u and v, the estimation of $d(u, v)$ will be erroneous. Hence, distance oracles of this category focus on selecting landmarks that a large number of shortest paths pass through them.

Coordinate-based sketches. We embed a graph into a geometric space so that shortest distances in the embedded geometric space preserve the shortest distances in the graph. Then, the shortest distance is estimated as the coordinate distance. Let c be the dimensionality of the embedded space. The space complexity of the coordinate based solution is $\Theta(c \cdot |V|)$. The query can be answered in $\Theta(c)$ time, independent of the graph size.

Graph embedding is the key to the success of distance oracles of this kind. State-of-the-art approaches [6, 7] first select a set of landmark vertices using certain heuristics (for example, selecting vertices of large degrees). For each landmark, BFS is performed to calculate the shortest distance between each landmark and each vertex. Then, we fix the coordinates of these landmarks using certain global optimization algorithms based on the accurate distances between landmarks. Finally, for each non-landmark vertex, we compute its coordinates according to its distances to the landmarks. Finding the coordinates of landmarks and non-landmark vertices is formulated as an optimization problem with the objective of minimizing the sum of squared errors:

$$\sqrt{\sum_{(u,v)} (|c(u) - c(v)| - d(u, v))^2} \quad (1)$$

where $c(u)$ is the coordinates of vertex u. Usually, this kind of optimization problem can be solved by the *simplex downhill* method [8].

1.3 Challenges for Billion-Node Graphs

We first discuss factors that affect the performance of distance oracles. We then reveal the weaknesses of current approaches and describe our contributions.

Key Performance Factors

For billion-node graphs, there are at least three key factors that have a major impact on the performance and quality of sketch-based distance oracles:

1. Landmark selection is critical for shortest distance estimation. Most heuristics for landmark selection, such as betweenness and centrality, are costly to evaluate in large graphs. For instance, computing betweenness takes $O(|V| \cdot |E|)$ time, which is prohibitive for billion-node graphs.
2. After landmarks have been determined, a large number of BFSs are required to find the distances between landmarks and each vertex in the graph. The total cost is $O(|V| \cdot |L|)$, which is a performance bottleneck for billion-node graphs.
3. For a pair of vertices, the distance oracle gives an estimation of their shortest distance. The quality of the estimation depends on the precomputed information as well as the rules of estimation. More sophisticated rules give more accurate answers.

Weaknesses of Previous Distance Oracles

Many distance oracles have been proposed (Section 8 provides more details). However, to the best of our knowledge, none of the existing approaches are able to provide accurate shortest distance estimation for billion-node graphs. The reason is three-fold:

- First, some distance oracles compromise accuracy for cost. For example, many distance oracles [6, 9] use vertices' degrees, instead of their centrality, for landmark selection to reduce the cost of building distance oracles. However, as we will show, degree is a very poor approximation of centrality. This leads to poor estimation of shortest distances.
- Second, some distance oracles take very long time to create. For example, Tretyakov et al. [10] randomly select M vertex pairs and count the number of shortest paths between the pairs that pass through a vertex to approximate the vertex's betweenness. Unfortunately, such approximation takes 23h for a 0.5 billion node graph on a 32-core server with 256G memory [10].
- Third, none of the previous distance oracles were designed for distributed graphs. Although some solutions [6, 5] store data or perform some algorithmic steps on a distributed platform, they use the platform as it is without having graph or distance oracle specific optimization. In other words, the power of distributed graph computing has not yet been fully leveraged for distance oracles.

As a result of the above weaknesses, most current research only handles million-node graphs. The largest graphs reported in previous distance oracle work include a social network with 43 million nodes [6] on 50 machines, a web graph with 65M nodes [5] on a distributed web graph store, and the Skype network with 0.5B nodes on a 32-core server with 256G memory. In our work, we use only a dozen commodity PCs to determine shortest distances in a billion-node graph.

Contribution and Organization

In this paper, we introduce several novel techniques to enable accurate estimation of shortest distances in billion-node graphs for the first time. The innovation lies in three aspects:

- *Local vs. Global Computation.* First, accurate estimation of betweenness is the key to landmark selection. However, betweenness is a *global* feature, meaning the computation involves the entire graph. Apparently, computing global features on a large graph is expensive: In order to compute the betweenness of a vertex, we need to find all shortest paths in the entire graph that pass this vertex. Also, a billion-node graph is usually distributed over multiple machines, and graph exploration in the *local* machine is much cheaper than graph exploration across the network. In view of this, we develop a highly accurate centrality estimator through graph exploration in each *local* machine. We show that our approach incurs little loss in accuracy but is a much more efficient. This makes it possible to obtain high quality landmarks in billion-node graphs.
- *Fast Distributed BFS.* BFS is inevitable for creating a distance oracle: The distance oracle relies on the exact shortest distance between every (landmark, vertex) pair to estimate the shortest distance between every (vertex, vertex) pair, and BFS is needed to calculate the exact shortest distances. As we mentioned, we deploy billion-node graphs in a cluster. Thus, the challenge becomes how to support distributed BFS in an efficient manner. In our work, we introduce a novel vertex caching mechanism to dramatically reduce network communication in BFS. We show that this

mechanism, together with the local centrality estimation mechanism mentioned above, enables our approach to scale up to billion-node graphs.

- *Accurate Query Answering.* How to effectively use the information provided by the distance oracle to accurately estimate the distance between a vertex pair is a critical problem. In many real life networks, in particular social networks, the average distance between a vertex pair is very short (e.g., within 6 steps). Hence, in these small-diameter networks, even a minor estimation error will lead to misunderstandings of the network. To improve the distance estimation accuracy, we propose a new distance estimation mechanism for a vertex pair u, v based on both the shortest distances from them to landmarks and their coordinate distance in an embedded space. Theoretical results and experimental results show that our new rule produces more accurate distances than estimations purely using either shortest distance to landmarks or coordinate distance.

The rest of the paper is organized as follows. Section 2 and 3 describe the background for distance oracles and the system architecture. Section 4 describes a novel landmark selection method. Section 5 introduces how to optimize distributed BFS for shortest distance computation. Section 6 discusses how to estimate shortest distance using the distance oracle. We present our experimental results in Section 7. We review related work in Section 8 and conclude in Section 9.

2. BACKGROUND

In this section, we introduce the background for our distance oracle. Specifically, we introduce the graph system on which our solution is built, the graph embedding technique and landmark selection in shortest path query answering.

2.1 The Graph Engine

We use Trinity [11], an in-memory, distributed graph infrastructure, to manage web scale graphs. Trinity combines the RAM of multiple machines into a unified memory space to user programs. Trinity supports very efficient memory-based graph exploration. For example, it is able to explore 2.2 million edges within 1/10 of a second in a network of Facebook size and distribution. Furthermore, Trinity also provides an efficient bulk message passing mechanism, and supports a Pregel [12] like Bulk Synchronous Parallel (BSP) computation model. In one experiment, using just 8 machines, one BSP iteration on a synthetic, power-law graph of 1 billion nodes and 13 billion edges takes less than 60 seconds. Trinity's efficient graph exploration and bulk message passing mechanism lay the foundation for developing our graph distance oracle.

2.2 Graph Embedding

Graph embedding is a key step to build a coordinate-based distance oracle. We will see that it is used in two steps (S3 and S4) of our distance oracle in Section 3.1. In graph embedding, we embed the graph into a geometric space. The geometric space we use is the Hyperbolic space under the Hyperboloid model. We then use the two-phase embedding method [13, 7, 6] to assign coordinates to vertices. That is, first we determine the coordinates of landmarks, and then we use the coordinates of the landmarks to determine the coordinates of other vertices. To determine the coordinates, we solve an optimization problem using the Simplex method. In the following, we first discuss the Hyperbolic space, then we describe how coordinates are determined.

The Hyperbolic space under the Hyperboloid model. There are three widely used geometric spaces: Euclidean [13, 7], Spherical [14], and Hyperbolic [15, 6]. Many real graphs such as the world wide web consist of a core in the center and many tendrils connecting to the core. When embedding such graphs into an Euclidean space, two vertices far from the core tend to have short distances as the line connecting them does not need to pass through the core. The Hyperbolic space is proposed to overcome this weakness [15, 6]. In a Hyperbolic space, the distance between two vertices is calculated along a curved line bent towards the origin, which makes the shortest path between non-core vertices tend to pass through the core. There are different models for coordinate assignment and distance computation over the same Hyperbolic space. The most widely used is the Hyperboloid model. In this model, a parameter called *curvature* $\delta \leq 0$ is used to control the bend towards the core. More formally, for a Hyperboloid model, the distance between $x = \langle x_1, x_2, ..., x_c \rangle$ and $y = \langle y_1, y_2, ..., y_c \rangle$ is defined as

$$\bar{d}(x,y) = arccosh\left(\sqrt{(1+\sum_{i=1}^{c} {x_i}^2)(1+\sum_{i=1}^{c} {y_i}^2)} - \sum_{i=1}^{c} x_i y_i\right) \times |\delta| \tag{2}$$

which contains two parts: *distance* and *curvature*. The expression before $|\delta|$ can be considered as a distance metric independent of δ. It was shown that the Hyperboloid model has two major strengths [6]: (1) it is computationally simpler than other models; and (2) the model is more flexible to tune since it can be separated into two disjointed components.

Assigning Coordinates. The objective is to minimize the distance distortion when embedding a graph into a geometric space. Let $c(u)$ be the coordinate of vertex u. We can measure the distortion by the absolute error:

$$err(u,v) = |d(u,v) - \bar{d}(c(u), c(v))| \tag{3}$$

or by the relative error:

$$err'(u,v) = \frac{err(u,v)}{d(u,v)} \tag{4}$$

Coordinate learning is a typical optimization problem. In step S3, we learn the coordinates of landmarks, given their shortest distances. We minimize the following objective function:

$$\arg \min_{\{c(v_i):v_i \in L\}} \sqrt{\sum_{1 \leq i \neq j \leq |L|} e(v_i, v_j)^2} \tag{5}$$

where $e(u,v)$ is given by Eq 3 or Eq 4. We use the *Simplex downhill* [8] method to calculate the optimal coordinates.

2.3 Landmark Selection

We review the landmark selection problem in shortest distance query answering. We first give the principle of landmark selection. A good set of landmarks should be *complete* and *minimal*.

Our goal is to predict the shortest distance between any two vertices with high accuracy. Let $L = \{l_1, l_2, ..., l_k\}$ be a set of landmarks. For any vertex pair (u,v), we estimate their shortest distance from their shortest distances to landmarks in L. If a landmark exists that is on a shortest path between u and v, then the estimation is accurate. Formally, we say a vertex pair (u,v) is *covered* by L if a shortest path exists between u and v that passes through a landmark in L. If all vertex pairs are covered by the landmark set L, then L is *complete*. On the other hand, if all vertex pairs covered by landmark l_i are also covered by landmark l_j $(i \neq j)$, then l_i is redundant. Formally, a landmark set L is minimal if there exists no $L' \subset L$ such that L' can cover the vertex pairs covered by L.

This leads to the *minimal complete landmark (MCL)* problem [9].

PROBLEM DEFINITION 1 (MCL). *Given a graph G, find a minimal landmark set that covers all vertex pairs.*

Practically, we have to relax the requirement of covering all vertex pairs in the above definition. A landmark set that is complete is usually very large, which negatively impacts the performance of query answering and may lead to unaffordable space cost. Thus, we want to put a size constraint on the set of landmarks. Of course, with the size constraint, we cannot ensure its completeness. This leads to a new problem: *maximal coverage landmark set with size constraint (MCL-s)* [9]:

PROBLEM DEFINITION 2 (MCL-S). *Given a graph G and a positive integer k, find a landmark set of size k that covers the largest number of vertex pairs.*

Both MCL and MCL-s are NP-Hard [9]. Many heuristics have been proposed for the two problems. Most heuristics follow the same framework: *greedily select the most promising vertex as a landmark until the upper limit is reached*. To do this, the key is how to measure a vertex as a good landmark. There are several measures, including node degree, closeness centrality, and betweenness. Betweenness has been empirically shown to be the best measure in most cases [9].

For a vertex v, its betweenness is the fraction of all shortest paths in the graph that pass through v. Formally, we have:

$$bc(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad (6)$$

where σ_{st} is the number of shortest paths between s and t, and $\sigma_{st}(v)$ is the number of shortest paths in σ_{st} that pass through v.

3. OVERVIEW

In this section, we give an overview of the approach we are taking to build a distance oracle. In addition, we also give a complexity analysis of our approach.

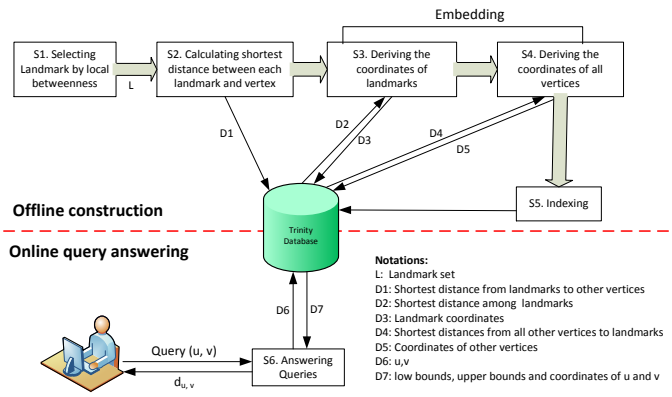


Figure 1: Architecture of a coordinate-based distance oracle.

3.1 The Flowchart

We build a coordinate-based distance oracle. Its architecture, which is shown in Figure 1, consists of two parts: offline distance oracle construction (steps S1-S5) and online shortest distance query answering (step S6). We will elaborate on query answering in Section 6. In this section, we focus on the construction of the distance oracle.

We build a distance oracle for a graph $G(V, E)$. The distance oracle consists of a set of nodes L in V that are known as landmarks, a set of coordinates $\{c(u)|u \in V\}$, and a set of distance vectors $\{d(u)|u \in V\}$. Each $c(u)$ is a c-dimensional vector $< x_1, ..., x_c >$ and each $d(u)$ is a $|L|$-dimensional vector $< d_1, ..., d_{|L|} >$, where d_i is the shortest distance to the i-th landmark in L. Next, we describe each step in building a distance oracle:

- *S1: Selecting landmarks by local betweenness*. We select top-$|L|$ vertices with the largest *local betweenness* as landmarks. As we mentioned, betweenness is widely accepted as the best heuristics for landmark selection. But computing *exact* betweenness is very costly for big graphs, especially when the graph is distributed. We introduce a new concept called local betweenness, and we use local betweenness to approximate the exact betweenness. Local betweenness only depends on shortest paths in local graphs that are confined in single machines. Thus, its computation is lightweight and does not require network communication. Nevertheless, we show that it is an extremely good approximation of the exact betweenness.
- *S2: Calculating shortest distances between each landmark and each vertex.* We use BFS to compute the shortest distances for unweighted graphs. Each vertex v maintains an $|L|$-dimensional vector $\langle d_1, ..., d_{|L|} \rangle$, where d_i is the shortest distance between v and landmark L_i. This requires $O(|L| \cdot |V|)$ space, which means for billion node graphs we cannot afford having a large value of $|L|$. In this work, we manage to get good results with a small $|L|$ (typically 100).
- *S3: Deriving the coordinates of landmarks*. We learn the coordinates of landmarks by minimizing the difference between the exact distances (calculated in S2) and the coordinate-based distances. Based on Eq. 5, we use the following objective function

$$\arg \min_{\{c(v_i)|v_i \in L\}} \sqrt{\sum_{1 \leq i \neq j \leq |L|} [\bar{d}(c(v_i), c(v_j)) - d(v_i, v_j)]^2} \quad (7)$$

where $c(u)$ is the coordinates of vertex u, and $\bar{d}()$ is the coordinate distance (Eq. 2), and $d()$ is the shortest distance. We use the simplex downhill method [8] to solve the optimizing problem.
- *S4: Deriving the coordinates of all vertices*. We learn the coordinates of each vertex based on their distances to landmarks (calculated in S2) and the landmarks' coordinates (calculated in S3). We use the following objective function:

$$\arg \min_{c(u)} \sqrt{\sum_{1 \leq i \leq t} [\bar{d}(c(u), c(v_i)) - d(u, v_i)]^2} \quad (8)$$

Unlike in S3, here, we already know the coordinates of landmarks. Furthermore, we randomly select a small number of t ($t < |L|$, typically $t = 16$ as in [6, 7]) landmarks to save time. The complexity of the Simplex downhill algorithm for deriving the optimal coordinates is linear to the number of used landmarks. It can be shown that with small values of $t = 16$ and $|L| = 100$, we can derive coordinates for all vertices with high accuracy.
- *S5: Indexing*. As we have mentioned, each vertex is associated with coordinates $\langle x_1, ..., x_c \rangle$ and a distance vector $\langle d_1, ..., d_{|L|} \rangle$. We index the data so that we can support fast retrieval of such information by vertex ID.

3.2 Complexity Analysis

Our distance oracle gives highly accurate answers to shortest-distance queries in constant time (See Section 6 for detail). We will evaluate its accuracy with extensive experiments. Here, we analyze the time cost of constructing the distance oracle and the space cost of the distance oracle itself.

Time complexity. Our distance oracle can be constructed in linear time with respect to the graph size. The time to construct the distance oracle is dominated by graph embedding, whose complexity is determined by the simplex downhill method. In general, simplex downhill with an objective function g takes $O(mD \times f(g))$ time, where $f(g)$ is the cost to evaluate g, D is the total number of dimensions, and m is the number of iterations[1]. We run simplex downhill in S3 and S4. In S3, we have $D = |L|c$ and $f(g) = |L|^2 c$. The second equation holds because we need to calculate the pairwise distance of $|L|$ c-dimensional coordinates. In all, the time complexity is $O(m|L|^3 c^2)$. In S4, we need to learn coordinates of $|V| - |L|$ vertices. To learn each vertex's coordinate, we have $D = c$ and $f(g) = tc$, where t is the number of selected landmarks (see Eq. 8). Hence, it costs $O(mtc^2)$ time to learn the coordinate for a single vertex. In all, S4 costs $((|V| - |L|)mtc^2)$ time. Hence, the time complexity to construct our distance oracle is $O((|V| - |L|)mtc^2 + mt|L|^3 c^2)$. In general, $|L|$, c, m, and t are small constants (typically $|L| = 100$, $c = 10$ and $t = 16$).

[1]For simplicity of analysis, we always assume that simplex downhill runs in m iterations.

Compared to the size of the graph (billion-node scale), these constants can be omitted. Hence, in general, it only needs linear time to construct the distance oracle.

Space cost. Our distance oracle takes $O((|L| + c)|V|)$ space, which is linear with respect to the graph size. Next, we show that for real life billion-node graphs, *the distance oracle requires an amount of space that is less than or comparable to the size of the graph*. On distributed, in-memory graph systems, the distance oracle can be deployed together with the graph on a dozen of machines. More specifically, let $\langle d \rangle$ be the average degree of the graph. Using 32 bits, we can encode 4 billion vertices. To support even larger graphs, we may use 64 bits or 8 bytes to encode the vertex id. Thus, for each vertex v, we need $8\langle d \rangle$ memory to store its neighbors. The distance vectors and the coordinates need $|L| + 8c$ space. Because most real graphs are small world networks, in general, 1 byte is enough to store the geodesic distance. The coordinates are real numbers, so each dimension needs 8 bytes. Hence, when $\frac{|L|}{8} + c < \langle d \rangle$, the space required by the distance oracle is smaller than the size of the graph. For example, when $|L| = 100$ and $c = 10$ (these are typical settings used in our distance oracle) we have $\frac{|L|}{8} + c = 22.5$. Hence, if $\langle d \rangle > 22.5$, the distance oracle takes less space than the graph. Most real networks satisfy this constraint, including the Facebook network which has an average degree of 125.

Given the above typical settings, our distance oracle is realistic even on the Facebook network with almost 1 billion users. Each vertex in the Facebook network needs $8\langle d \rangle + |L| + 8c = 1000 + 100 + 80 = 1180$ bytes. Overall, Facebook needs about 1TB of memory. This means that Facebook and the distance oracle on Facebook can be deployed on one or two dozens of commodity servers.

4. LANDMARK SELECTION

It is known that betweenness is an effective heuristic for landmark selection. But on billion node graphs, calculating betweenness is too costly. In this section, we argue that we need a distributed, lightweight, approximate betweenness measure. We propose *local betweenness* as approximate betweenness, and we introduce a lightweight, distributed method of computing local betweenness.

4.1 Motivation

We firsts show that *approximate betweenness* is a more appropriate choice than the exact betweenness on big graphs. There are three reasons:

- First, calculating the exact betweenness is costly. The fastest algorithm needs $O(|V||E|)$ time to compute exact betweenness [16]. This is computationally prohibitive for billion-node graphs. Furthermore, it is hard to reduce the complexity. By its definition, the calculation of betweenness relies on the shortest path computation. Enumerating all shortest paths costs at least $O(|V||E|)$ time, as finding all shortest paths from a single vertex costs at least $O(|E|)$ time.
- Second, although some parallel approaches for computing exact betweenness have been proposed [17, 18, 19, 20, 21], in general, it is still very costly for billion-node graphs. A straightforward parallel compuation of the exact betweenness has complexity $\Omega(|V|^2)$ [19], which is prohibitive on big graphs. Some efforts focused on reducing the space cost to $O(|V| + |E|)$ but the cost of all-pair path enumeration cannot be reduced.
- Third, it may not be necessary to find the exact betweenness: Eventually, we only use the top-k betweenness vertices as landmarks anyway. If there exists a lightweight, approximate measure, whose top-k vertices coincide with the top-k vertices ranked by the exact betweenness measure, we can ensure the optimality of landmark selection while avoiding the cost of finding exact betweenness.

Next, we show that a *lightweight distributed solution* is necessary for calculating the approximate betweenness on big graphs. *Most previous approximate betweenness methods are implemented on centralized platforms.* For example, betweenness can be approximated by counting only the shortest paths below a certain length (which can be calculated with significantly less cost) [22] or shortest paths starting from a limited number of vertices [17]. These betweenness can be easily implemented on centralized platforms. But there are many obstacles to extending these approximate solutions to large, distributed graphs. Consider the approximate solution [17] that enumerates the shortest paths starting from k randomly selected vertices. Overall $O(k|E|)$ communication is needed, and it dominates the computation cost.

4.2 Local Betweenness

We propose an efficient and effective solution to computing approximate betweenness for large graphs distributed across many machines. Instead of finding the shortest paths in the entire graph, we find the shortest paths in each machine. Then, we use the shortest paths in each machine to compute the betweenness of vertices on that machine. We call betweenness computed this way *local* betweenness. Clearly, the computation does not incur any network communication. After we find local betweenness for all vertices, we use a single round of communication to find the top-k vertices that have the highest local betweenness value, and we use these vertices as landmarks.

We show that, contrary to the perception that local betweenness is very inaccurate because each machine only contains a small portion of the entire graph, it turns out to be a surprisingly good alternative for the exact betweenness measure.

4.2.1 Local graphs

A graph G can distribute over a set of machines by hashing on the vertex id. Hash-based vertex distribution might not the optimal way of partitioning a graph, but it incurs the least cost, and hence it has been widely accepted as the de facto graph partitioning scheme for very large graphs. Each machine i keeps a set of vertices V_i as well as the adjacent list of each vertex in V_i. Specifically, let $N(v)$ denote v's neighboring vertices for any $v \in V_i$, and let $N(V_i)$ denote all of V_i's neighbors, that is, $N(V_i) = \cup_{v \in V_i} N(v)$. Thus, machine i contains information of V_i and $N(V_i)$. We define the *local graph* and the *extended local graph* on machine i:

- The *local graph* $G_i = (V_i, E_i)$ is the subgraph induced by V_i, where E_i contains edge (u, v) if both u and v are in V_i.
- The *extended local graph* $G'_i = (V'_i, E'_i)$ contains both V_i and V_i's neighbors as vertices, that is, $V'_i = V_i \cup N(V_i)$, and E'_i contains every edge (u, v) if either u or v is in V_i.

Clearly, G_i is a subgraph of G'_i. Example 1 illustrates the relationship between G_i and G'_i.

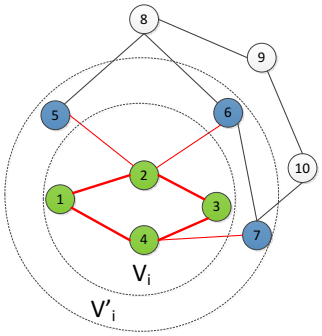


Figure 2: The local graph, the extended local graph, and the entire graph, from the perspective of machine i.

EXAMPLE 1 (LOCAL GRAPHS). *Consider a graph with 10 vertices, as shown in Figure 2. Assume machine i contains 4 vertices, that is, $V_i = \{1, 2, 3, 4\}$. The graph in the innermost circle is the local graph. Machine i also has information about vertices $5, 6,$*

and 7, as they are in the adjacency lists of vertex 2 and 4. The graph inside the second circle, except for the edge $(6,7)$, is the extended local graph. Machine i does not realize the existence of vertices $8, 9$, and 10. Note that edge $(6,7)$ does not belong to the extended local graph since none of its ends belongs to V_i.

Next, we show that the extended local graph G_i' contains enough information to effectively approximate the exact betweenness. We first quantify the size of the extended local graph (Lemma 1), and then we quantify the quality of the shortest paths in the extended local graph (Lemma 2).

LEMMA 1 (SIZE OF THE EXTENDED LOCAL GRAPH). *Suppose a graph G is randomly partitioned over k machines, and the average degree of G is $\langle d \rangle$. Then, for any extended local graph G_i', its expected number of edges is $\frac{|E|(2k-1)}{k^2}$, and its expected number of vertices is at most $\frac{|V|}{k}[1 + \frac{\langle d \rangle (k-1)}{k}]$.*

PROOF. Local graph G_i has on average $\frac{|V|}{k}$ vertices. Each vertex v has on average $\langle d \rangle$ neighbors. Among them, $1/k$ is expected to reside in the same machine as v does. Thus, vertex v has on average $\langle d \rangle (1 - \frac{1}{k})$ remote neighbors. Some vertices may share remote neighbors. Consequently, the expected size of V_i' is at most

$$E(|V_i'|) = \frac{|V|}{k}(1 + \frac{\langle d \rangle (k-1)}{k})$$

The probability that one edge is not in G_i' is equivalent to the probability that both of its two ends do not belong to G_i. This probability is $p = (1 - \frac{1}{k})^2$. Thus, the expected size of E_i' is

$$E(|E_i'|) = |E|(1-p) = \frac{|E|(2k-1)}{|k^2|}$$

□

From Lemma 1, we see that when k is large, $|V_i'|$ is approximately $|V|\frac{\langle d \rangle + 1}{k}$ on average and $|E_i'|$ is approximately $\frac{2|E|}{k}$ on average. In particular, $|V_i'|$ is significantly smaller than $|V|$ when $k \geq \langle d \rangle + 1$, and $|E_i'|$ is consistently smaller than $|E|$ when $k > 2$. Hence, in general, computation on G_i' is quite lightweight.

LEMMA 2 (QUALITY OF LOCAL SHORTEST PATHS). *For any vertices u and v, let $d(u,v)$ be their shortest distance in G, and let $d_i(u,v)$ be their shortest distance in G_i'. Let $\Delta_i(u,v) = d_i(u,v) - d(u,v)$. The upper bound of $\Delta_i(u,v)$ is given as follows:*

- *Case 1: $u, v \in V_i$,*

$$\Delta_i(u,v) \leq \max\{0, d_i(u,v) - 3\}$$

- *Case 2: $u \in V_i$ and $v \in V_i' - V_i$ (or vice versa),*

$$\Delta_i(u,v) \leq \max\{0, d_i(u,v) - 2\}$$

- *Case 3: $u, v \in V_i' - V_i$,*

$$\Delta_i(u,v) \leq \max\{0, d_i(u,v) - 1\}$$

PROOF. We give the proof for Case 1 only. The other two cases can be proven in a similar way. First, we show that when $d_i(u,v) \leq 3$, we have $\Delta_i(u,v) = 0$. If $d_i(u,v) = 1$, which means u and v are adjacent in G_i', then G_i' has an exact shortest path, so $\Delta_i(u,v) = 0$. If $d_i(u,v) = 2$, we can show that $d(u,v) = 2$ must hold as well, because otherwise u, v are adjacent in G, and since both u, v are in G_i', then u, v must be adjacent in G_i' as well, which contradicts $d_i(u,v) = 2$. Thus, we have $\Delta_i(u,v) = 0$. If $d_i(u,v) = 3$, we can show that $d(u,v) = 3$ must hold as well, because otherwise we have $d(u,v) = 1$ or $d(u,v) = 2$, which can be easily shown to be contradicting $d_i(u,v) = 3$ (For example, $d(u,v) = 2$ means u, v share a common neighbor, but then this common neighbor must be in G_i', which contradicts $d_i(u,v) = 3$). Second, when $d_i(u,v) \geq 4$, it must be true that $d(u,v) \geq 3$, because the above reasoning shows that when $d(u,v) \leq 2$, we have $d_i(u,v)$ smaller than 3. This proves Case 1. □

We use an example to demonstrate Lemma 2.

EXAMPLE 2 (SHORTEST DISTANCES ON A LOCAL MACHINE). *Let us continue with the previous example. Given u, v both in V_i', we evaluate their shortest distances in G_i' and G. For $u = 1, v = 3$, we have $d_i(u,v) = 2$. From Lemma 2, we have $\Delta_i(u,v) = 0$, hence 2 is the actual shortest distance. For $u = 2, v = 7$, we have $d_i(u,v) = 3$. From Lemma 2, we have $\Delta_i(u,v) \leq 1$. Actually, there exists a 2-hop shortest path $2, 6, 7$ in G. For $u = 6, v = 7$, we have $d_i(u,v) = 4$ and $\Delta_i(u,v) \leq 3$. The actual shortest distance is 1 in G.*

Lemma 2 has the following implications:

- For many vertex pairs, their shortest distances found in an extended local graph are the exact shortest distances. Particularly, when
  - $d_i(u,v) = 2$ when at least one of u, v belongs to V_i, or
  - $d_i(u,v) \leq 3$ and $u, v \in V_i$

  we know we have found the exact shortest paths. Most real life networks are small-world networks, i.e., the average shortest distance is quite small. In most social networks, any two persons can reach each other within six steps. It is also shown that real life networks "shrink" when they grow in size, that is, their diameters become smaller [23]. Hence, short geodesic distances, such as 2 or 3, cover a significant number of vertex pairs. Thus, we already find exact shortest distances for many vertex pairs by just studying the local graphs.
- Shortest paths discovered from extended local graphs are of disparate quality. Their quality mostly depends on vertices' location in a local graph. From Lemma 2, we can see that when both u, v are in V_i, the local shortest distance $d_i(u,v)$ is of the best quality: It is either the exact shortest distance or within a small error from the exact shortest distance. In contrast, when both u and v are in $V_i' - V_i$, the approximation error may be large as in the worst case they may be adjacent to each other in G.

4.2.2 A parallel algorithm for local betweenness

We present a parallel algorithm to find the approximate betweenness (Algorithm 1). The idea is quite simple. On each machine i, for each vertex $v \in V_i$, we count the number of shortest paths in the local graph G_i' that pass through v. This number is the approximate betweenness of v.

Algorithm 1 Local betweenness on machine i

1: Randomly sample n vertices from V_i;
2: Perform BFS from each sampled vertex to generate shortest paths in G_i';
3: Select a shortest path $P(u,v)$ with probability $p_{u,v}$, which is given by Eq 10;
4: For each $v \in V_i$, compute $b(v)$, the number of shortest paths that pass through v;
5: Return $b(v)$ for each $v \in V_i$ as its local betweenness value;

There are three unique aspects of the algorithm: i) *The algorithm is based on approximate instead of exact shortest paths*. Approximate shortest paths are shortest paths discovered on extended local graphs. Furthermore, as Line 1 of Algorithm 1 shows, we only enumerate shortest paths from vertices in V_i, instead of from vertices in the larger V_i'. We justify our approach in more detail below; ii) *We do not treat approximate shortest paths equally, instead, we sample them by a probability*. We describe our probabilistic approach in more detail below; iii) *The algorithm is easily parallelizable*. Moreover, we compute betweenness on each machine without communicating with other machines. Only after all the computation is finished do we aggregate information from each machine to find the top-k vertices with the highest betweenness value.

Sampling probability. Not every approximate shortest path is of the same quality. As Lemma 2 suggests, the estimation has the highest quality for a pair of vertices if both of them are in V_i, and

the estimation can potentially have a large error if both of them are in $V_i' - V_i$. We use the upper bound in Lemma 2 to quantify the quality of a shortest path found from extended local graphs.

We define the quality of an approximate shortest path between u and v in terms of its closeness to the exact shortest path as:

$$Q(u,v) = \begin{cases} \max\{0, d_i(u,v) - 3\} & \text{if } u,v \in V_i; \\ \infty & \text{if } u,v \in V_i' - V_i; \\ \max\{0, d_i(u,v) - 2\} & \text{otherwise.} \end{cases} \quad (9)$$

A small $Q(u,v)$ implies a high quality for the approximate shortest path. In particular, $Q(u,v) = 0$ implies that the discovered shortest path is an exact shortest path. For the case where $u, v \in V_i' - V_i$, Lemma 2 shows that the error might be quite big (proportional to the exact shortest distance). Since the exact shortest distance is not known, we can choose a constant value as the default quality. In our case, we set the quality to ∞ to disqualify local shortest paths between those vertex pairs.

Given the quality function $Q(u,v)$, we can sample shortest paths by their quality. The probability is defined as:

$$p_{u,v} = \exp(-Q(u,v)) \quad (10)$$

In general, the larger the $\Delta_i(u,v)$, the less likely it is that the estimated shortest distance is the exact shortest distance. If the estimation is accurate, that is, $\Delta_i(u,v) = 0$, then we have $Q(u,v) = 0$, which leads to $p_{u,v} = 1$, meaning the shortest path will be used in the betweenness calculation for sure.

Shortest path generation from V_i vs. from V_i'. As Line 1 of Algorithm 1, only the shortest paths passing through vertices in V_i are counted. Vertices in $V_i' - V_i$ are ignored. The reason is two-fold. First, excluding $v \in V_i' - V_i$ is more efficient. Otherwise, each $v \in V_i' - V_i$ needs one communication to send its local shortest path number to the machine that holds it. As a result, overall $\sum_i |V_i' - V_i|$ communications are necessary. Second, our experimental study shows that generating from V_i is already effective enough to approximate betweenness.

5. DISTRIBUTED SHORTEST DISTANCES

After identifying landmarks, we must find the shortest distances between every vertex and every landmark. This saves a considerable amount when compared to computing the shortest distances for all the pairs, as the number of landmarks is much smaller than the total number of vertices. But since the graph is distributed, naive shortest distance algorithms still incur a large cost. In this section, we optimize the performance of shortest distance computation in a distributed environment.

5.1 Naive Level-Synchronized BFS

First, we introduce a naive *level-synchronized BFS* approach to compute shortest distances in a distributed environment. Algorithm 2 outlines the process that is carried out on each machine in a synchronous manner. Given a landmark vertex r, we find the shortest distance between r and every vertex in the graph. At level l, each machine i finds vertices of distance l to r. Then, we locate their neighboring vertices. If a neighboring vertex has not been visited before, then it has distance $l + 1$ to r. However, since the neighboring vertices are distributed across all machines, only their host machines (the machines they reside on) know their current distances to r. Thus, we ask their host machines to update the distance of these neighboring vertices. In order to save communication costs, we group these vertices by the machines they reside on and send each machine m a single message $S_{i,m}$, which contains vertices that reside on machine m. Each remote machine i, upon receiving messages $S_{1,i} \cup \cdots \cup S_{k,i}$ from all of the k machines, updates their distances to either $l + 1$ if they have not been visited before, or keeps their current distances (equal to or less than l) unchanged. After each level, all the machines synchronize, to ensure that vertices are visited level by level.

Algorithm 2 Level-synchronized BFS

Input: r;
Output: $dist[]$
1: Let V_i be the set of vertices in local machine;
2: **for all** $v \in V_i$ **do**
3: **if** $v = r$ **then** $dist[v] = 0$ **else** $dist[v] = \infty$;
4: **end for**
5: **for all** $l = 0$ to ∞ **do**
6: let U be the vertices in V_i whose distance to r is l;
7: **if** $U = \emptyset$ **then return**;
8: let S be U's neighboring vertices (both remote and local);
9: group vertices in S by their machine ID, i.e., $S = \{(m, S_{i,m})\}$ where m is a machine ID, and $S_{i,m} \subseteq S$ is the set of vertices that reside on m;
10: machine i sends $S_{i,m}$ to machine m, for each m;
11: machine i receives $S' = S_{1,i} \cup \cdots \cup S_{k,i}$ from machines $1, \cdots, k$;
12: **for all** $v \in S'$ **do**
13: **if** $dist[v] = \infty$ **then** $dist[v] \leftarrow l + 1$;
14: **end for**{synchronization;}
15: **end for**

5.2 Vertex Caching

In the above naive BFS algorithm, the cost is dominated by sending messages to remote machines to update vertices' distances to landmarks. We reduce the communication cost by caching a subset of vertices on each machine. In Algorithm 2, machine i sends message $S_{i,m}$ to machine m. Assume $u \in S_{i,m}$, i.e., machine i requests u's host machine m to update the distance of u (line 10). If this is the first time we reach u, i.e., its current distance to r is ∞ (the default value), then machine m updates u's distance to $l+1$, otherwise it ignores the request and keeps its current distance unchanged (line 13). In other words, the distance update request for vertex u is wasteful if $dist[u] \neq \infty$. Avoiding wasteful requests can reduce the communication cost. A straightforward way to do this is to cache each vertex's current distance on each machine. Then, we only need to send messages for those vertices whose current distances to r are ∞. In this manner, we can reduce the communication cost. However, for billion node graphs, it is impossible to cache every vertex on every machine. Then, the problem is: *which vertices to cache?* To answer this question, we analyze the benefits of caching.

LEMMA 3 (BENEFITS OF CACHING A SINGLE VERTEX). *Consider a level-synchronized BFS starting from a landmark node r. Let T_r be the BFS tree rooted at r. Let V_h be all the vertices of depth h in T_r (in other words, vertices in V_h will be reached at level h during the BFS). Assume vertex $v \in V(h)$. Let $N(v)$ be v's neighbors in the original graph. If v is cached, we can save up to $O(|N(v) - V_{h-1}|)$ remote requests.*

PROOF. We consider the worst case: None of v's neighbors $N(v)$ resides on the same machine as v. We will derive the upper bound of the number of remote requests we can save. Since $v \in V_h$, then v's neighbors must be in V_{h-1}, V_h, or V_{h+1}. Let $u \in N(v)$ be one of v's neighbors. We consider 3 cases for u.

1. **Case 1.** $u \in V_{h-1}$: In this case, when we access v through u, v's distance to r is still unknown (∞). Thus, we need to request v's host machine to update v's distance to r. For every v's neighbors in V_{h-1}, we need to send a request (assuming in the worst case, all of the neighbors are in different machines). Thus, we need $|N(v) \cap V_{h-1}|$ requests.
2. **Case 2.** $u \in V_h$: In this case, when we access v through u, v's distance is already updated to h during the BFS at level $h - 1$. If v is cached on each machine, we can avoid this remote request. Thus, the total number of requests we can avoid is $|N(v) \cap V_h|$.
3. **Case 3.** $u \in V_{h+1}$: For the same reason as above, we can avoid $|N(v) \cap V_{h+1}|$ requests.

Combining the 3 cases above, we can save $|N(v) \cap (V_h \cup V_{h+1})|$ requests, or equivalently, $|N(v) - V_{h-1}|$ requests. □

Since $|N(v)| = deg(v)$, where $deg(v)$ denotes v's degree, we

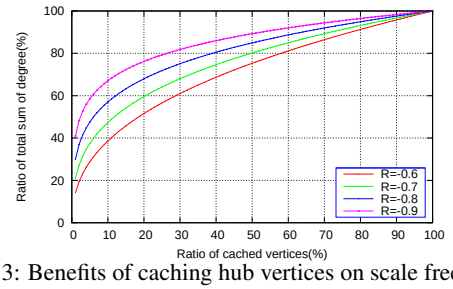


Figure 3: Benefits of caching hub vertices on scale free graphs

have $|N(v) - V_{h-1}| \leq deg(v)$. Hence, $deg(v)$ is an upper bound of the benefits of caching v.

Next, we show that in real networks where v is a *hub*, meaning v has a large degree, the upper bound $deg(v)$ is easily reached. In real networks, $deg(v)$ is positively correlated to the eccentricity of v: The larger the $deg(v)$, the smaller the maximal shortest distance from v to any other vertex. Thus, for any r, on average, a hub v has a smaller depth h in T_r than non-hub vertices, and V_{h-1} contains fewer neighbors of v. As an extreme example, when $v = r$, we have $V_{h-1} = \emptyset$ and $deg(v)$ remote accesses are avoided; when v is adjacent to r, we have $V_{h-1} = \{r\}$ and $deg(v)-1$ remote accesses are avoided. Hence, $deg(v)$ *is a good approximate for the benefit of caching a hub*.

Finally, we analyze the aggregate benefits of caching hub vertices in real graphs. Lemma 4 quantifies the benefits when we cache top-K vertices of the largest degree in a scale free graph. We use the following model for scale-free graphs [24]:

$$deg(v) \propto r(v)^R \tag{11}$$

where $r(v)$ is the degree rank of vertex v, i.e., v is the $r(v)$-th highest-degree vertex in the graph, and $R < 0$ is the rank exponent.

LEMMA 4 (BENEFITS OF CACHING HUBS). *For a graph whose degree distribution is given by Eq 11, the fraction of remote accesses that can be avoided by caching vertices of top-K maximum degree is*

$$O(\frac{\sum_{k=1}^{K} k^R}{\sum_{k=1}^{|V|} k^R}) \tag{12}$$

PROOF. By caching a vertex v, we can save at most $deg(v)$ remote accesses. Hence, we only need to count the fraction of total degree of vertices with top-K maximum degree. □

To obtain a more intuitive understanding, we give some simulation results in Figure 3 with R varying from -0.6 to -0.9 (many real networks' rank exponent is in this range). From the simulation, we can see that by caching a small number of hub vertices, a significant number of communications can be saved. For example, when $R = -0.9$, caching 20% of the top-degree vertices can reduce communications by 80%.

6. QUERY ANSWERING

Recall that in our distance oracle, each vertex is associated with a coordinate and a distance vector that contains the shortest distance to each landmark. In this section, we show how we use this information to answer shortest distance queries. The direct solution is to estimate the shortest distance using the coordinate distance. We refine the estimation using the lower and upper bounds that can be derived from the computed shortest distances.

Refined distance estimation. From the shortest distances between landmarks and each vertex, we derive lower and upper bounds (Lemma 5) for the shortest distance of any pair of vertices. These bounds can be established by the triangle inequality, and have been widely used in previous distance estimation solutions.

LEMMA 5 (LOWER AND UPPER BOUND). *Given a landmark set L, for any two vertices u, v, let $l(u,v) = \max\{|d(u,l) - d(v,l)| : l \in L\}$, and $r(u,v) = \min\{d(u,l) + d(v,l) : l \in L\}$, we have*

$$l(u,v) \leq d(u,v) \leq r(u,v)$$

Let $\bar{d}(u,v)$ be the coordinate distance between u and v. The key idea of the refined estimation is that we estimate the distance between u and v as $\bar{d}(u,v)$ only if it lies within the range of $[l(u,v), r(u,v)]$. Otherwise, we estimate the distance as $l(u,v)$ or $r(u,v)$. More specifically, given any pair of vertices u, v, the distance is estimated as

$$\hat{d}(u,v) = \begin{cases} \lfloor d^*(u,v) + 0.5 \rfloor & \text{unweighted or integer-weighted graphs} \\ d^*(u,v) & \text{otherwise} \end{cases} \tag{13}$$

where $d^*(u,v)$ is defined as:

$$d^*(u,v) = \begin{cases} \bar{d}(u,v) & \text{if } l(u,v) \leq \bar{d}_{u,v} \leq r(u,v); \\ l(u,v) & \text{if } \bar{d}_{u,v} < l(u,v); \\ r(u,v) & \text{if } \bar{d}_{u,v} > r(u,v); \end{cases} \tag{14}$$

Note that we round off $d^*(u,v)$ in Eq 13 when the graph is unweighted or integer-weighted. Because on these graphs, the shortest distances are always integers.

Compared to direct estimation by coordinate distance, the new distance estimation is more accurate. This is shown in Lemma 6, whose correctness follows directly from Lemma 5.

LEMMA 6 (ACCURACY). *For any vertex pair u, v,*

$$|d^*(u,v) - d(u,v)| \leq |\bar{d}(u,v) - d(u,v)| \tag{15}$$

Algorithm. We estimate the shortest distance between two vertices u and v as follows:

Algorithm 3 QueryAnswering(u, v)

1: If u and v are adjacent, **Return** 1;
2: If u and v have a common neighbor, **Return** 2;
3: Obtain distance vectors and coordinates of u and v from index;
4: Calculate $\bar{d}(u,v)$ by Eq. 2 according to the coordinates;
5: Calculate $l(u,v)$ and $r(u,v)$ by the distance vectors;
6: **Return** the estimated distance by Eq. 13 and Eq. 14;

Clearly, if the distance is 1 or 2, the estimation provided by the algorithm is accurate. In line 2, the cost is $O(\langle d \rangle)$ where $\langle d \rangle$ is the average degree. The estimation of the core distance has complexity $O(|L| + c)$, because we only need $O(|L|)$ time to calculate the low and upper bounds and $O(c)$ time to calculate the coordinate distance. Recall that for big graphs, we use an index to access distance vectors and coordinates on disk. Hence, for big graphs it costs an additional constant number of disk I/Os to retrieve the data.

The refined estimation comes at the cost of extra space $O(|L||V|)$ because we need to record the shortest distance between landmarks and each vertex. Real networks are small-world networks, i.e., the shortest distances are small. Usually, 4 bits are enough to store the shortest distance. Hence, such extra space is quite insignificant compared to space used to store the graph itself.

7. EXPERIMENTS

In this section, we present our experimental study. Section 7.1 presents the experiment setup. Section 7.2 compares the performance (construction and query response time) and accuracy of our approach with state-of-the-art distance oracles. Section 7.3 evaluates the effectiveness of each component of our distance oracle. Section 7.4 shows the results on billion node graphs.

7.1 Setup

We ran all the experiments on Trinity deployed on 10 machines, each of which had a 2.67 GHz dual core Intel(R) Xeon CPU and 72GB memory. We implemented our distance oracle with different configurations. We used two options for landmark selection: by

degree (DE) and by local betweenness (LB). For graph embedding, we used two error metrics: absolute error (AS) and relative error (RS) (Eq 3 and Eq 4). Hence, overall we implemented four versions of the distance oracle: DE+AS, LB+AS, DE+RS, LB+RS. For comparisons, four state-of-the-art practical distance oracles were also implemented: Sketch [5], SketchCE [25], SketchCESC [25] and TreeSketch [25]. Sketch [5] uses randomly selected vertices as landmarks, and their shortest paths to each vertex as the sketches. SketchCE [25] improves the accuracy by eliminating cycles on the shortest paths calculated by Sketch. SketchCESC [25] improves the accuracy by considering the shortcut. TreeSketch conducts BFS searches starting from the two query nodes, and then uses the shortcut between them to improve the accuracy.

Sampling strategy. We first sampled a set of vertices S ($|S| = 100$). Each vertex in S and any other vertex will be used as a query vertex pair. Hence, overall $|S||V|$ queries were sampled. This sampling method is better than the naive sampling approach. The naive method directly samples a certain number of vertex pairs and tends to miss vertex pairs with long shortest distance. Because long shortest distances in general have lower frequencies. In contrast, our sampling approach ensures sampled queries cover a wider range of shortest distances. For accuracy and query performance evaluation, we reported the average of all sampled vertex pairs.

Coverage. Recall that the goodness of a landmark set L is determined by the vertex pairs it covers. We use coverage to evaluate the effectiveness of using local betweenness for landmark selection.

$$Cov(L) = \frac{2|\{(u,v)|\exists w \in L, d(u,v) = d(u,w) + d(w,v)\}|}{|V|(|V|-1)} \quad (16)$$

The larger $Cov(L)$, the more vertex pairs L covers.

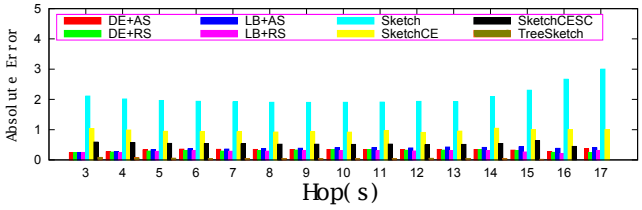
Data. We use both synthetic and real life networks for evaluation (Table 1). Real networks are typical online social networks: Orkut, Flickr, LiveJournal [26], each of which has millions of users. We also use three relatively small graphs to evaluate local betweenness and exact betweenness: CA, the collaboration network of Arxiv of Condensed Matter category (CA) [27], ST the Stanford web graph [28], and GO, the friendship network of Gowalla users [29]. We also generated two web-scale synthetic networks following the RMAT [30] model with parameters $a = 0.57, b = 0.19, c = 0.19, d = 0.05$. The first one has 0.5 billion nodes and 2 billion edges. The second one has 1 billion nodes and 4 billion edges. To the best of our knowledge, they are the largest graphs that have ever been tested for distance oracles.

7.2 Overall Performance and Accuracy

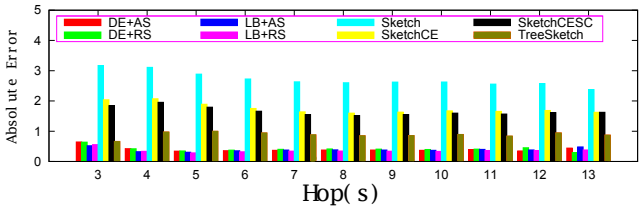
Table 2 shows the distance oracle construction time and query response time. We randomly sampled 1000 vertices to calculate the local betweenness. For each graph, 100 landmarks were selected. The construction time is acceptable in real applications. For the largest graph RMAT2, it costs a total of 36/69 hours. Since it is built as an offline phase, such cost is acceptable. Million-node graphs only take several hours. The result shows that the embedding procedure (embedding of landmarks as well as other vertices) dominates the construction time for million node graphs. For billion node graphs, the cost of BFS and local betweenness become non-negligible. We see that local betweenness was efficiently computed. For million-node graphs, it takes less time than BFS. For billion-node graphs, the computation cost is comparable to BFS. The increase of computation cost for local betweenness on large graphs is because the machine number is fixed in our experiment. As a result, the size of local graphs will increase with the entire graph size. Finally, we observe that the landmark embedding cost is constant, independent of the graph size. This is because the number of landmarks is fixed at 100 in the experiment.

Query response time is the average response time of sampled vertex pairs. The results show that our solution does not sacrifice query performance. Even for the billion-node graphs, it only costs less than 1ms. This performance is good enough to support online shortest path query.

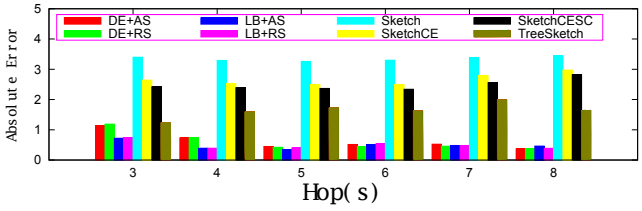
We evaluate the accuracy of our approach by comparing it with the state-of-the-art distance oracles. The results on 3 million-node graphs are shown in Figure 4. Our distance oracle is more accurate in most cases (TreeSketch produces the most accurate results on Flickr). Our distance oracles consistently produces accurate answers. For example, the absolute error of our distance oracles is almost consistently less than 0.5, while the error of TreeSketch on 2 of the 3 graphs is above 1, suggesting its sensitivity to network structures. Our distance oracles are robust under different landmark selection mechanisms or error metrics.



(a) Flickr



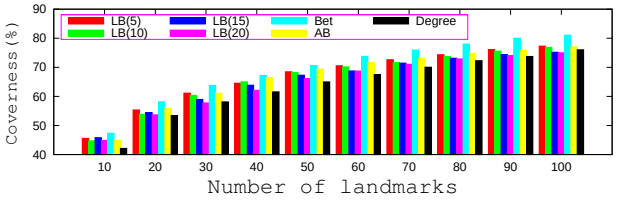
(b) LiveJournal



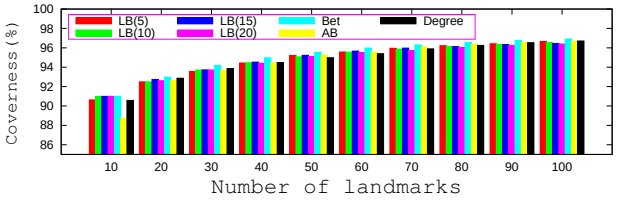
(c) Orkut

Figure 4: Accuracy on real graphs

7.3 Results of Each Component



(a) CA



(b) GO

Figure 5: Coverage as the function of #landmarks

Local betweenness. We compare the effectiveness of local betweenness (LB) with that of exact betweenness (Bet), node degree, and approximate betweenness (AB), which counts the number of shortest paths from a set of sampled vertices [31]. The computation of LB is under random partitions with varying partition numbers $\{5, 10, 15, 20\}$. The number of sampled vertices used in AB

Table 1: Graph data

Billion-node graphs	N	M	$\langle d \rangle$	Million-node graphs	N	M	$\langle d \rangle$	Small graphs	N	M	$\langle d \rangle$
RMAT1	0.5G	2G	8	Orkut	3.07M	117.19M	76.4	CA	21,363	91,342	8.6
RMAT2	1G	4G	8	Flickr	1.62M	15.75M	19.4	ST	255,265	1,941,926	15.2
				Live.	4.84M	43.36M	18.0	GO	196,591	950,327	9.7

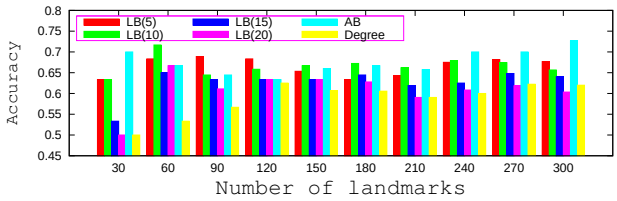
Table 2: Performance of distance oracle

Graph	Construction Time					Response Time
	total time(degree/LB)	landmark selection(degree/LB)	BFS	landmark embedding	others' embedding	
Flickr	4.03h/4.07h	1s/2.8min	33.4min	3.45h	1.6min	0.70ms
LiveJournal	4.25h/4.46h	1s/12.8min	43.8min	3.46h	3.65min	0.68ms
Orkuts	4.07h/4.36h	1s/18.1min	31.8min	3.49h	2.82min	0.69ms
RMAT1	21.08h/32.78h	28s/11.7h	11.1h	3.6h	6.38h	0.73ms
RMAT2	36.2h/69.83h	58s/33.63h	19.4h	3.73h	13.07h	0.65ms

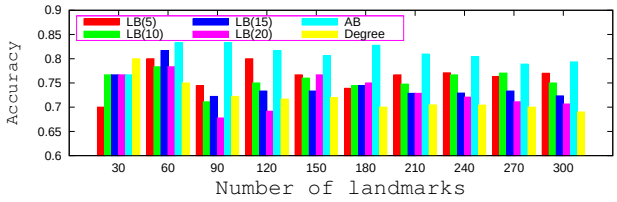
is 2000. Thus, AB needs to explore $2000|E|$ edges of the graph. In the same amount of time, we can run BFS at least 2000 times on each local graph. Hence, we sample 2000 vertices within each machine to compute LB.

We show coverage as a function of the number of landmarks in Figure 5. We can see that exact betweenness always performs the best. Local betweeness is quite close to exact betweenness, comparable to AB and node degree. We also find that the effectiveness of LB is almost independent of the number of partitions.

We further show that vertices of the top local betweenness value have significant overlap with vertices of the top exact betweenness value. Let V_1, V_2 be the top-K vertices under exact betweenness and LB. We compute the overlapping ratio as $\frac{|V_1 \cap V_2|}{|V_1|}$. The overlapping ratios between degree and exact betweenness, and that between AB and exact betweenness are also calculated. The results are shown in Figure 6. We can see that the effectiveness of LB is close to AB, and in some cases (for example in CA) is even better than AB. It is also evident that the effectiveness of LB is almost independent of the number of partitions. LB is a practical solution for landmark selection. For 120 landmarks, on CA the overlapping ratio between LB and exact betweenness is close to 70%, and on GO it is almost 80%. Such coverage and overlapping ratio are enough for selecting good landmarks.



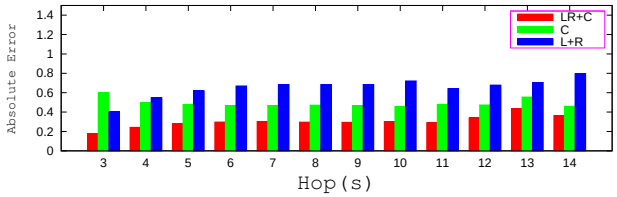
(a) CA



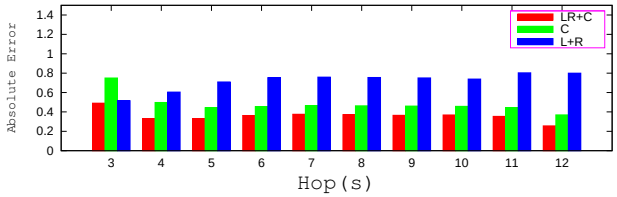
(b) GO

Figure 6: Overlapping of top-K vertices

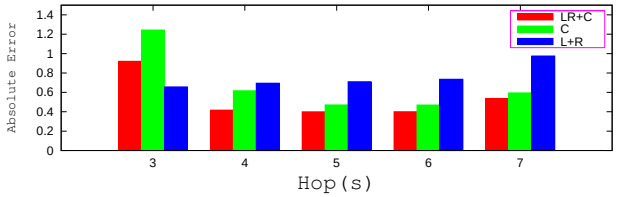
Hubs caching. Figure 7 and Figure 8 show the number of *communications* and *time* as a function of the number of *cached hubs*. We can see that the #communications is monotonically decreasing with the growth of #cached hubs. But the overall running time is not necessarily decreasing with the growth of #cached hubs. It is because when the #cached hubs increases it takes more time to query from caches. When this cost is larger than the cost saved by reducing the #communication, the overall run time will increase. This is demonstrated in the experimental results. We can see that there is a spike on Flickr, LiveJournal and RMAT when too many hubs are cached.



(a) Flickr



(b) LiveJournal



(c) Orkut

Figure 9: Accuracy on real graphs

Distance estimation. We compared different distance estimation mechanisms. The baseline approach only uses the lower and upper bounds (denoted as LR). We can either estimate according to coordinate distances only (denoted as C), or according to coordinate distances and lower/upper bounds (denoted as LR+C). Figure 9 shows the average query answering error on three real networks. It is evident that in almost all cases, LR+C had the minimal error. For LR+C, we gave the percentage of vertex pairs whose estimated distance is exactly the coordinate distance (i.e., the coordinate distance lies between the lower and upper bound). Figure 10 shows that on three real networks a significant number of quires has the coordinate distance as the answer. This justifies our motivation to construct a coordinate-based distance oracle.

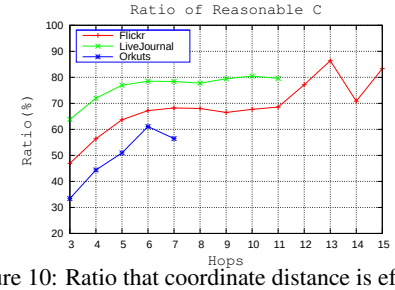
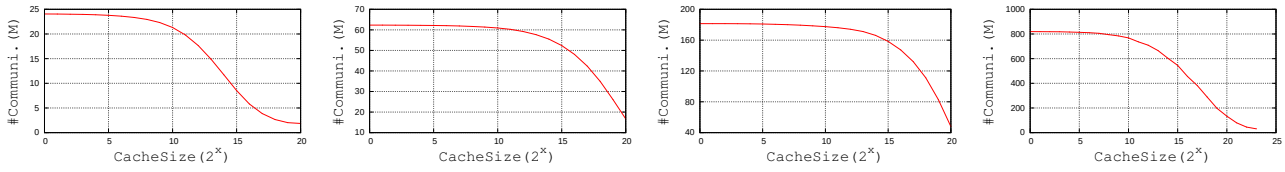


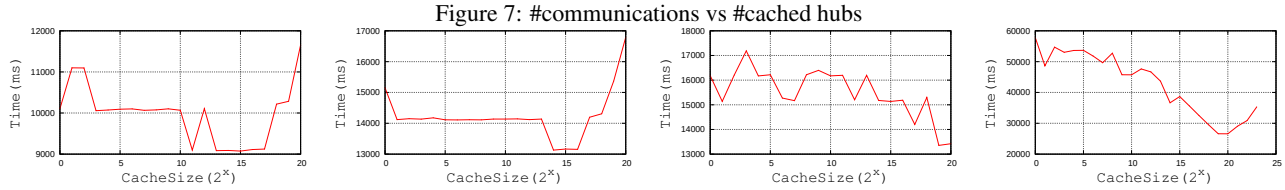
Figure 10: Ratio that coordinate distance is effective

7.4 Results on billion node graphs



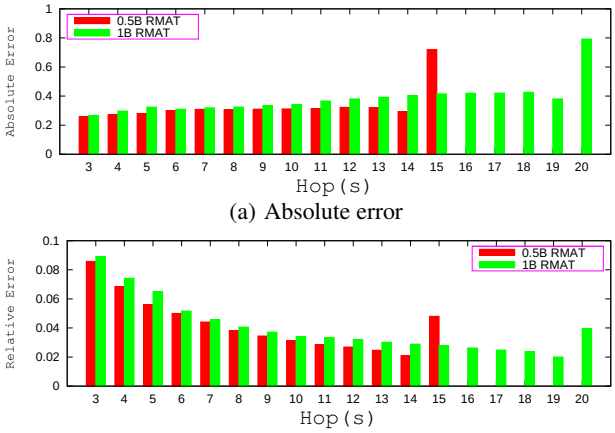
(a) Flickr (b) LiveJournal (c) Orkut (d) RMAT

Figure 7: #communications vs #cached hubs



(a) Flickr (b) LiveJournal (c) Orkut (d) RMAT

Figure 8: Time vs #cached hubs



(a) Absolute error

(b) Relative error

Figure 11: Accuracy of distance oracle on billion-node graph

We highlight again that our distance oracle can scale up to billion node graphs. As shown in Table 2, it only takes about 20 to 70 hours to construct on billion node graphs. Since it is built offline, this performance is acceptable. The query response is also quite efficient. It only takes less than 1 ms to answer the shortest distance query on a billion node graph. The accuracy results on two billion-node graphs are shown in Figure 11. We can see that for geodesic distances less than 14 (which account for the majority of the shortest paths), our distance oracles generate distances with absolute error consistently less than 0.3 and relative error consistently less than 8%.

8. RELATED WORKS

Distance oracles. Exact distance oracles in general need to store all pairwise distances, which occupy quadratic space. This gives rise to approximate distance oracles. Thorup and Zwick [32] proposed a distance oracle of $O(kn^{1+1/k})$ space, $O(kmn^{1/k})$ construction time, and $O(k)$ query complexity (with at most $2k-1$ multiplicative distance estimation). By tuning k, we find the best tradeoff between construction time, space, query time, and approximation ratio. When $k=1$, the distance oracle returns the exact distance, but it requires quadratic space, and quadratic construction time, which is prohibitive in billion-node graphs. When $k=2$, the worst distance is 3-multiplicative of the exact distance and the oracle occupies $O(n^{1.5})$ space. Many successors to this research further reduce the construction time on weighted [33] or unweighted graphs [34], or reduce the space cost on power-law graphs [35] or Erdös-Rényi random graphs [36]. These oracles are of theoretic interest and cannot be applied to billion-node graphs. Most real graphs are small-world, i.e., any vertex pair has a small shortest distance. Thus, even for $k=2$, the estimation error is unacceptable.

To handle large graphs, practical distance oracles take a sketch-based approach. There are two families of sketch-based solutions. The first [9, 25, 5, 37, 10] uses the shortest distance of vertex u to a set of landmarks as the sketch. The second [7, 6] uses the coordinate of u in certain embedded space as its sketch. Sketch based distance oracles cost linear space and return a query result in constant time but in general can hardly give a theoretic distance approximation bound. Landmark selection and distance estimation are the two key components of sketch-based solutions. These solutions differentiate from each other in these two components. Guvichev et. al [25] uses cycle elimination and tree-structured sketch to improve the distance estimation accuracy. Potamias et al. [9] shows that selecting the optimal landmark set is an NP-hard problem and shows that betweenness is a good landmark selection heuristic. Tretyakov et al. [10] uses the distance to the least common ancestor of u and v in a shortest path tree to estimate a tighter upper bound to improve the distance estimation accuracy. Das Sarma et al. [5] proposed a randomized seed selection approach to construct a sketch, which is of $O(k \log N)$ size. For distance oracles that use coordinate distances in embedded spaces, geometric space selection is critical. Euclidean space [38, 7] and hyperbolic space [39, 6] are the two widely used spaces. For Euclidean space, an upper bound of the distortion can be derived [38, 40]. But hyperbolic space is shown to be more network structure aware [39]. Zhao et al. [7, 6] proposed two solutions: Orion [7] and Rigel [6] in turn, that employ graph embedding for distance oracles. Orion uses Euclidean space and Rigel uses hyperbolic space. It was shown that Rigel is better than Orion in accuracy [6].

These distance oracles are generally not practical for billion node graphs. They either have low query accuracy, or are computationally inefficient. Furthermore, none of these solutions consider an optimized distance oracle for a distributed computing platform. As a result, the largest graphs they can handle are a web graph with 65M nodes [5] and a social network with 43M nodes [6].

Betweenness and Distributed BFS. Betweenness is widely regarded as a good heuristic to select landmarks for an effective distance oracle [9]. Betweenness was first introduced by Freeman [41]. The fastest exact algorithm to compute the betweenness for all vertices needs $O(|V||E|)$ time [16] on an unweighted graph, which is too expensive for large graphs. Many succeeding efforts focus on effective approximation or parallelized computing. Brandes et al. [31] shows that exact betweenness can be well approximated by extrapolating from shortest paths originating from a subset of k starting nodes (called pivots). It turns out that random sampling of pivots achieves good approximation [31]. Mária et al. [22] uses the shortest path with a length no larger than L for a counting in betweenness calculation. Bader et al. [42] uses adap-

tive sampling to approximate exact betweenness while reducing the computation cost. These approximate solutions do not consider parallelized optimization. Many versions of parallelized betweenness are also proposed [17, 18, 19, 20, 21] on different parallelized platforms, such as massive multithreaded computing platforms [17, 18], distributed memory systems [19], and multi-core systems [20, 21]. All these distributed betweenness algorithms only calculate *exact betweenness*, without considering how to devise an effective and approximate betweenness according to the characteristics of a corresponding distributed computing platform.

Distributed levelwise BFS has been optimized from different aspects. There are a large number of waste attempts (random access) to test wether a neighbor has been visited. Considering this, Beamer et al. [43] applies a hybrid strategy, which uses the top-down algorithm initially, and uses the bottom-up algorithm when the number of nodes on current level is sufficient large. Buluc et al. [44] developed one-dimensional and two-dimensional vertex partitioning strategies to optimize the distributed levelwise BFS. These approaches share the same levelwise BFS framework with ours, but didn't optimize distributed BFS by a smart caching strategy.

9. CONCLUSION AND DISCUSSIONS

In this paper, we introduce a novel distance oracle on billion-node graphs. This is the first research to make highly accurate shortest distance estimation possible for billion-node graphs. We solve the scalability and the accuracy problems by optimizing three key factors that affect the performance of distance oracles, namely landmark selection, distributed BFS, and distance estimation. Our experiment results on both real networks and synthetic networks show that our approach is practical for billion-node graphs, and it beats existing approaches in terms of scalability and accuracy on million-node or smaller graphs.

Extension on weighted graph. The proposed optimization techniques can also be applied on weighted graphs. The difference is that we need to calculate weighted shortest path. Bellman-Ford algorithm is a typical choice for weighted shortest path computation. It can be easily parallelized and the hub caching optimization is still effective in the parallelized Bellman-Ford algorithm. For landmark selection, shortest paths in local graphs are still good choice for the approximate betweenness. But we need to reevaluate the quality of local shortest paths, which is an interesting future work. The coordinate leaning can completely inherit the learning model on unweighted graph. In the query answering phase, we use the same strategy to generate the result. The only difference is that we can not round off the estimated distance on unweighted graph (see Eq. 13).

10. REFERENCES

[1] http://www.worldwidewebsize.com/.
[2] http://www.facebook.com/press/info.php?statistics.
[3] http://www.w3.org/.
[4] D. R. Zerbino and E. Birney, "Velvet: algorithms for de novo short read assembly using de bruijn graphs." *Genome Research*, 2008.
[5] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy, "A sketch-based distance oracle for web-scale graphs," in *WSDM '10*.
[6] X. Zhao, A. Sala, H. Zheng, and B. Y. Zhao, "Fast and scalable analysis of massive social graphs," *CoRR*, 2011.
[7] X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao, "Orion: shortest path estimation for large social graphs," in *WOSN'10*.
[8] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, 1965.
[9] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis, "Fast shortest path distance estimation in large networks," in *CIKM '09*.
[10] 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 '11*.
[11] http://research.microsoft.com/en us/projects/trinity/.
[12] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *SIGMOD '10*.
[13] T. S. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *INFOCOM'01*.
[14] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: a decentralized network coordinate system," *SIGCOMM Comput. Commun. Rev.*, 2004.
[15] Y. Shavitt and T. Tankel, "On the curvature of the internet and its usage for overlay construction and distance estimation," in *INFOCOM '04*.
[16] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, 2001.
[17] D. A. Bader and K. Madduri, "Parallel algorithms for evaluating centrality indices in real-world networks," in *ICPP '06*.
[18] K. Madduri, D. Ediger, K. Jiang, D. A. Bader, and D. Chavarria-Miranda, "A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets," in *IPDPS '09*.
[19] N. Edmonds, T. Hoefler, and A. Lumsdaine, "A space-efficient parallel algorithm for computing betweenness centrality in distributed memory," in *HIPC'10*.
[20] G. Tan, D. Tu, and N. Sun, "A parallel algorithm for computing betweenness centrality," in *ICPP '09*.
[21] D. Tu and G. Tan, "Characterizing betweenness centrality algorithm on multi-core architectures," in *ISPA'09*.
[22] M. Ercsey-Ravasz and Z. Toroczkai, "Centrality scaling in large networks," *Phys. Rev. Lett.*, 2010.
[23] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *KDD '05*.
[24] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," *SIGCOMM Comput. Commun. Rev.*, 1999.
[25] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum, "Fast and accurate estimation of shortest paths in large graphs," in *CIKM '10*.
[26] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," in *IMC'07*, 2007.
[27] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discov. Data*, 2007.
[28] J. Leskovec, K. J. Lang, A. DasGupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," 2008.
[29] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *KDD '11*.
[30] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *In SDM*, 2004.
[31] U. Brandes and C. Pich, "Centrality estimation in large networks," in *Special issue "Complex Networks' Structure and Dynamics" of the International Journal of Bifurcation and Chaos*, 2007.
[32] M. Thorup and U. Zwick, "Approximate distance oracles," 2001.
[33] S. Baswana and T. Kavitha, "Faster algorithms for approximate distance oracles and all-pairs small stretch paths," in *FOCS'06*.
[34] S. Baswana and S. Sen, "Approximate distance oracles for unweighted graphs in expected o(n2) time," *ACM Trans. Algorithms*, 2006.
[35] W. Chen, C. Sommer, S.-H. Teng, and Y. Wang, "Compact routing in power-law graphs," in *DISC'09*.
[36] M. Enachescu, M. Wang, and A. Goel, "Reducing maximum stretch in compact routing." in *INFOCOM'08*.
[37] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *SODA '05*.
[38] S. Rao, "Small distortion and volume preserving embeddings for planar and euclidean metrics," in *Symposium on Computational Geometry*, 1999.
[39] Y. Shavitt and T. Tankel, "Hyperbolic embedding of internet graph for distance estimation and overlay construction," *IEEE/ACM Trans. Netw.*, 2008.
[40] J. R. Lee, "Volume distortion for subsets of euclidean spaces," *Discrete & Computational Geometry*, 2009.
[41] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, 1977.
[42] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail, "Approximating betweenness centrality," in *WAW'07*.
[43] S. Beamer, K. Asanovi?, and D. A. Patterson, "Searching for a parent instead of fighting over children: A fast breadth-first search implementation for graph500," Tech. Rep. UCB/EECS-2011-117.
[44] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *SC2011*, ser. SC '11, 2011.