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

ABSTRACT

Approximate shortest paths for extremely large-scale complex networks is a challenging problem, where existing works require large overhead to achieve high accuracy and diversity for estimated paths, especially for large graphs with millions of vertices. In this paper, we propose an online search approach based on preprocessed indexes, to approximate point-to-point shortest paths. The approach is able to find more accurate and diverse paths with limited index overhead and require low search overhead. The level of accuracy and required resource can be balanced by dynamically controlling the search space to meet various application needs. Furthermore, a new heuristic index construction algorithm is introduced that can greatly increase the approximation accuracy and involve no additional index overheads. To handle extreme size graphs, we build a query processing system with our algorithm on distributed graph processing platforms. The system also support parallel processing of online searches to achieve high throughput for large number of queries. We evaluate our algorithm on various real world graphs from different disciplines with up to billions of edges and our system can process hundreds of thousand queries per second on these graphs.

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 [16]. As their sizes keep increasing, scaling up algorithms to handle extreme size graphs with billions of 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 is finding shortest paths in the network, an operation that 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 [26], analyzing influential people [12]. Estimating minimum round trip time between hosts without direct measurement is another application in technology networks [23].

Although previous works have studied the shortest path problem on large road networks extensively and had effective approaches. A large category of networks known as the complex networks has very different structures, i.e. following power law degree distributions, exhibiting small diameters, etc. Approaches for road networks do not perform well on complex networks. In this paper, we focus 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 [17, 25, 3, 18, 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 [24, 8, 17, 10, 25, 18]are widely used for approximate shortest path/distance between vertices. 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. 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 on applying online search to indexed graph limit the search space to sub-graphs constructed by vertices in labels of source and target vertices [10, 18]. The accuracy and diversity of approximated path are constrained this way, e.g. only short-cut edges directly connecting vertices in labels can be found. To overcome this problem, we propose to performs a heuristic search on the indexed graph that is

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

guided by locally collected information from labels of nearby vertices. The advantage is that the search can expand the search space into edges that have not been indexed to achieve higher accuracy and diversity of approximated path with limited index size. The heuristic search that we use is called decentralized search which was introduced in [13, ?]. Here the "'decentralized"' means that the decision of the search is made based solely on local information which, in our context, is the labels of neighbor vertices at each step of the search.

Decentralized search is very light weighted. The number of visited vertices for decentralized search is bounded by the diameter of the network. Considering that complex networks have relatively short diameters, decentralized search can finish in limited number of steps. The search can also adjust its search space to balance between different level of performance and required resources for each search. This makes the search very versatile to meet various application needs.

The performance of decentralized search relies heavily on the index. Landmark selecting problem has been well studied in [17, 22]. 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. To achieve better accuracy without increasing index overhead, we introduce a heuristic index construction algorithm to control shortest paths to be indexed during preprocessing. The propose approach outperforms random shortest path indexing by large margin on real networks.

Based on our algorithm design, we further develop a query-processing system based on distributed cloud infrastructure to support large scale graph with billions of edges. 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. 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.

The light-weighted decentralized search allows a large number of queries to run in parallel so that the system can achieve high query processing throughput. There are two properties of decentralized search that makes it very suitable for parallel processing. First, decentralized search has small 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 only have *RAR* data dependency on the index and underlying graph. Multiple searches can run independently on the same graph and index. These two properties makes it possible for a large number of searches running in parallel efficiently without reaching the physical limit of machines, i.e. memory size or network bandwidth. For example, in our experiments, we show 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.

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 to improve online search accuracy without increasing index overheads;
- We achieve efficient query processing and good scalability with distributed implementation and parallel processing;
- Experiments on various real world 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 2 we show previous works on exact and approximate approaches. Section 3 provides notations and definitions used in this paper. We explain index guided decentralized search for shortest path approximation in Section 4. Section 5 discusses index construction algorithm. In Section 6 we show details on our distributed implementation. The evaluations of our algorithm is in Section 7. We conclude our work in Section 8.

2. RELATED WORKS

The majority of the exact approaches are based on either 2-hop cover [5, 2] or tree decomposition [3, 27]. For the former one, finding optimal 2-hop covers is a challenging problem. Reference [2] takes a different approach that solve 2-hop cover problem with graph traversals which has better scalability. Reference [11] borrowed the highway concept from shortest path algorithms on road networks and construct a spanning tree as a "'highway"'. Reference [7] introduced an effective disk-based label indexing method based on independent set.

Since the exact approaches do not scale well, approximate algorithms are also well studied for large-scale complex networks. Landmark based algorithms are extensively studied for approximating shortest path/distance [24, 8, 17, 6, 15]. Although theoretical study of such algorithms does not reveal promising results [24], they work well in practice. Reference [17, 22] 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, 25, 18] that index shortest paths are proposed to alleviate this problem. The problem has also been formed as a learning problem [4] and mapped to low-dimension Euclidean coordinate spaces [28] to find approximated answers.

Our work falls into the category of applying online searches to indexed graphs. A* search is used for online query based on indexes constructed by landmarks to answer exact shortest path queries [8]. However, the cost of each A* search is still very high for large scale networks. There are also a few work [10, 18] perform online searches on sub-graphs consist of labels of source and target vertex. Although search space in [10] is small, path accuracy and diversity are compromised due to the constraints of the search space. In [18], by controlling the width of shortest path tree the online search visits, differentiated accuracy levels can be achieved. But the search space can only be coarsely controlled. For graphs with power-law degree distribution, it is impractical to have the width larger than 1.

3. PRELIMINARIES

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 $p(s,t) = (s, v_1, v_2, ..., t)$ with a length of $|p(s,t)|$ close to the exact distance between s and t. 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 focus on unweighted, undirected graphs in this paper.

Our method is motivated by the idea of using landmarks as the basis for indexes. Specifically, given a graph G and a set of k landmarks $(l_1, l_2, ..., l_k)$, an index contains a label $L(v)$ for each vertex which store the shortest path to each landmark. The label can be constructed by building a shortest path tree SPT from each landmark.

The least common ancestor of two vertex in a tree is the furthest ancestor from the root, we denote it as $LCA(s,t)$. The shortest path distance satisfies the triangle inequality, for an arbitrary pair of vertices s and t, the following bound holds:

$$d_G(s,t) \le min_l \{d_G(s, LCA_l(s,t)) + d_G(LCA_l(s,t), t)\} \quad (1)$$

This upper bound, which is referred to as LCA distance and denoted by $d_{LCA}(s,t)$, can be used as an approximation of the distance from s and t. 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 explains how to apply decentralized search on indexed graphs. Several aspects of the search including termination condition, bidirectional search and tie breaking strategy are discussed.

4.1 Index guided decentralized search

To answer a shortest path query, decentralized search iteratively collect local distance information and visit vertex with least approximated distance to the target. More specifically, for a given pair of source s and target vertex t on an indexed graph. The search first set the source vertex as the vertex to visit in first step and appends it to the approximated path $\tilde{p}(s,t)$. At each step, suppose that the search is visiting vertex u, it traverse all the neighbor vertices of u. For each neighbor vertex v_i, the $d_{LCA}(v_i,t)$ is calculated. Then the search set the neighbor vertex with smallest $d_{LCA}(v_i,t)$ as the vertex to visit in next step and appended v_i to $\tilde{p}(s,t)$.

Terminating when the search reaches the target vertex is a valid but not an ideal termination condition. Since shortest paths have optimal substructure, i.e. the path between any two vertices along a shortest path is also the shortest path of them. So the search can stop once it reach arbitrary vertex $u \in L(t)$ as the path $p_L(u,t) \subseteq L(t)$ is a shortest path from u to t. Decentralized search cannot find a shorter path than $p_L(u,t)$. The detailed algorithm of decentralized search is depicted in Algorithm 1.

At each step, by examining neighbor vertices the search is able to explore a subset of the edges that is not indexed 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. 1(a) from vertex 14 to vertex 17, decentralized search find a edge $(6,7)$ as vertex 7 has a LCA distance of 3 which is shorter than LCA distance 5 from vertex 1 to 17.

Algorithm 1 Decentralized search

```
function DecentralizedSearch(s, t)
    p̃(s,t) ← ∅
    u ← s
    append u to p̃(s,t)
    while u ∉ L(t) do
        d_min ← ∞
        w ← u
        for each v_i adjacent to u do
            if d_LCA(v_i,t) < d_min then
                d_min ← d_LCA(v_i,t)
                w ← v_i
        u ← w
        append u to p̃(s,t)
    p_remain ← p_L u,t excluding u
    append p_remain to p̃(s,t)
    return p̃(s,t)
```

THEOREM 1. *If the target vertex is reachable from the source, the decentralized search terminate in as much as $2\sigma_{max}$ steps, where σ_{max} is the diameter of the graph.*

PROOF SKETCH. For an arbitrary source vertex s and a reachable target vertex t, the following bound holds:

$$d_{LCA}(s,t) = d_G(s, LCA(s,t)) + d_G(LCA(s,t), t) \le 2\sigma_{max}$$

And at each step, suppose decentralized search is visiting vertex u and $u \ne t$. Assume the tightest upper bound in equation 1 is achieved on shortest path tree SPT_l rooted at landmark l. Let v be the neighbor vertex of u on path path $p_{LCA_l}(u,t)$. Since SPT_l does not have cycle, $p_{LCA_l}(v,t) \in p_{LCA_l}(u,t)$. Therefore the following equation holds:

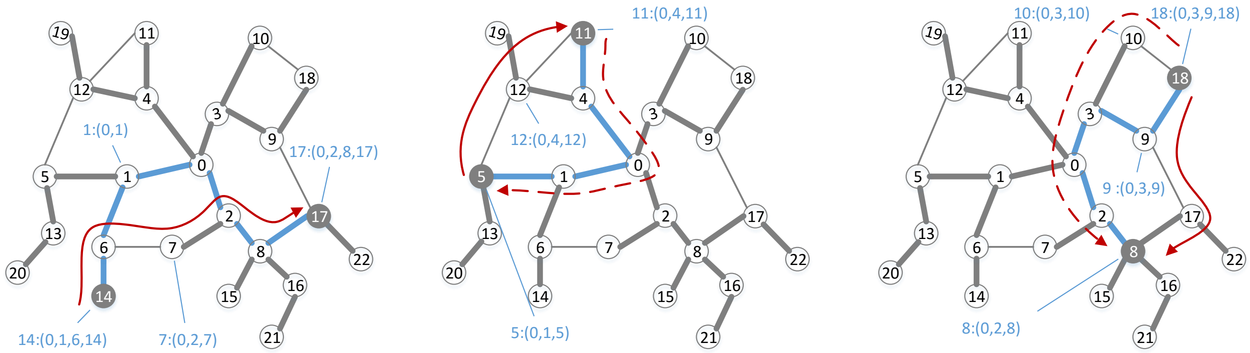
$$d_{LCA}(v,t) = |p_{LCA_l}(v,t)| = |p_{LCA_l}(u,t)| - 1 = d_{LCA}(u,t) - 1$$

Since decentralized search always pick the neighbor with shortest LCA distance to the target, the LCA distance to the target at each step decreases at least by 1. Therefore, the decentralized search terminates in at most $2\sigma_{max}$ steps. □

The time complexity of Decentralized search depends on the max degree and the diameter of the graph. Decentralized search take at most $2\sigma_{max}$ steps to finish according to Theorem 1. For each step, the search 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, and time complexity for each LCA computation is $O(h)$[1], where h is the height of the indexed shortest path tree and $h \le \sigma_{max}$. 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, space complexity for offline index and space complexity for online query. The space required for offline index is $O(k\sigma_{max}n)$, where n is number of vertices. For each query, $O(k\sigma_{max})$ space is required to store the labels of target vertex and the vertex that is being examined. $O(2\sigma_{max})$ space is required to store the approximated path. Combining them

[1] $O(h)$ is for simple online algorithm, off-line algorithms can achieve time complexity of $O(1)$.



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

Figure 1: Examples of decentralized search on indexed graph. Bold lines denote the indexed edges. Curved lines denoted paths being found, with arrows showing the direction. Dark vertices denote source and target vertex. Labels of vertices are shown in $vertex : label$ format.

together, the online search space complexity of decentralized search is $O(k\sigma_{max})$.

4.2 Bi-directional search

In this section, we show how to apply the idea of bidirectional search to decentralized search. In bidirectional decentralized search, the backward search starts at target vertex and is driven by the goal to reach the source vertex. The forward search and backward search may explore different search spaces due to the different start point and search goal. By exploring a different search space the backward search may find a shorter approximated path. This, however, is quite different from the application of bidirectional search in BFS or A* search where the main focus is to reduce search space.

An example of directional decentralized search is shown in Fig. 1(b). Let 11 be source and 5 be target. The backward search can find a shorter path $p_{bwd} = (5, 12, 11)$ than the path $p_{fwd} = (11, 4, 0, 1, 5)$ found by the forward search. The backward search find a shorter path by exploring edge $(5, 12)$. However, this edge is invisible to forward search because when the search traversing vertex 4, it prioritize 0 than 12 due to the former one has lower LDA distance to target vertex 5.

It is not guaranteed that the forward search and backward search will eventually meet at any vertex except the source and target vertex due to the different search space. As shown in our previous example, $p_{fwd} \cap p_{bwd} = (11, 5)$. Actually, in decentralized search, the forward search and backward search are mostly two independent search, that the only interaction of them is when combining the search results. Search results can be combined simply by picking the shorter path or finding short cuts connecting resulting paths.

4.3 Tie breaking strategy

Ties happen frequently in decentralized search, especially when the number of landmarks is small. In the decentralized search, a tie means multiple neighbor vertices have same LCA distance to the target in a step. The reason a tie happens is that there is not sufficient information in the index that can separate neighbor vertices for a query. For example in Fig. 1(c), consider a search from 18 to 8. When traversing neighbors of vertex 18, both vertex 10 and 9 have the same LCA distance to target vertex 8, but their actual distances to vertex 8 are different due to that short cut edge $(9, 17)$ is currently invisible to the decentralized search.

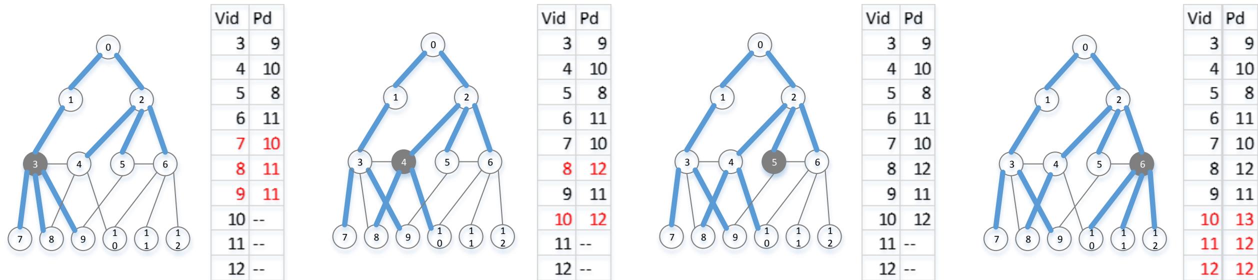
The search space of the decentralized search can be increased by expanding the search onto each tie vertex. This increase both the performance, i.e. chances to find a shorter path and number of paths being found, and the cost of the search. By controlling the search space this way, decentralized search is able to achieve differentiate level of accuracy.

Two extreme ways to deal with ties are either only visit one vertex or visit all vertices in the next step. The former one incurs least search cost and has the least possibility to find a shorter path, we refer to it as single branch decentralized search. The latter one requires most effort and can lead to the shortest path the decentralized search could possibly find, we refer to it as full branch decentralized search.

5. GREEDY INDEX CONSTRUCTION

This section describes the greedy index construction algorithm. For a vertex, all shortest paths from a landmark can be indexed as the label of it. We study the problem of deciding which shortest path to be indexed can lead to better online query accuracy for average cases.

As the core of decentralized search is to iteratively find neighbor vertices that has shortest LCA distance to the target. From the point of view of a vertex u, for landmark l, if the indexed shortest path p_i intersects with more indexed shortest paths of other vertices, then for queries that have u as target vertex, the higher the possibility that other vertices have a LCA with u that is further from l on shortest path tree SPT_l. 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 increase accuracy for average cases. To represent the "centrality" of a shortest path, we use the sum of vertex centrality along the path. Although betweenness centrality fits our needs very well, the computation cost is high [20]. So we use degree as an alternative and refer the



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

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

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

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

Figure 2: Heuristic index construction pick shortest path with highest path degree during breadth first search

sum of degree of vertices along a path as path degree, denoted by Pd.

Index construction procedure can be easily modified to index the shortest path with highest path degree. Note that path degree of shortest path follows optimal substructure, i.e. if a shortest path $(u, .., w, ..., v)$ has the highest path degree among all the shortest paths from u to v, then the path degree of $(u, ..., w)$ is also the highest among all the shortest paths from u to w. To modify the index construction, during BFS, suppose the search is visiting vertex u and reach its neighbor v with non empty $L(v)$. A label update is performed if $|L(v)| > |L(u)|$ and $Pd(u) + \rho(v) > Pd(v)$, where $\rho(v)$ denote the degree of vertex v. The detailed algorithm of greedy index construction is depicted in 1.

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.

Note that if the landmark set is relatively large, then following the highest path degree heuristic may lead to redundant labels, i.e. similar indexed shortest path tree for multiple landmarks, which can compromise the accuracy of online search. A simple way to solve this problem is to prioritize shortest paths which overlaps fewer vertices with shortest paths that have already been indexed.

6. DISTRIBUTED IMPLEMENTATIONS

To handle extremely large graphs and large amount of queries, we implement decentralized search on a distributed general graph processing platform, Powergraph [9]. As decentralized search has low online search space complexity and have very low data dependency upon each other, it is well suited to run multiple searches in a parallel way.

An overview of our shortest path query processing system is shown in Fig. 3. The system first partition the graph with Powergraph onto multiple machines. Then several BFSs are performed to construct the index. After the index has been 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.

Algorithm 2 Greedy index construction on landmark l

function INDEX CONSTRUCTION(l)
  For each v in G: $L(v) \leftarrow \emptyset$
  For each v in G: $Pd(v) \leftarrow 0$
  $Q \leftarrow \emptyset$
  $L(l) = l$
  $Pd(l) = \rho(l)$ ▷ ρ denotes degree
  $Q.push(l)$
  while $Q \neq \emptyset$ **do**
    $u = Q.pop()$
    for each v_i adjecent to u **do**
      if $L(v_i) \neq \emptyset$ **then**
        $L(v_i) = L(u)$
        append v_i to $L(v_i)$
        $Pd(v_i) = Pd(u) + \rho(v_i)$
        $Q.push(v_i)$
      else if $|L(v_i)| > |L(u)|$ **and** $Pd(v_i) < Pd(u) + \rho(v_i)$ **then**
        $L(v_i) = L(u)$
        append v_i to $L(v_i)$
        $Pd(v_i) = Pd(u) + \rho(v_i)$
  return L

6.1 Decentralized search vertex-program

Decentralized search can be easily implemented as vertex-programs in Gather-Apply-Scatter model used by Powergraph. Index is stored distributively as vertex data. Each query instance contains the approximated path and label of target vertex, as it is not accessible on each machine locally. Each step of decentralized search is split into gather, apply and scatter phase. In Gather phase, LCA distance to target vertex d_{LCA} is collected from each neighbor and accumulated by finding the neighbor with the smallest LCA distance to target. The accumulated result is returned as a next step candidate. In apply phase, the candidate is appended to the approximated path $p_{appr.}$ and the termination condition is checked. If it is met, the result path will be recorded and the query will be terminated. Otherwise, program will proceed to scatter phase to start a new vertex-program on the

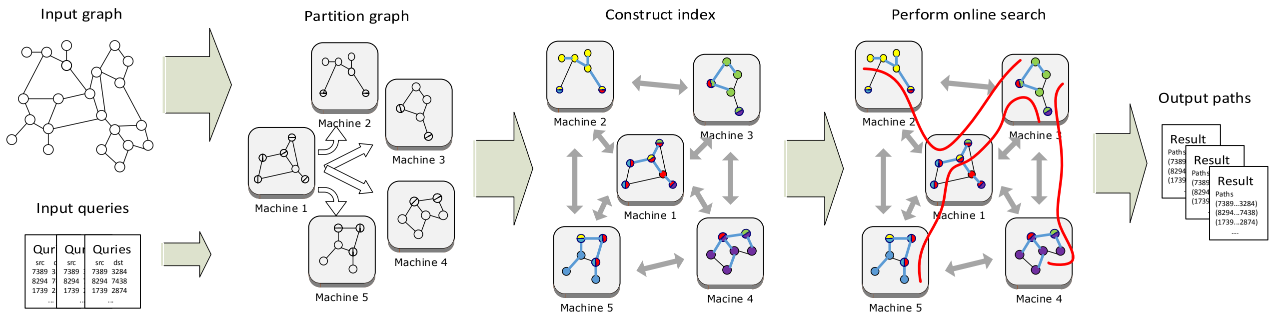


Figure 3: A overview of distributed shortest path query processing system

candidate vertex and pass on the query instance to them. Algorithm 3 shows the detailed algorithm.

Algorithm 3 Decentralized search vertex program on u

---

function GATHER($L(v), L(t)$) ▷ on neighbor vertex v
  return $d_{LCA}(v, t)$, v
function SUM($d_{LCA}(v_1, t)$, v_1, $d_{LCA}(v_2, t)$, v_2)
  if $d_{LCA}(v_1, t) \leq d_{LCA}(v_2, t)$ **then**
    return $d_{LCA}(v_1, t), v_1$
  else
    return $d_{LCA}(v_2, t), v_2$
function APPLY($L(t)$, $\tilde{p}(s, t)$, $d_{LCA}(v, t)$)
  if $v \in L(t)$ **then**
    $p_{remain} \leftarrow$ path from v to t in $L(t)$
    append p_{remain} to $\tilde{p}(s, t)$
    store $\tilde{p}(s, t)$
    $term$ = true
  else
    append v to $\tilde{p}(s, t)$
    $term$ = false
function SCATTER($L(t)$, $\tilde{p}(s, t)$, $term$)
  if $\neg term$ **then**
    Activate(v, $L(t)$, $\tilde{p}(s, t)$)

The communication for the decentralized search happen during gather and scatter phase. In gather phase, the label of target vertex need to be passed to multiple machines, and the size is $O(k\sigma_{max})$. And each gather function return a d_{LCA} along side of its id. So for each query, only $O(k\sigma_{max})$ size of data is transferred. In scatter phase, communication happens when a vertex is chosen as the next step candidate 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, $O(k\sigma_{max})$ size of data may be transferred. The search will take as much as $2\sigma_{max}$ steps. So the overall communication overhead for each query is $O(k{\sigma_{max}}^2)$.

During decentralized search, only the approximated path $\tilde{p}$ is updated at each step. Therefore, there is only RAR type of data dependency among multiple decentralized searches on the underlying graph. Depends on implementations, there may be output dependency, i.e. WAW, when output $\tilde{p}$ to the same container on each machine.

The low memory and communication cost, along side with RAR only data dependency during the search makes a large number of decentralized search 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

According to tie breaking strategy, multiple neighbors may be returned as candidates at gather phase. In this case, the search instance copies itself into multiple search instances, append each candidate to each search instance at apply phase and activate them at scatter phase. The problem is, as search instances created at apply phase become independent when the search proceeding to next step, during the future steps, even one search instance find a shorter path than others, it can hardly terminate other searches as such synchronizations is too costly in distributed settings. With this limitation, a search may end up with excessive number of child search instances. To overcome this problem, in our implementation, only one candidate will be labeled as a "‘main"’ candidate at each step. For candidates that are not the "‘main"’ candidate, an extra termination condition is applied. In the next step, if the search cannot find a shorter path than expected, i.e. $|p_{appr.}| + d_LCA$, the search will stop, and no result will be recorded. This can control the search space effectively without compromise much accuracy.

6.3 Prune LCA computation

A major part of computation load of decentralized search is from large number of LCA computations. It is possible to prune number of LCA computation required at each step for decentralized search to reduce the overall computation load. Suppose the search is visiting vertex u, which means the $d_{LCA}(u, t)$ has already been calculated in previous step. If a neighbor v is a child of u on the indexed shortest path tree SPT_l, then the LCA computation for v and t on SPT_l does not need to be performed as it is clearly that $d_{LCA_l}(v, t) > d_{LCA_l}(u, t)$. In practice, this principle can prune almost half of the total number of LCA computations of a single search on average.

Table 1: Datasets

Dataset	Type	$\lvert V_{wcc}\rvert$	$\lvert E_{wcc}\rvert$	$\overline{\sigma}$
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.

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 in 7.5. The throughput of our query processing system is shown in 7.6. In the last, we study the scalability of our system in 7.7.

7.1 Datasets

We evaluate our algorithm on 8 graphs from different disciplines as shown in table 1. All graphs are complex networks that have power-law degree distribution and relatively small diameter. To simplify our experiments, we treat each graph as undirected, un-weighted graphs. We only use the largest weakly connected component of each graph which consist of more than 90% of vertices for each graph. All datasets are collected from Snap [14] and NetworkRepository [21].

7.2 Experiment settings

We evaluate our algorithms in both distributed setting and centralized setting. For the distributed setting, we use 20 Amazon EC2 m4.xlarge virtual machines. Each virtual machine has 4 vCPUs and 16 GB memory. For centralized version, we use a Cloudlab [19] 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 [17]. When 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.

Queries in our experiments are randomly generated. For accuracy evaluations, since performing BFS for exact distance on large graphs is extremely slow, we randomly choose 1,000 vertices of each graph as source vertices and randomly pick 100 target vertices for each source vertex. We are able to generate 100,000 queries for accuracy evaluation in this way.

Table 2: Path diversity (k = 2)

Graph	Path cnt(DS)	Path cnt(TS)	$\overline{r_p}$
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

7.3 Approximation Accuracy

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

$$E_{\tilde{p}(s,t)} = \frac{|\tilde{p}(s,t)| - d_G(s,t)}{d_G(s,t)}$$

We show the results of 4 variants of decentralized search with mixture of different tie breaking 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 online search method, TreeSketch [10][2]. We vary the size of landmark sets and show the averaged results of 100,000 queries 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. When the full branch decentralized search are carried on the index constructed by our greedy heuristic, the performance gain is most noticeable, with 43.3% to 87.7% lower average error ratio for 1 landmark and 50% to 80% lower average error ratio for 20 landmarks than TreeSketch on all graphs.

Full branch tie strategy always outperforms single branch tie strategy with large margins with same landmark sets. The average error ratio of full branch search with regular index is 17.4% to 45.7% lower than single branch for 1 landmark and 19.5% to 61.8% lower with 20 landmarks on various graphs. As the number of landmark increases, the accuracy gain increases.

Decentralized search carried on index constructed by greedy heuristic has lower error ratio than decentralized search with regular index. The average error ratio is 14.5% to 60.2% lower for single branch and 21.1% to 63.3% lower for full branch for 1 landmark. For 20 landmarks, the search is 10.4% to 68.8% lower for single branch and 12.8% to 75% lower for full branch.

7.4 Path Diversity

We show in this section that decentralized search achieves better path diversity by finding more paths and not being constrained by the index. Table 2 shows average number pf paths with shortest approximated distance returned by decentralized search with full branch tie strategies compared to TreeSketch. The average path count of full branch decentralized search is much higher than that of TreeSketch, from 3.73 to 345.13 times for various graphs.

[2]The online search in [18] is similar to TreeSketch in nature so that we didn't include it to the comparison.

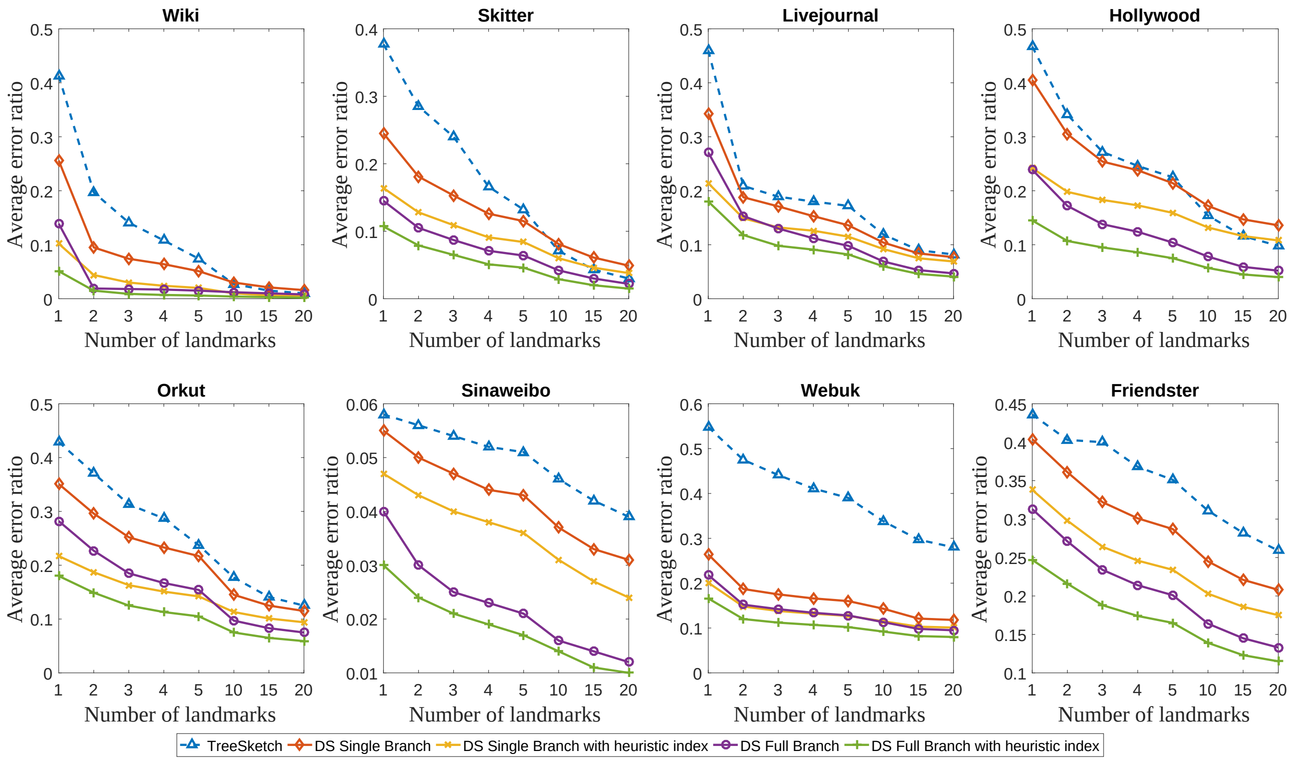


Figure 4: The accuracy of the approximated distance of decentralized search compared to TreeSketch.

Table 3: Space overhead

Graph	Index size per landmark(MB)	Query size(B)
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

Moreover, decentralized search is not restricted by label of source and target vertices. We define the ratio r_p as the number of vertices not in label source and target compared to total number of vertices except the source and target vertex:

$$r_p(s,t) = \frac{|\{u : u \in p(s,t), u \notin L(s) \cup L(t)\}|}{|v : v \in p(s,t), v \neq s, v \neq t|}$$

The higher the r_p's value, the lower the dependence of a path to label of source and target vertices. As shown in Table 2, the average r_p for full branch decentralized search on various graphs ranges from 0.301 to 0.501.

7.5 Overhead

The index and query overhead is shown in table 3. In our implementation, both the label for each vertex and approximated paths are 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.

7.6 Throughput

Fig. 5 shows the throughput of our system in log scale for both serial mode and parallel mode. The throughput of parallel mode is calculated based on running 100,000 queries simultaneously. The throughput of parallel mode is much higher than serial mode, from 94.7 to 544.4 times for single branch decentralized search and 74.7 to 679.1 times for full branch decentralized search.

We can also see that the due to the search duplicates itself for full branch decentralized search, the throughput is 3.2% to 30.4% of single branch for serial mode and 2.3% to 21.1% of single branch for parallel mode.

7.7 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 independent single branch searches, thus have the similar trend.

We first evaluate the search time when number of machines increasing. Results shown in Fig. 6 are based on the results of running 1,000,000 queries simultaneously. We can see the throughput increases as the number of machines increases. The throughput on 16 machines is 3.0 to 5.9 times higher than on a single machine for various graphs.

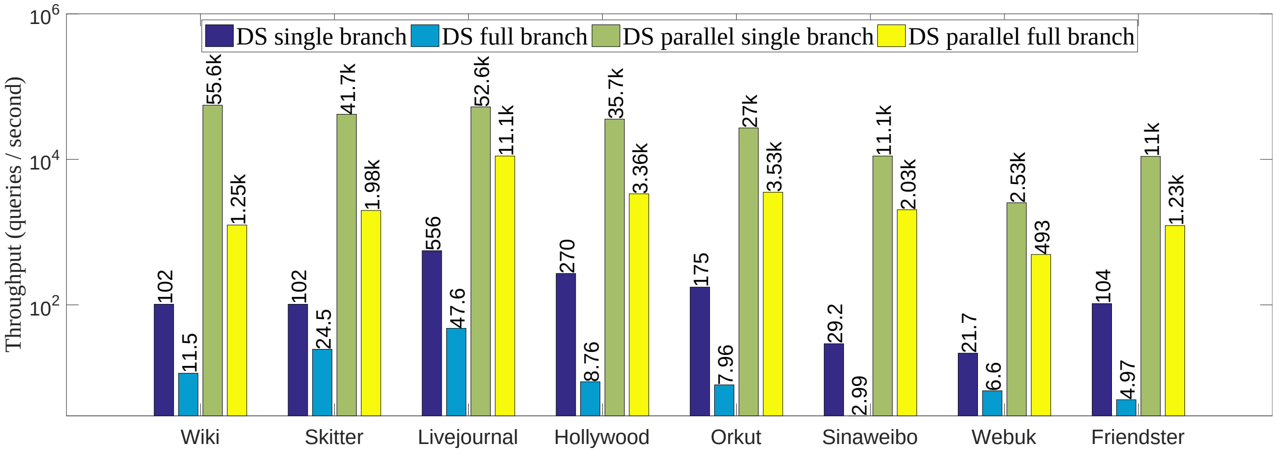


Figure 5: The throughput of serial and parallel version decentralized search.

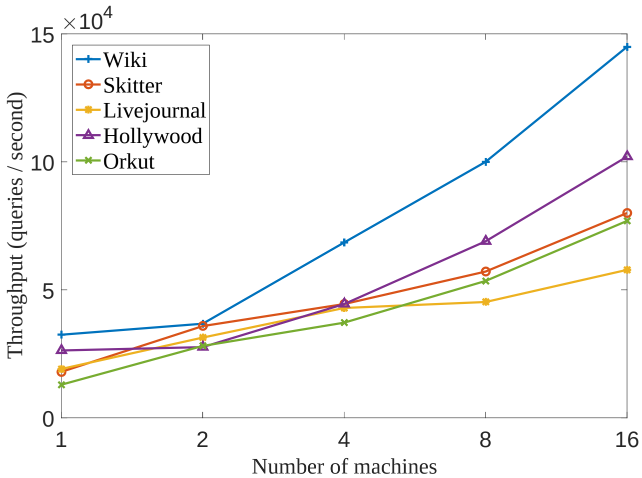


Figure 6: The throughput as the number of machines increases with 1 millions queries running simultaneously

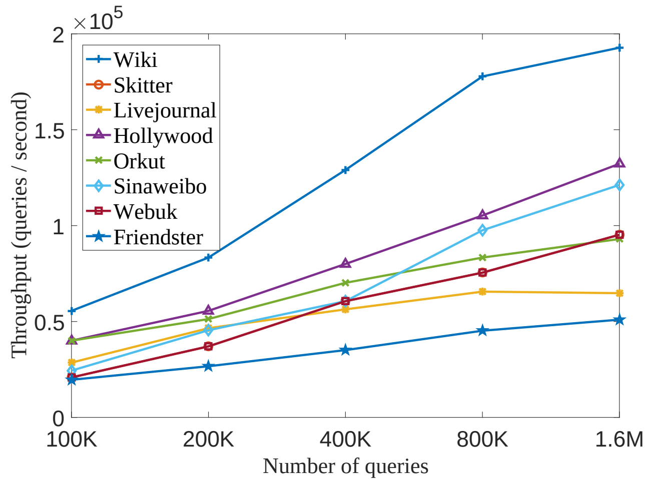


Figure 7: The throughput on 20 machines as the number of queries running simultaneously increases

Note that we do not have results for the 3 larger graphs (Sinaweibo, Webuk, Friendster) as they are not able to fit into less than 8 machines.

We also show the trend of the throughput as number of queries running simultaneously increases. All experiments are carried on 20 machines. We can see in Fig. 7 the constant growth of throughput as the number of queries running simultaneously increases. The growth of throughput slow down for large number of queries as the system limits are reached, i.e. memory size or network bandwidth. The throughput for 1,600,000 queries running simultaneously is 2.3 to 5.0 times higher than for 100,000 queries for various graphs.

8. CONCLUSION

In this paper, we describe a novel method to combine online and offline processing to allow approximate shortest path 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. We also develop an effective heuristic approach for constructing indexes that can improve the accuracy without increasing overhead. We implement our algorithm for cloud computing graph processing platforms, and demonstrate that our system can handle extremely large graphs and achieve high query processing throughput. The scalability of our system is good as both number of machines and number of queries running simultaneously increases.

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. Newman. *Networks: an introduction*. OUP Oxford, 2010.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.