Decentralized Search for Shortest Path Approximation in Large-scale Complex Networks

ABSTRACT

Finding approximate shortest paths for extremely large-scale complex networks is still a challenging problem, where existing work requires too much overhead to achieve accurate results and good path diversity for graphs with hundreds of millions of edges. In this paper, we develop a online search method based on preprocessed indexes, to approximate shortest path of arbitrary pairs of vertices. We demonstrate that the algorithm achieves much higher accuracy and path diversity with less index overhead compared to previous work. The algorithm can also achieve differentiated level of accuracies by dynamically controlling the search space to meet various application needs. In order to perform decentralized search more efficiently, we propose a new heuristic index construction algorithm that can greatly increase the approximation accuracy. We further develop support for parallel searches to further reduce the average search time for large amount of queries. We implement our algorithm in distributed settings to deal with extreme size graphs. Our algorithm can handle graphs with billions of edges and perform searches for millions of queries in parallel. We evaluate our algorithm on various real world graphs from different disciplines.

CCS Concepts

•**Information systems** → **Data structures;** *Data mining;*

Keywords

Graphs, Shortest Paths, Decentralized Search

1. INTRODUCTION

Various types of graphs are commonly used as models for real-world phenomenon, such as online social networks, biological networks, the world wide web, among others. As their sizes keep increasing, scaling up algorithms to handle extreme size graphs with billions of vertices and edges remains a challenge that has drawn increased attention in recent years. Specifically, straightforward operations in graph theory are usually too slow or costly when they are applied to graphs at this scale. One problem that has drawn a lot of attention in the past is finding shortest paths in the network. This operation serves as the building block for many other tasks. For example, a natural application for road network is providing driving directions [1]. In social networks, such applications include social sensitive search [25], analyzing influential people [12], among others. Estimating minimum round trip time between hosts without direct measurement is another application in technology networks [22].

Although previous works have studied shortest path problem on large road networks extensively and have very effective approaches, a large category of networks known as the complex networks has very different structures. Approaches that worked on road networks do not perform well on complex networks, as the latter follow power law degree distributions and small diameters. In this paper, we are focusing on the shortest path problem for complex networks in particular, as their extreme sizes and unique topologies make the problem particularly challenging.

Our design is motivated by recent studies that combine both offline processing and online queries [16, 24, 3, 17, 11]. In these methods, the step of preprocessing aims to construct indexes for the networks, which are later used in the online query phase to dramatically reduce the query time. Among these approaches, landmark based algorithms [23, 8, 16, 10, 24, 17]are widely used for approximate shortest path/distance between vertices. Usually such algorithms select a small set of landmarks, and construct an index that consists of labels for each vertex, that store distances or shortest paths to landmarks. The approximation accuracy of landmark based algorithms heavily depends on the number of landmarks. Usually to achieve high accuracy, a relatively large set of landmarks is required, which lead to large preprocessing overheads. Indexes that can answer path queries usually have much larger space overhead than indexes that can only answer distance queries. One goal of our design, therefore, is to provide accurate results while still maintain low overhead for indexing.

Previous works that combie online search with index only explore short cuts among vertices between label of source and target vertices [10, 17]. The accuracy and path diversity are rather limited this way. Instead of estimating shortest path solely by examining vertices contained in labels of source and target vertices, our algorithm performs a

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

heuristic search on each vertex it visited and pick the vertex to visit in next step by the distance information gathered from neighbor vertices of current step. Our approach is able to explore edges that have not been indexed and examine vertices that does not included in labels of source and target vertices to achieve higher accuracy and path diversity with limited index size. The heuristic search that we use is called decentralized search which was introduced by [13]. Here the "'decentralized"' means that the decision at each step is made based solely on local information which, in our context, is the labels of neighbor vertices.

Decentralized search can expand its search space to balance between different level of performance, i.e. accuracy and path diversity, and resources required for each search. The search is very versatile to meet various application needs without redoing the preprocessing.

The performance of decentralized search relies heavily on the index. To achieve better accuracy without increase index overhead, we introduce a heuristic index construction algorithm to control shortest paths to be indexed during preprocessing.

We implement decentralized search on a distributed platform to support large scale graph with billions of edges. To take advantage of resources available for a cluster of machines, our algorithm can support large amount of queries to run in parallel. There are two properties of decentralized search which makes it very suitable for parallel processing. First, decentralized search is very light-weighted in terms of space complexity and communication complexity. The search does not need to store any information on a per vertex basis like BFS or A* search, very limited space overhead is required for each search. Second, decentralized search does not have data dependency on the index and underlying graph. Multiple search can run independently on the same graph and index. These properties makes it possible for large number of searches running in parallel efficiently without reaching the physical limit of machines, i.e. memory limit or network bandwidth. For example, in our experiments, we showed that millions of decentralized search can run in parallel on graphs with billions of edges on a cluster of commodity machines, and finish in tens of seconds.

Based on our algorithm design, we further develop a query-processing system based on distributed cloud infrastructure. In this platform, users first submit their graphs for preprocessing needs. The graph processing engine will assign resources according to application's need for accuracy and construct an index for the input graph. Later, users may submit large volumes of queries repeatedly, for which responses will be generated. The light-weighted decentralized search allows a large amount of queries to run in parallel so that queries can be answered in a timely manner. Applications that generate queries (on the client side) can provide their desired accuracy levels and the graph processing engine can dynamically adjust search space of decentralized search to meet differentiated levels of accuracies.

1.1 Contributions

Our contributions can be summarized as follows:

- We propose index guided decentralized search for shortest path approximation;
- We design a heuristic index construction algorithm that can improve accuracy of the decentralized search without increasing preprocessing or online search overheads.
- We achieve efficient query processing and good scalability with implementations in distributed settings and parallel processing.
- Experiments on various complex networks demonstrate that the proposed algorithm is promising in approximating shortest path compared to existing works.

The rest of this paper is organized as follows. In Section 3, we give notations and definitions used in this paper. We explain decentralized search for shortest path approximation in Section 4. Section 5 shows our index construction algorithm. In Section 6 we show details on our distributed implementation. And we show the evaluations of our algorithm in Section 7. In Section 2 we described previous works on exact and approximate approaches. We conclude our work in Section 8.

2. RELATED WORKS

Existing work on shortest path/distance can be classified into exact approaches and approximate approaches.

Exact Approaches: Majority of the exact approaches are based on either 2-hop cover [5], [2] or tree decomposition [3], [26]. For the former one, finding optimal 2-hop covers is a challenging problem. [2] takes a different approach that solving 2-hop cover problem with graph traversals which has better scalability. [11] borrowed the highway concept from shortest path algorithms on road networks and construct a spanning tree as a "highway". [7] introduced an effective disk-based label indexing method based on independent set.

Approximate Approaches: Since the exact approaches do not scale very well, approximate algorithms are also well studied for large-scale complex networks. Landmark based algorithms are extensively studied for approximate shortest path/distance [23], [8], [16], [6], [15]. Although theoretical study of such algorithms does not reveal promising results [23], they usually work well in practice. [16], [21] studied various landmark selection strategies for constructing better indexes. A common problem for distance-only indexes is that they do not perform well for close pairs of vertices [3]. Algorithms [10], [24], [17] which index shortest paths are proposed to alleviate this problem. Beside landmark based approaches, there are several other approximate approaches. [4] forms the shortest path problem as a learning problem to predicting pairwise distance. [27] maps vertices to low-dimension Euclidean coordinate spaces to answer distance queries in constant time.

Combining online search with indexes: [8] uses A* search for online query based on indexes constructed by landmark based algorithms. However, the cost of each A* search is still very high for large scale networks. [10] perform BFS on a sub-graph generated by the labels of source and target vertex. Although search space is greatly reduced, it still needs around seconds to handle graphs with millions of vertices.

3. PRELIMINARIES

3.1 Notations

In our problem, we consider a graph $G = (V, E)$. For a source node s and a target node t, we are interested in finding a path $p(s,t) = (s, v_1, v_2, ..., v_{l-1}, t)$ with a length of $|p(s,t)|$ close to the exact distance between s and t in the graph. Let $P(s,t)$ be the set of all paths from s to t. The distance between s and t is $d_G(s,t) = min_{p(s,t)\in P(s,t)}|p(s,t)|$.

We will focus on unweighted, undirected graphs in this paper.

3.2 Landmark based indexes

Our method is motivated by the idea of using landmarks as the basis for indexes. Specifically, given a graph G and a small (constant) set of landmarks L, $|L|$ BFS traversals are needed to precompute the index. An index for the graph contains a label $L(v)$ for each vertex which store the shortest path to each landmark $(l_i, p(v, l_i))$ where $|p(v, l_i)| = d_G(v, l_i)$.

3.3 The least common ancestor distance

The least common ancestor of two vertex in a tree is the deepest vertex that contain both vertices as descendants, we denote it as $c_l(s,t)$. The shortest path distance satisfies the triangle inequality, for an arbitrary pair of vertices s and t, we have the following bound:

$$d_G(s,t) \leq min_{l\in L}\{d_G(s, c_l(s,t)) + d_G(c_l(s,t), t)\} \quad (1)$$

This upper bound, which we refer to it as LCA distance and denoted by $d_{LCA}(s,t)$, can be used as an approximation of the distance from s and t. And we denote the path indicated by this distance as $p_{LCA}(s,t)$.

4. DECENTRALIZED SEARCH FOR SHORTEST PATH APPROXIMATION

We propose to solve the point-to-point shortest path estimation problem using decentralized search with landmark based index. This section explain how decentralized search work with the index and underlying graph. Several aspects of the search including termination criterion, bidirectional search and tie breaking strategy are also discussed.

4.1 Index guided decentralized search

We perform decentralized search on an indexed graph as follows. For a given pair of source and target vertex, the search start at the source vertex append it to the approximated path. The search examines each neighbor of vertex currently visited, for each neighbor, the LCA distance to the target vertex is calculated. The neighbor with least LCA distance will be picked as the vertex to visit next and append it to the approximated path. The search continues until it reach the target vertex.

However, terminating when the search reaches the target vertex is a valid but not an ideal terminate criteria. Since the label of each vertex stores the shortest path of each vertex to each landmark. And shortest path follows the optimal substructure, i.e. the path between any two vertices along the shortest path is also the shortest path of them. So a search can stop once it reaches any vertex in the label of the target vertex. The path contained in the label of target vertex is already the the shortest path due to the optimal substructure, we can directly concatenate it to the visited vertices to form a approximated path.

The detailed algorithm of decentralized search is depicted in 1.

Algorithm 1 Algorithm decentralized search

```
function DecentralizedSearch(G, s, t)
    p ← ∅
    u ← s
    p = p ∪ u
    while u ∉ L(t) do
        d_min ← ∞
        w ← u
        for each v adjecent to u do
            if d_LCA(v, t) < d_min then
                d_min ← d_LCA(V, t)
                w ← v
        u ← w
        p = p ∪ u
    p_remain ← path from u to t in L(t) exclude u
    p = p ∪ p_remain
    return p
```

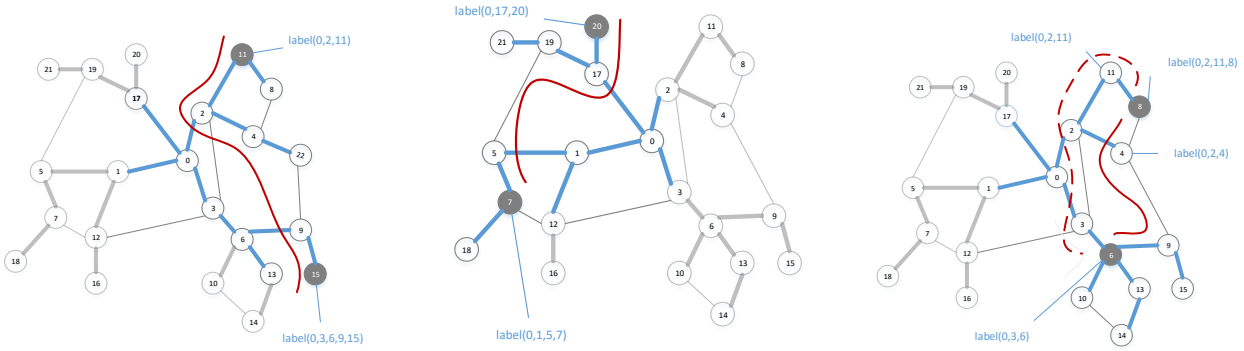
At each step, by examining neighbor vertices the search is able to explore large part of the graph that is not included in the label of source and target vertex, which can potentially increase both accuracy and diversity of the path being found. For example in Fig. **??** from 15 to 11, by following the procedure of decentralized search, instead of finding a short cut edge with both ends in labels of vertex 15: $(0, 3, 6, 9, 15)$ and 11: $(0, 2, 11)$, decentralized search can find a edge $(9, 4)$ that can lead to a shorter path which is denoted by solid curved line.

So will the index guided decentralized search eventually terminate? As long as the source vertex s and target vertex t are reachable from each other, decentralized search will terminate in as much as σ_{max} steps, where σ_{max} is the diameter of the graph. To see this, first, the distance from s to any vertex in $L(v)$ is shorter than σ_{max}. And at each step, suppose decentralized search is visiting vertex u, there must be a neighbor vertex v that is on the path indicated by LCA computation of u and t that meet $d_{LCA}(v,t) \leq d_{LCA}(u,t) - 1$. Since decentralized search always pick the neighbor with least LCA distance to the target, the LCA distance to the target at each step will decrease at least by 1. Therefore, decentralized search will terminate in at most σ_{max} steps.

The time complexity of Decentralized search depends on the max degree and the diameter of the graph. Decentralized search take at most σ_{max} steps to finish. For each step, the search need to check at most δ_{max} neighbor vertices, where δ_{max} is the max vertex degree of the graph. For each neighbor, k LCA computations are required where k is the number of landmarks. The compuational time required for each LCA computation is $O(h)$ where h is the height of the indexed shortest path tree. And we have $h \leq \sigma_{max}$ So the worst case time complexity of decentralized search is $O(k{\sigma_{max}}^2\delta_{max})$.

The space complexity for decentralized search contains two parts, offline index space complexity and online query space complexity. The space required for offline index is $O(k\sigma_{max}n)$. For each query, $O(k\sigma_{max})$ space is required to store the labels of target vertex and the vertex that is being examined, $O(\sigma_{max})$ space is required to store the approximated path. Combining them together, the online search space complexity of decentralized search is $O(k\sigma_{max})$.

4.2 Bi-directional search



(a) Decentralized search (b) Bi-directional Decentralized Search (c) Tie Strategy

Figure 1: Cumulative distribution of the number of terminal nodes sampled by different algorithms.

In this section we show how to apply the idea of bidirectional search to decentralized search. In bidirectional decentralized search, the backward search starting at target vertex with a goal to reach the source vertex. Driven by a different goal and launched at a different start point, the backward search may have a different search space compared to the forward search. It is not guaranteed that the forward search and backward search will eventually meet at any vertex except the source and target due to this difference. Actually, in decentralized search, the forward search and backward search are more like two independent search, that only resulting paths are required to be combined.

The main purpose for performing a backward search is to increase the possibility of finding a shorter approximate path and increase path diversity by exploring a different search space. This, however, is quiet different from the application of bidirectional search in BFS or A* search where the main focus is to reduce search space. For example in Fig. **??**, the search starts from 20 to 7 can find a shorter path $p = (20, 17, 19, 5, 7)$ than the search starts at 7. Due to that 0 has a smaller LCA distance to 7 than 19, the edge $(19, 5)$ cannot be found by the search starts from 20.

4.3 Handle ties

Tie happens frequently in decentralized search, especially when the number of landmarks is small. A tie here means during decentralized search, there are multiple neighbors of a visited vertex that have same LCA distance to the target. The reason for tie is that there is not sufficient information in the index that can separate some vertices for a query. For example in Fig. **??**, to find path from 8 to 6, when traversing neighbors of vertex 8, both vertex 11 and 4 have the same LCA distance to vertex 6, but their actual distances to vertex 6 are different due to edges currently invisible to the decentralized search. Note that tie is always query-dependent, different queries may have very different tie patterns even if the related searches examine similar search spaces.

Expanding the search onto each tie vertex require the search examine different sets of vertices and edges which increase the cost of the search, but also increase the chance to find a shorter path as well. Two obvious solution to deal with ties are either randomly pick one vertex to visit or visit all vertices in the next step. The former one incur no additional cost and will have the least possibility to find a shorter path, we refer to it as single branch decentralized search. The latter one require most effort and will lead to the shortest path the decentralized search could possibly find, we refer to it as full branch decentralized search.

[k-bounded tie, may add later if time allows]

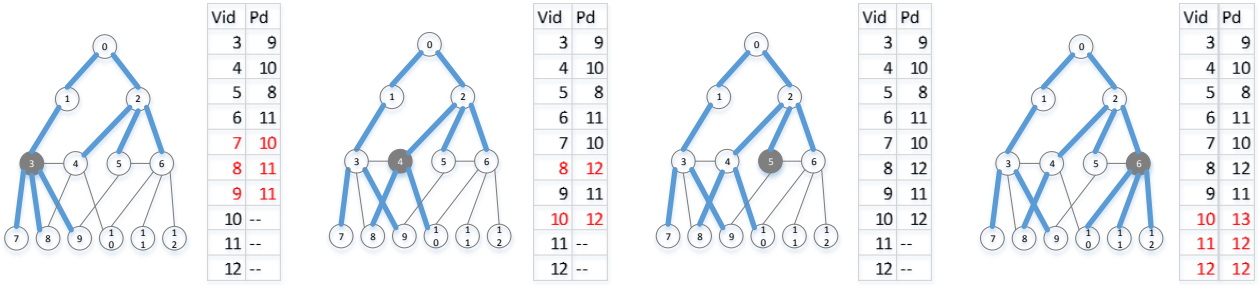
5. INDEX CONSTRUCTION

This section describes our index construction algorithm. Previous works have studied various landmark selection strategies which have a significant impact on the accuracy of online query. In our study, we observe that even with the same landmark set, choosing which shortest path from vertex to landmark to be indexed also plays an important role for the accuracy of online search.

5.1 Greedy index construction algorithm

On the core of decentralized search is to iteratively find vertices that share the least common ancestor with target vertex at a higher level of the indexed shortest path tree. From the point of view of a vertex, a good shortest path from each landmark to be indexed should be the one that intersects with most of other shortest paths. As the more intersects with other shortest paths, the higher the chance there exist a LCA at a higher level for average cases. With this intuition, we design our heuristic greedy index construction algorithm to index the shortest path with highest "centrality", i.e. intersects with most other shortest paths. To represent the "centrality" of a shortest path, we use the sum of vertex centrality along the path. Betweenness centrality fits our needs very well as it directly represent number of shortest paths each vertex on, but the computation complexity to even estimate the value for every vertex in a graph is still high [19]. So we use degree as an alternative and refer the sum of degree of vertices along a path as path degree, denoted by Pd.

Our greedy index construction algorithm, which greedily index shortest paths with highest path degree, can be easily modified from the regular index construction procedure. Note that path degree of shortest path follows optimal substructure, i.e. if $(u, .., w, ..., v)$ has the highest path degree among all the shortest path from u to v, then the path de-



Vid	Pd
3	9
4	10
5	8
6	11
7	10
8	11
9	11
10	--
11	--
12	--

(a) Traversing 3: 7, 8, 9 stores the shortest path as they have been reached for first time.

Vid	Pd
3	9
4	10
5	8
6	11
7	10
8	12
9	11
10	12
11	--
12	--

(b) Traversing 4: 8 updates the label for larger path degree. 10 stores shortest path as it's been reached for first time.

Vid	Pd
3	9
4	10
5	8
6	11
7	10
8	12
9	11
10	12
11	--
12	--

(c) Traversing 5: 9 does not update its label since the path degree of shortest path through 5 is smaller.

Vid	Pd
3	9
4	10
5	8
6	11
7	10
8	12
9	11
10	13
11	12
12	12

(b) Traversing 6: 10 updates the label for larger path degree. 11, 12 stores shortest path for first time.

Figure 2: Greedy algorithm that index shortest path with highest path degree during breadth first search

gree of $(u, ..., w)$ is also the highest among all the shortest path from u to w. Normally, the index is constructed with a BFS and a label is assigned to each node the first time the search reach it. To modify it, we first need to add a variable to cache the path degree Pd of the shortest path stored in the label of each vertex. Then during BFS traversal, suppose the search is visiting vertex u and reach its neighbor v. If $L(v)$ is not empty, and v is on a higher level of BFS tree than u, then a label update is performed. If $Pd(u) + v.degree > Pd(v)$, then the label and related path degree of v are updated. Fig. 2 shows an example of how to greedily select shortest path with the highest path degree during breadth first search. When traversing vertex 4, even though vertex 8 has already been indexed with a shortest path $(0, 1, 3, 8)$ into its label, due to that $(0, 2, 4, 8)$ has a higher path degree, the label of vertex 8 is updated. The same thing happens to vertex 10 while traversing vertex 6.

The detailed algorithm of greedy index construction is depicted in 1.

Algorithm 2 Algorithm greedy index construction vertex program running on u

```
function INDEX CONSTRUCTION(root)
    PD ← ∅
    L ← ∅
    Q ← ∅
    L[root.id] = root.id
    PD[root.id] = root.degree
    Q.push(root)
    while Q ≠ ∅ do
        u = Q.pop()
        for eachvadjecenttou do
            if v.id ∉ L then
                L[v.id] = L[u.id] ∪ v.id
                PD[v.id] = PD[u.id] + u.degree
                Q.push(v)
            else if L(v).size() < L(u).size() + 1 then
                continue
            else if PD[v.id] < PD[u.id] + u.degree then
                L[v.id] = L[u.id] ∪ v.id
                PD[v.id] = PD[u.id] + u.degree
    return L
```

6. DISTRIBUTED IMPLEMENTATIONS

To handle extremely large graphs and large amount of queries, we implement our algorithm in distributed settings. Since decentralized search has low online search space complexity and does not have data dependency upon each other, it is well suitable to run in a parallel way. In this section, we discuss how to implement decentralized search on a distributed general graph processing platform, Powergraph[9].

6.1 Decentralized search vertex-program

An overview of our shortest path query processing system is shown in 3. The system first partition the graph with Powergraph onto multiple machines. Then several BFSs are performed to construct the index. After the index is built, multiple shortest path queries can run in parallel with decentralized search. Large volumes of queries can submit repeatedly, for which responses will be generated.

Decentralized search can be easily implemented as vertex-program in appliance to Powergraph's Gather-Apply-Scatter model. Each search instance contains approximated path and label of target vertex, as index is stored distributively and label of target vertex may not be accessible on each machine locally. In Gather phase, the search aim to find the neighbor with shortest LCA distance to the target. For each query, LCA distance d_LCA calculated from labels L of each neighbor and target vertex is collected and accumulated by finding the neighbor with the smallest LCA distance, A next hop candidate is returned after this phase. In apply phase, the program appends the candidate to the approximated path $p_{appr.}$ and and check whether the terminate criteria is met for each query. If so, the result path will be recorded and the query will be terminated. Otherwise, program will proceed to Scatter phase to start new vertex-programs on the candidate vertex and pass on the query instance to them. Algorithm 3 shows the detailed algorithm.

There are two thing to notice for the decentralized search vertex program. First, the search only update the approximated path $p_{appr.}$. There is no data dependency among multiple searches except when in the last results are stored to the same container for each machine. Second, the communication for decentralized search can happen during gather and scatter phase. In gather phase, the label of target ver-

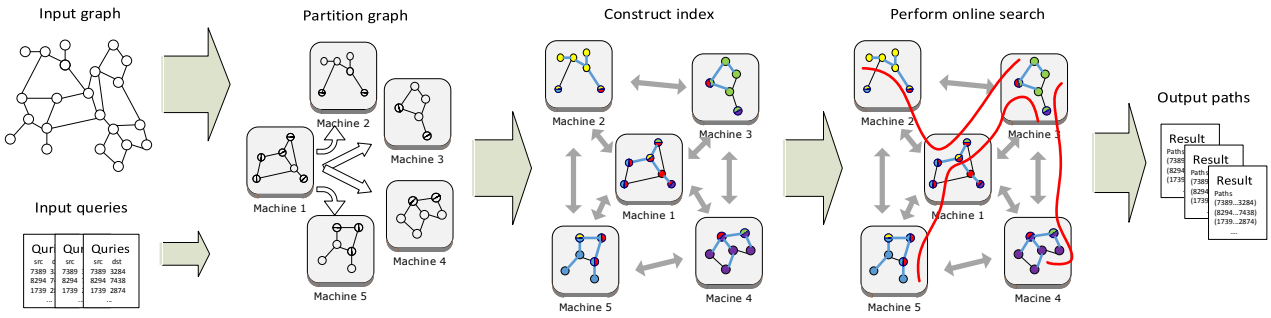


Figure 3: A system overview

Algorithm 3 Algorithm decentralized search vertex program running on u

function GATHER($L(v)$, $L(t)$)
  return $d_LCA(v,t)$, $v.id$
function SUM($d_LCA(v_1,t)$, $vid1$, $d_LCA(v_2,t)$, $vid2$)
  return $min(d_LCA(v_1,t), d_LCA(v_2,t)$, related vid
function APPLY($p_{appr.}(s,t)$, $d_LCA(v,t)$)
  $p_{appr.}(s,t) += vid$
  if terminate criteria is met **then**
    store $p_{appr.}(s,t)$
    $term$ = true
  else
    $term$ = false
function SCATTER($p_{appr.}(s,t)$, $term$)
  if $\neg term$ **then**
    Activate(v, $p_{appr.}(s,t)$)

tex need to be passed to multiple machines, and the size is $O(k\sigma_{max})$. And each gather function return a d_LCA along with its id that has $O(1)$ size. So for each query, only $(O(k\sigma_{max}) + O(1)) * M$ size of data is transferred where M is the number of machines. In scatter phase, communication happens when a vertex is chosen as next hop and need to be activated, the whole search instance, including approximated path and label of target vertex need be transmitted to the new vertex program. For each query, $O(k\sigma_{max}) + O(\sigma_{max})$ size of data may be transferred.

The low memory and communication cost, along side with light data dependency makes multiple decentralized search instance very suitable to run in parallel. To modify the vertex program for parallel processing, each vertex program maintain a list of search instance. During gather phase, label of target vertex for each query is transmitted to other machines, and d_LCA is calculated for each query. Each query is updated during apply phase. In scatter phase, each query is examined for whether activating a certain vertex or not.

6.2 Distributed tie breaking strategy

If a tie happens, according to tie strategy, multiple neighbors may be returned as candidates at gather phase. Then at apply phase, the search instance will be fork itself into multiple search instance, each for a single candidate. The problem here is, not like centralized version, during the future computation, even a search instance find a shorter path than others, it cannot terminate other searches as such synchronization is quite costly in distributed settings. With this limitation, the search space are quite difficult to control and the search may end up with excessive number of child search instances. So in our implementation, at each step, only one candidate will be chosen as a "'main"' candidate. For candidates that are not "'main"' candidates, an extra terminate criteria will be applied. At the next step, if the search cannot find a shorter path than expected, i.e. $|p_{appr.}| + d_LCA$, then the search will stop, and no result will be recorded. This can control the search space effectively without compromise accuracy too much.

6.3 Prune LCA computation

It is possible to prune number of LCA computation required at each step for decentralized search. Suppose the search is visiting vertex u, which means the d_LCAu, t has already been calculated in previous step. When traverse u's neighbors, if a neighbor v is a child on the indexed shortest path tree T_i, then the LCA computation for v and t on that tree T_i does not need to be performed as it is certain that $d_LCA(i)v, t > d_LCA(i)u, t$. In practice, such way to pruning can reduce half of the total number of LCA computations of a single search on average.

7. EVALUATIONS

In this section, we show the results of experimental evaluation of decentralized search. We first give an overview of the datasets and introduce the experiment settings of our evaluations. The quality of approximated path generated by decentralized search is evaluated from two aspects, distance accuracy and path diversity, in 7.3 and 7.4 respectively. We then show both overhead of index and online search in 7.5. In the last, we study the scalability of our system in 7.6.

7.1 Datasets

We evaluate our algorithm on 8 graphs from different disciplines as shown in table 1 in ascending order on the number of edges. All graphs are complex networks that have power-law degree distribution and relatively small diameter. All datasets have at least millions of edges, the largest one has almost two billion edges. To simplify our experiments, we treat each graph as undirected, un-weighted graphs. We

Table 1: Datasets

Dataset	Type	$\|V_{wcc}\|$	$\|E_{wcc}\|$	$\overline{\sigma}$
Wiki	Communication	2.4M	4.7M	3.9
Skitter	Internet	1.7M	11.1M	5.07
Livejournal	Social	4.8M	43.4M	5.6
Hollywood	Collaboration	1.1M	56.3M	3.83
Orkut	Social	3M	117M	4.21
Sinaweibo	Social	58.7M	261.3M	4.15
Webuk	Web	39.3M	796.4M	7.45
Friendster	Social	65M	1.8B	5.03

Datasets with number of vertices and edges in the largest weakly connected components, and average shortest distance $\overline{\sigma}$ of 100,000 vertex pairs.

only use the largest weakly connected component of each graph which consist of more than 90% of vertices for all graphs. All datasets are collected from [14] and [20].

7.2 Experiment settings

We evaluate our algorithms in both distributed settings and centralized settings. For distributed settings, we use 20 Amazon EC2 m4.xlarge virtual machines. Each virtual machine has 4 vCPUs and 16 GB memory. We also implemented a centralized version on a Cloudlab [18] c8220 server with two 10-core 2.2GHz E5-2660 processors and 256GB memory. Powergraph [9] is the platform for distributed version and Snap [14] is the platform for centralized version. All algorithms are implemented in C++.

All the evaluations use the same landmark selection strategy, a variation based on $DEGREE/h$ strategy from reference [16] which take both vertex degree and distance to existing landmarks into consideration. On selecting a new landmark, each vertex receive a rank which is the product of its degree and the sum of distance to all the existing landmarks. The vertex with highest rank will be added to the landmark set. We denote number of landmarks as k.

Quries in our experiments are randomly generated. For accuracy evaluations, since performing BFS on large graphs in our datasets is extremely slow, to control the number of exact pairs of shortest path distance we need to find using BFS, we randomly choose 1,000 vertices of each graph as source vertices. For each source vertex, we then randomly choose 100 vertices as target vertices. We are able to generate 100,000 queries with 1000 BFS. All results in 7.3 are averaged among 100,000 queries.

7.3 Approximation Accuracy

We first evaluate the approximation accuracy of decentralized search. We use the average approximation error as the measure of accuracy which is defined as follows:

$$E_{p_{appr.}(s,t)} = \frac{|p_{appr.}(s,t)| - d_G(s,t)}{d_G(s,t)}$$

We show the results of 4 variants of decentralized search with mixture of different tie strategy and index construction approach. All the decentralized search are performed with bidirectional search. We also list the performance of an state-of-the-art existing method, TreeSketch, from reference [10]. [Reference [17] also includes a online search method but is quite similar to TreeSketch in nature so that we haven't included here.] We perform all 5 approaches for 100,000 queries under various size of landmark sets and show the averaged results in Fig. 4.

We can see in Fig. 4 that decentralized search achieves better accuracy in most of cases. Especially with small landmark sets, i.e. $k < 5$, decentralized search outperforms TreeSketch on all the graphs. And Full branch decentralized search outperforms TreeSketch with any landmark sets on all eight graphs. When the search are carried on the index constructed by our greedy heuristic, the performance gain is much noticeable, with 1.76 to 8.09 times lower average error ratio for 1 landmark and 1.25 to 2.59 times lower average error ratio for 20 landmarks on various graphs.

For tie breaking strategies, full branch tie strategy always outperforms single branch tie strategy with large margins as it explores more vertices, that of course, comes with larger search overheads that will be discussed in 7.5. The average error ratio of full branch is lower than 1.21 to 1.84 times of single branch with 1 landmark and 1.24 to 2.61 times with 20 landmarks on various graphs. As the number of landmark increases, the performance gain also increase. The single branch and full branch set the accuracy range that is achievable by Decentralized search.

Decentralized search carried on index constructed by greedy heuristic has much lower error ratio than decentralized search with regular index. The average error ratio of decentralized search is 1.17 to 2.51 times lower for single branch and 1.27 to 2.73 times lower for full branch for 1 landmark than decentralized search based on regular index. For higher landmark size, regular index actually outperforms index constructed by greedy heuristic in some cases, the reason should be that the greedy heuristic may introduce higher redundancy, i.e. multiple landmark generate similar index due to same goal, when number of landmark increases. The ratio ranging from 0.77 to 1.41 for single branch and 0.57 to 1.33 for full branch under 20 landmarks.

Table 2: Path diversity (k = 2)

Graph	Path cnt(DS)	Path cnt(TS)	Out ratio
Wiki	28.9	1.9	0.372
Skitter	24.1	2.4	0.418
Livejournal	30.8	1.9	0.338
Hollywood	9.9	2.6	0.471
Orkut	19.2	3.2	0.465
Sinaweibo	32.0	3.0	0.301
Webuk	704.1	2.0	0.501
Friendster	16.8	2.8	0.39

7.4 Path Diversity

We evaluate the path diversity of paths returned by decentralized search in this section. By forking the search into multiple search during tie breaking, the algorithm can return various paths with same length, i.e. the estimated shortest distance between two vertices. Note that only paths with estimated shortest distance are recorded, longer paths are disgarded. Table 2 shows average number pf paths returned by decentralized search with full branch tie strategies and TreeSketch. The average path count of full branch decentralized search is significant higher than that of TreeSketch, from 3.73 to 345.13 times for various graphs.

Moreover, not only full branch decentralized search return higher number of paths, but the returned paths are also not

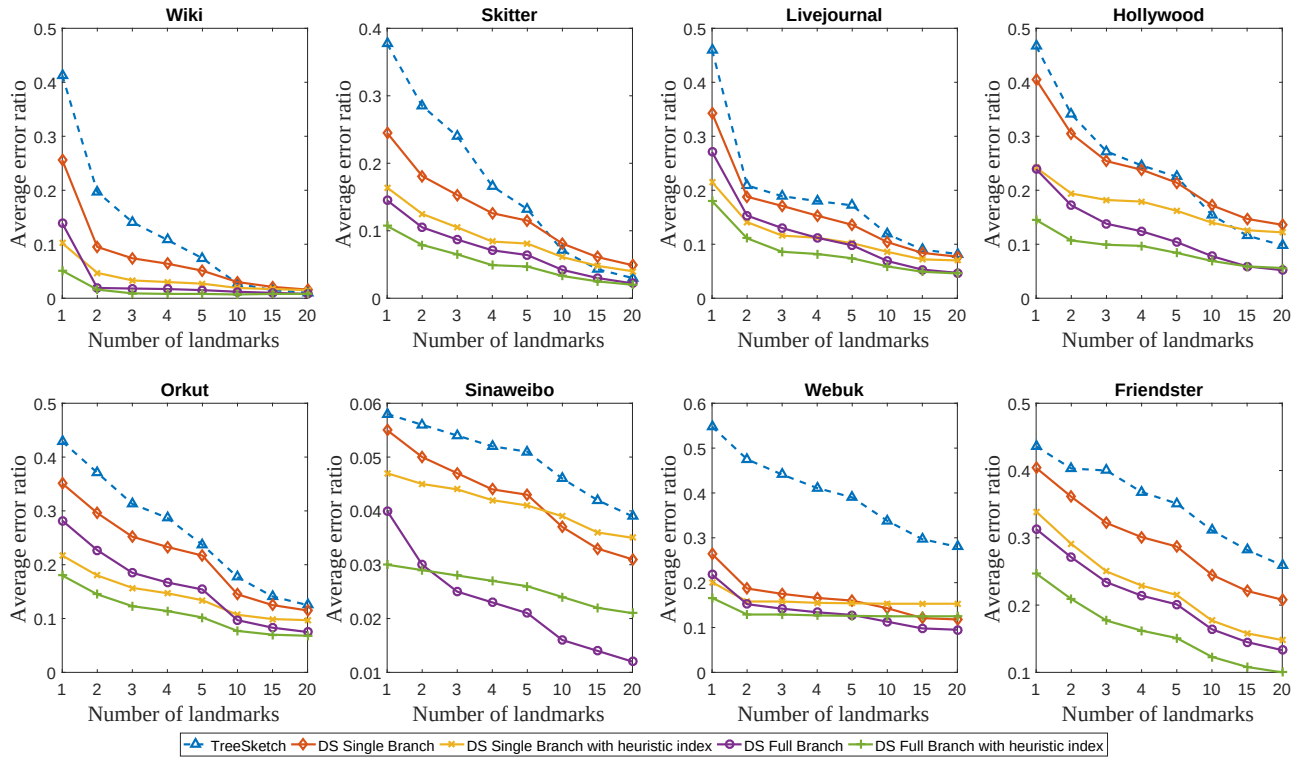


Figure 4: The accuracy of the approximated distance of decentralized search and various optimizations compared to TreeSketch.

Table 3: Index overhead per landmark

Graph	Index size(MB)	Query size(Byte)
Wiki	189.9	369.8
Skitter	143.7	412.9
Livejournal	429.4	419.9
Hollywood	84.1	377.5
Orkut	246.3	390.7
Sinaweibo	4313.8	367.4
Webuk	4068.9	506.5
Friendster	5497.7	437.7

constrained by the label of source and target vertices like TreeSketch do, i.e., paths found by TreeSketch only contain vertices in labels of source and target vertex. We define the out ratio of a path longer than 1 hop as follow:

$$Out\_ratio(p(s,t)) = \frac{|\{u : u \in p(s,t), u \notin L(s) \cup L(t)\}|}{|p(s,t)| - 2}$$

It shows the degree of a path relying on the labels of source and target vertex. The higher the value, the lower the dependence. The average out ratio for full branch decentralized search on various graphs ranges from 0.301 to 0.501.

7.5 Overhead

The index overhead is shown in table 3. In our implementation, the index of each vertex is stored as vector of C++ standard library. And each vertex id is represented by 8-byte unsigned long. The size shown in table 3 is the sum of vector size of each vertex.

Fig. 5 shows the online search overhead in log scale for both non-parallel mode and parallel batch mode. [add treesketch data later] By examing all the neighbors of a vertex at each step, decentralized search introduces much higher online search overhead than TreeSketch. Nevertheless, due to that decentralized search has very low foot print, large amount of searches can run independently in parallel efficiently. We can see in Fig. 5, the batch mode perform 100,000 queries simultaneously, which lead to very low average search time for each query. The search time is reduced to from 0.18% to 1.1% for single branch and from 0.15% to 1.3% for full branch compared to non-parallel mode.

For both non-parallel and parallel mode, full branch tie break strategy introduces much higher overhead than single branch strategy. Ranging from 3.3 to 30.8 times higher search time for non-parallel version and from 5.1 to 44.4 times higher average search time for parallel version.

[search memory overhead]

[search time to graph stat analysis]

7.6 Scalability

Since our algorithm is designed for large-scale networks, scalability is another major concern of our algorithm. Due to that we implement the algorithm in a distributed setting and execute queries in a parallel way, we study how our algorithm performs when number of machines and queries increase. We only perform single branch decentralized search here as full branch search is equal to multiple single branch searches.

We first evaluate the search time when number of ma-

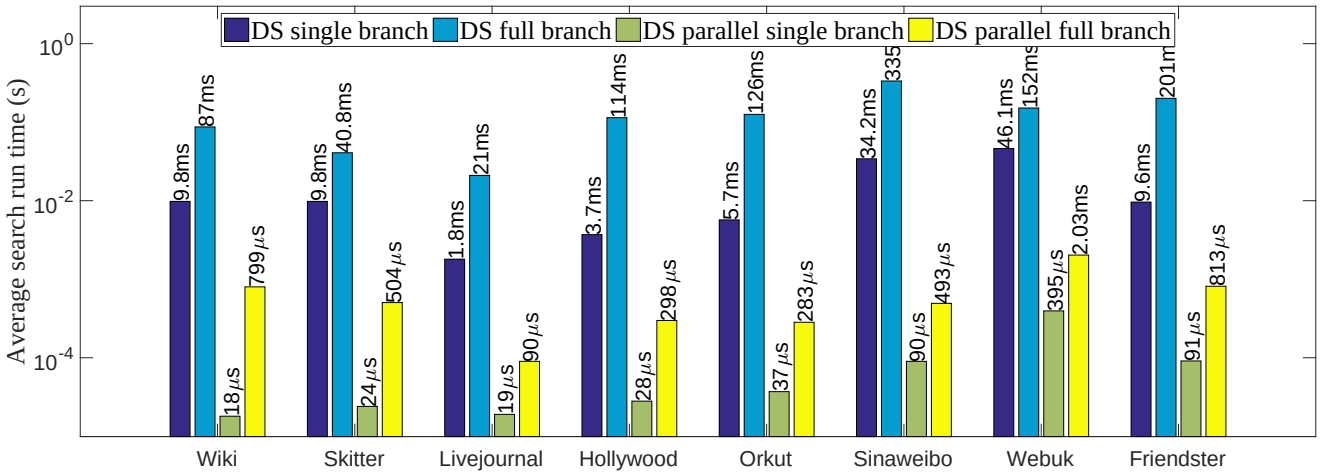


Figure 5: The average search time of decentralized search with various optimizations.

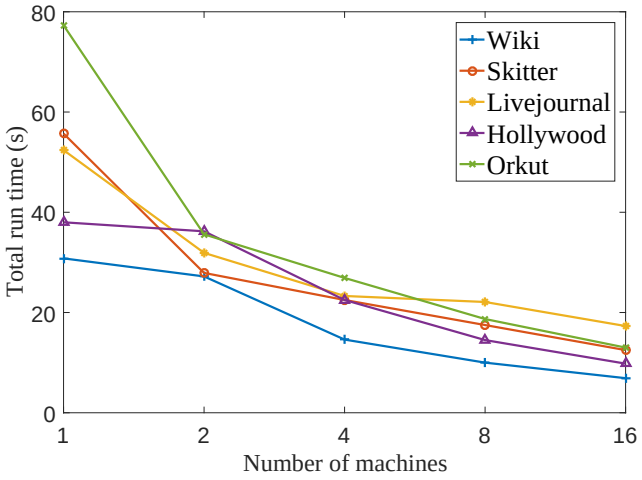


Figure 6: The average search time as the number of machines increase

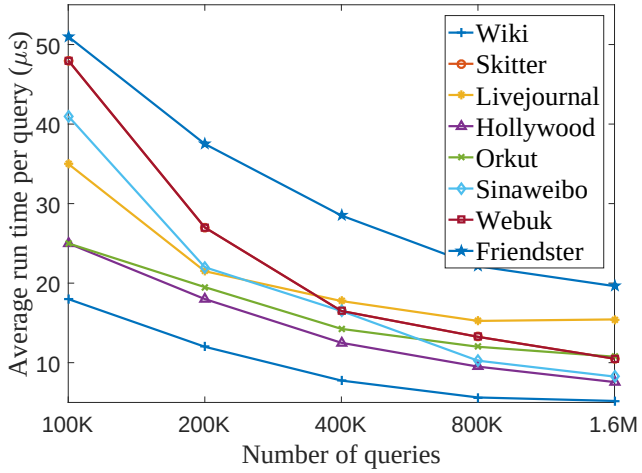


Figure 7: The average search time as the number of queries increase

chines increasing. Results shown in Fig. 6 are averaged over 1,000,000 queries. We can see the trend is that the average run-time decreases as the number of machines increase. The average search time decreases fast when small number of machines are deployed and slow down when large number of machines are used. The total run time on 16 machines range from 0.17 to 0.33 of the total run time on a single machine for various graphs.

[use the strong scaling figure here?]

We observe great scalability of decentralized search as number of queries increase. All experiments are carried on 20 machines in this section. We can see in Fig. 7 that the average search time quickly goes down when the number of queries is small. Because for small number of queries, constant overheads such as engine start/stop is the major part of run time. For large number of queries, the average run time still keep dropping. The growth of total search time never catch up with the growth of number of queries until it reaches the system limit, i.e. memory limit or network limit.

8. CONCLUSION

In this paper, we describe a novel method to combine online and offline processing to allow approximate shortest path searches for extremely large graphs with high distance accuracy, path diversity and low overhead. We demonstrate that different accuracy and overhead levels can be achieved by controlling the search space of decentralized searches. We also propose a more effective heuristic approach for constructing indexes of the network that can improve the accuracy of the decentralized search without increasing preprocessing and online searching overhead. We implement our algorithm for cloud computing graph processing platforms, and demonstrate that our system can handle extremely large graphs and processing millions of queries in parallel.

9. REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *Proceedings of the 10th International Conference on Experimental Algorithms*,

SEA'11, pages 230–241, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 349–360, New York, NY, USA, 2013. ACM.

[3] T. Akiba, C. Sommer, and K.-i. Kawarabayashi. Shortest-path queries for complex networks: Exploiting low tree-width outside the core. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 144–155, New York, NY, USA, 2012. ACM.

[4] M. Christoforaki and T. Suel. Estimating pairwise distances in large graphs. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 335–344, Oct 2014.

[5] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 937–946, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[6] V. Floreskul, K. Tretyakov, and M. Dumas. Memory-efficient fast shortest path estimation in large social networks. In *Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

[7] A. W.-C. Fu, H. Wu, J. Cheng, and R. C.-W. Wong. Is-label: An independent-set based labeling scheme for point-to-point distance querying. *Proc. VLDB Endow.*, 6(6):457–468, Apr. 2013.

[8] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 156–165, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[9] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 17–30, Hollywood, CA, 2012. USENIX.

[10] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 499–508, New York, NY, USA, 2010. ACM.

[11] R. Jin, N. Ruan, Y. Xiang, and V. Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 445–456, New York, NY, USA, 2012. ACM.

[12] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM.

[13] J. Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.

[14] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[15] M. Maier, M. Rattigan, and D. Jensen. Indexing network structure with shortest-path trees. *ACM Trans. Knowl. Discov. Data*, 5(3):15:1–15:25, Aug. 2011.

[16] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 867–876, New York, NY, USA, 2009. ACM.

[17] M. Qiao, H. Cheng, L. Chang, and J. Yu. Approximate shortest distance computing: A query-dependent local landmark scheme. *Knowledge and Data Engineering, IEEE Transactions on*, 26(1):55–68, Jan 2014.

[18] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:*, 39(6), Dec. 2014.

[19] M. Riondato and E. M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 413–422, New York, NY, USA, 2014. ACM.

[20] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[21] F. Takes and W. Kosters. Adaptive landmark selection strategies for fast shortest path computation in large real-world graphs. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM International Joint Conferences on*, volume 1, pages 27–34, Aug 2014.

[22] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, pages 143–152, New York, NY, USA, 2003. ACM.

[23] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, Jan. 2005.

[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 *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1785–1794. ACM, 2011.

[25] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 563–572, New York, NY, USA, 2007. ACM.

[26] F. Wei. Tedi: Efficient shortest path query answering on graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 99–110, New York, NY, USA, 2010. ACM.

[27] X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y.

Zhao. Orion: Shortest path estimation for large social graphs. In *Proceedings of the 3rd Wonference on Online Social Networks*, WOSN'10, pages 9–9, Berkeley, CA, USA, 2010. USENIX Association.