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

Abstract—**Finding approximated 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 requires low search overhead. Furthermore, a new heuristic index construction algorithm is introduced that can greatly increase the approximation accuracy and involve no additional index overhead. To handle extreme size graphs, we build a query processing system with our algorithm on distributed graph processing platforms. The system also supports parallel processing of online searches to achieve high throughput for a large number of queries. We evaluate our algorithm on various real-world graphs from different disciplines with up to billions of edges, and we demonstrate that our system can process hundreds of thousand queries per second on these graphs with reduced overhead.**

I. 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 [17]. As their sizes keep increasing, scaling up algorithms to handle graphs with billions of edges remains a challenge that has drawn increased attention in recent years. Specifically, straight-forward graph algorithms 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, such as social sensitive search [29], estimating minimum round trip time [26], utility optimization [30], data center load balancing [24], network routing [8].

The emphasis of previous works on shortest path problem is mainly on road networks. An emerging category of networks known as the complex networks has very different structures, i.e., they follow power law degree distributions and exhibit small diameters. Our design is motivated by recent studies that combine both offline processing and online queries [18], [28], [2], [20], [12]. Among these approaches, landmark based algorithms are widely used to approximate shortest path or distance between vertices [27], [9], [18], [11], [28], [20]. Such algorithms select a small set of landmarks and construct an index that consists of labels for each vertex, which stores distances or shortest paths to landmarks. A relatively large set of landmarks is required for accurate approximation, which leads to large preprocessing overhead. 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 [11], [20]. The accuracy and diversity of the approximated paths 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 perform a heuristic search called the decentralized search [13], [14] on the indexed graph that is guided by locally collected information from labels of neighbor 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 the approximated paths with limited index size.

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 usually have relatively short diameters, decentralized search can finish in a limited number of steps. The search can also adjust its search space to balance between different levels of performance and required resources for each search.

The performance of decentralized search relies heavily on indexes. Landmark selecting problem has been well studied in [18], [25]. We observe that even with the same landmark set, choosing which shortest path from a vertex to the landmark to be indexed also plays an important role in the accuracy of the online search. Therehore, we introduce a heuristic index construction algorithm to control shortest paths to be indexed during preprocessing. The proposed approach outperforms random shortest path indexing by a 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.

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 make it very suitable for parallel processing. First, decentralized search has small space complexity and communication complexity. As 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 searches only have read after read (RAR) data dependencies on indexes and underlying graph. Multiple searches can run independently on the same graph and index. 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.

A. 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 II we show previous works on exact and approximate approaches. Section III provides notations and definitions used in this paper. We explain index guided decentralized search for shortest path approximation in Section IV. Section V discusses index construction algorithm. In Section VI we show details on our distributed implementation. The evaluations of our algorithm are in Section VII. We conclude our work in Section VIII.

II. Related Works

The majority of the exact approaches for shortest path problem are based on either 2-hop cover [5], [1] or tree decomposition [2], [31]. For the former one, finding optimal 2-hop covers is a challenging problem. Reference [1] proposed to solve the 2-hop cover problem with graph traversals which achieves better scalability. Reference [12] borrowed the highway concept from shortest path algorithms on road networks and constructed a spanning tree as a "highway" for complex networks. Most exact approaches do not scale well as the size of graphs increases.

Approximate algorithms have also been studied to achieve better scalability on large-scale complex networks. The majority of approximate algorithms is based on using landmarks as basis to construct offline indexes [27], [9], [18], [7], [16], [6], [19]. Although theoretical studies of such algorithms do not reveal promising results [27], they work well in practice. Landmark selection strategies for indexing is a critical problem in landmark based algorithms. Such a problem is proven to be NP-hard and various heuristics are provided in [18], [25]. [19] improved the index construction efficiency on distributed settings. Both shortest paths and distances can be indexed. A common problem for distance-only indexes is that they do not perform well for close pairs of vertices [2]. Algorithms that index shortest paths can alleviate this problem by exploit least common ancestors of close vertices [11], [28], [20]. The problem has also been formulated as a learning problem [4] and mapped to low-dimension Euclidean coordinate spaces [33] to find approximated answers. The landmark based approaches also extended to weighted graphs [32].

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 [9]. Although it is able to answer exact shortest path queries, the cost of each A* search is still very high for large-scale networks. There are also a few works [11], [20] that perform online searches on sub-graphs that consist of labels of the source and the target vertex for each query. Although the search space in [11] is small, path accuracy and diversity are compromised due to the constraints of the search space. In [20], by adjusting the width of shortest path tree the online search visits, differentiated accuracy levels can be achieved. But for graphs with power-law degree distributions, it is impractical to expand the search to a width of more than 1 as the search space will become too large.

III. 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 $d_G(s,t)$ between s and 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 that stores the shortest path to each landmark. The label can be constructed by building a shortest path tree SPT using BFS from each landmark.

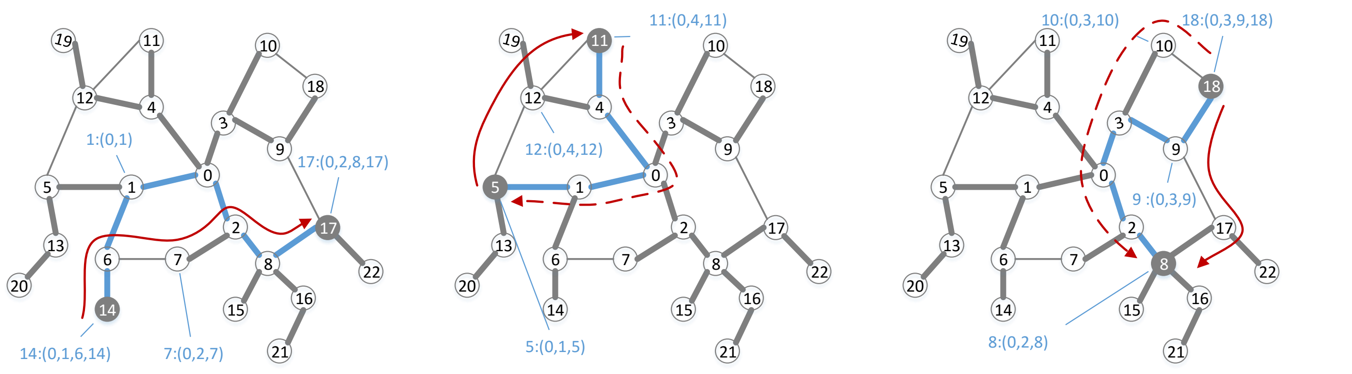
The least common ancestor of two vertices in a tree is the farthest ancestor from the root, which we denote as $LCA(s,t)$. The shortest distance satisfies the triangle inequality, i.e., for an arbitrary pair of vertices s and t, the following bound holds:

$$d_G(s,t) \leq min_l\{d_G(s, LCA_l(s,t)) + d_G(LCA_l(s,t), t)\} \tag{1}$$

This upper bound, which is referred to as the 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 $p_{LCA}(s,t)$. The LCA distance and related path for a specific landmark l is denoted as $d_{LCA_l}(s,t)$ and $p_{LCA_l}(s,t)$ respectively.

IV. Decentralized search for shortest path approximation

We propose to solve the point-to-point shortest path approximation problem using decentralized search with landmark-based indexes. This section explains how to apply decentralized search on indexed graphs. Several aspects of the search, such as termination condition, bidirectional search and tie breaking strategy, are discussed.



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

Fig. 1. Examples of decentralized searches on indexed graphs. Bold lines denote the indexed edges. Curved lines denote paths being found, with arrows showing the directions. Dark vertices denote source and target vertices. Labels of vertices are shown in the $vertex : label$ format.

A. *Index guided decentralized search*

To answer a shortest path query, decentralized search iteratively collects local distance information and visits the vertex with the 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 sets the source vertex as the vertex to visit in the 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 traverses all the neighbor vertices of u. For each neighbor vertex v_i, the metric $d_{LCA}(v_i,t)$ is calculated. Then the search sets the neighbor vertex with the smallest $d_{LCA}(v_i,t)$ as the vertex to visit in the next step, and appends v_i to $\tilde{p}(s,t)$.

When the search reaches the target vertex, the search process naturally stops. However, this is not the ideal termination condition. We observe that, as shortest paths have optimal substructure, i.e., the path between any two vertices along a shortest path is also the shortest path of them, the search procedure can stop once it reaches an arbitrary vertex u, such that $u \in L(t)$ (u is in the label of t. Evidently, decentralized search cannot find a shorter path than $p_L(u,t)$. The detailed algorithm of decentralized search is shown in Algorithm 1.

Observe that in this algorithm, by examining neighbor vertices, the search is able to explore a subset of the edges that are not indexed, to increase both accuracy and diversity of the path being found. For example, in Fig. 1(a), observe that for the path from vertex 14 to vertex 17, decentralized search finds an edge $(6,7)$ as vertex 7 has a LCA distance of 3, which is shorter than the LCA distance 5 from vertex 1 to 17.

Regarding the termination condition, we have the following theorem:

Theorem 1. *If the target vertex is reachable from the source, the decentralized search terminates in at most $2\sigma_{max}$ 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:

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

Next, observe that at each step, decentralized search is visiting vertex u and $u \neq t$. Assume the tightest upper bound in Equation (1) is achieved on the shortest path tree SPT_l rooted at landmark l. Let v be the neighbor vertex of u on the path $p_{LCA_l}(u,t)$. Since SPT_l has no cycles, $p_{LCA_l}(v,t) \in p_{LCA_l}(u,t)$. 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 picks 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. □

Algorithm 1 Decentralized search

function DECENTRALIZEDSEARCH(s, t)
  $\tilde{p}(s,t) \leftarrow \emptyset$
  $u \leftarrow s$
  append u to $\tilde{p}(s,t)$
  while $u \notin L(t)$ **do**
    $d_{min} \leftarrow \infty$
    $w \leftarrow u$
    for each v_i adjacent to u **do**
      if $d_{LCA}(v_i,t) < d_{min}$ **then**
        $d_{min} \leftarrow d_{LCA}(v_i,t)$
        $w \leftarrow v_i$
    $u \leftarrow w$
    append u to $\tilde{p}(s,t)$
  $p_{remain} \leftarrow p_L u, t$ excluding u
  append p_{remain} to $\tilde{p}(s,t)$
  return $\tilde{p}(s,t)$

The time complexity of the decentralized search procedure depends on the maximum degree and the diameter of the graph. As decentralized search takes at most $2\sigma_{max}$ steps to finish according to Theorem 1, for each step, the search checks at most δ_{max} neighbor vertices, where δ_{max} is the maximum vertex degree of the graph. For each neighbor, k LCA computations are required, and the time complexity for each LCA computation is $O(h)$[1], where h is the height of the indexed shortest path tree and $h \leq \sigma_{max}$. Therefore, 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 indexing, and space complexity for online query. The space required for offline indexing is $O(k\sigma_{max}n)$, where n is the 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. Therefore, $O(2\sigma_{max})$ space is required to store the approximated path. Combining them together, the online search space complexity of decentralized search is $O(k\sigma_{max})$.

B. 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 the 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 this difference. 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 the source and 5 be the target. The backward search can find a shorter path $p_{bwd} = (5, 12, 11)$ than the path $p_{fwd} = (11, 4, 0, 1, 5)$ by the forward search. The backward search finds a shorter path by exploring edge $(5, 12)$. However, this edge is invisible to the forward search because when the search traverses vertex 4, it prioritizes 0 than 12 due to the former one has a lower LDA distance to the target vertex 5.

It is not guaranteed that the forward search and the backward search will eventually meet at any intermediate vertex. 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 searches, where the only interaction of them is when the search results are combined, where the shorter paths are returned.

C. 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. 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 distances to target vertex 8, but their actual distances to vertex 8 are different, due to that the shortcut 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 tied vertex. This increases both the performance, i.e., chances to find a shorter path and the number of paths being found, and the cost of the search. By controlling the search space this way, decentralized search is able to achieve different levels of accuracy.

Two extreme ways to deal with ties are either only visiting one vertex, or visiting all vertices in the next step. The former one incurs the 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.

D. Extension on directed graphs

To treat directed graphs, for each landmark, the label of a vertex u need to store both the path from the landmark to the vertex, denoted as $p_{l \to u}$, and the path from the vertex to the landmark, denoted as $p_{u \to l}$. When the search is visiting vertex u, only out-edges of u need to be traversed. To calculate LCA distance from u to the target vertex t, $p_{l \to u}$ is used for u and $p_{t \to l}$ is used.

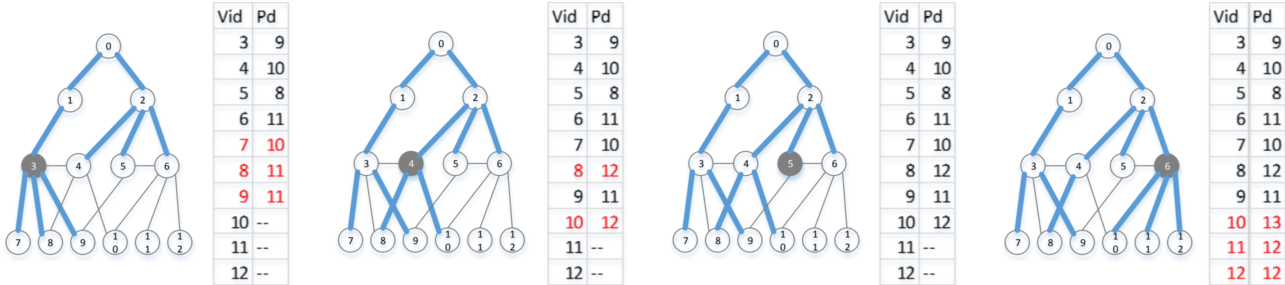
V. Greedy Index Construction

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

As the core of decentralized search is to iteratively find neighbor vertices that have shortest LCA distances to the target. From the point of view of a vertex u, if the indexed shortest path intersects with many indexed shortest paths of other vertices, the possibility that other vertices have a small LCA distance to u is going to be higher. With this intuition, we design our heuristic greedy index construction algorithm to store the shortest path with the highest "centrality", i.e., a path that intersects with most other shortest paths. 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, its computation cost is too high [22]. Therefore, we use degrees as an alternative and refer to the sum of degrees of vertices along a path as path degree, denoted by Pd.

Based on the path degree concept, our index construction procedure can be easily modified to index the shortest path with the highest path degree. Note that path degrees of shortest paths follow optimal substructures, 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 index the shortest path with the highest path degree, during BFS, suppose the search is visiting vertex u and reach its neighbor v with non empty $L(v)$, we perform a label update

[1] $O(h)$ is for simple online algorithm, off-line algorithms can achieve time complexity of $O(1)$ [3].



(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.

Fig. 2. Heuristic index construction pick shortest path with highest path degree during breadth first search

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 Algorithm 2.

Fig. 2 shows an example of how to greedily select the shortest path with the highest path degree during BFS. 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 happens to vertex 10 while traversing vertex 6.

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 $L(v_i) = \emptyset$ **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

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 trees for multiple landmarks, which can compromise the accuracy of online searches. A simple way to solve this problem is to prioritize shortest paths which overlap less with shortest paths that have already been indexed.

VI. DISTRIBUTED IMPLEMENTATIONS

To handle extremely large graphs and large numbers of queries, we implement decentralized searches on a distributed general graph processing platform, Powergraph [10]. As decentralized search has low online search space complexity and data dependencies 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 use Powergraph to partition the graph 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. Large volumes of queries can submit repeatedly, for which responses will be generated at high throughput.

A. Decentralized search vertex-program

Decentralized search can be implemented as vertex-programs in Gather-Apply-Scatter model used by Powergraph. Indexes are stored in a distributed way as vertex data. Each query instance contains the approximated path and the 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 phases. In the Gather phase, the LCA distance to the target vertex d_{LCA} is collected from each neighbor and accumulated with a *sum* function by finding the neighbor with the smallest d_{LCA} as a next step candidate. In the Apply phase, the candidate is appended to the approximated path $\tilde{p}$ and the termination condition is checked. If it is met, the result path will be recorded and the query will be terminated. Otherwise, the program will proceed to the Scatter phase to start a new vertex-program on the candidate vertex and pass on the query instance. Algorithm 3 shows the detailed algorithm.

The communication for the decentralized search happens during the Gather and the Scatter phase. In the Gather phase, the label of target vertex need to be passed to multiple machines, and the size is $O(k\sigma_{max})$. Each Gather function returns a d_{LCA} along side of its id. Therefore, only $O(k\sigma_{max})$ size of data is transferred in total. In the Scatter phase, communication happens when activating the next step candidate. The whole search instance, including approximated path and label of target vertex, needs to be transmitted. The total size is $O(k\sigma_{max})$. Since the search will take as much as $2\sigma_{max}$

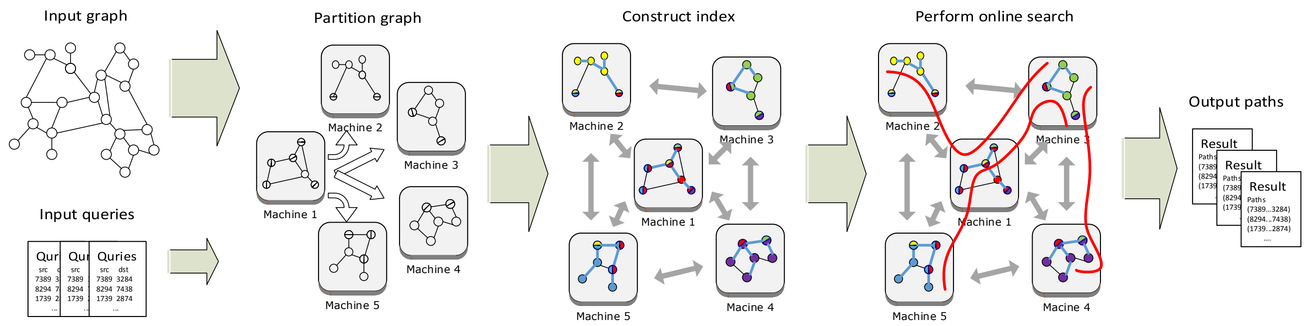


Fig. 3. A overview of distributed shortest path query processing system

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 $d_{LCA}(v_1,t) \le d_{LCA}(v_2,t)$ **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 $\tilde{p}(s,t)$
    $termination = true$
  else
    append v to $\tilde{p}(s,t)$
    $termination = false$
function SCATTER($L(t)$, $\tilde{p}(s,t)$, $termination$)
  if $\neg termination$ **then**
    Activate(v, $L(t)$, $\tilde{p}(s,t)$)

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. Depending on implementations, there may be output dependency, i.e. WAW, when output $\tilde{p}$.

B. Distributed tie breaking strategy

In our distributed implementation, the search instance clones itself into multiple search instances according to the tie breaking strategy. The problem is, as cloned search instances become independent in future steps, even one of them finds a shorter path than the others, it can hardly terminate other searches as such synchronizations are too costly in distributed environments. Therefore, a search may end up generating excessive number of child search instances. To overcome this problem, we pick one candidate at each step as a "main" candidate. For candidates that are not the "main" candidate, an extra termination condition is applied: In the following step, if the search cannot find a shorter path than expected, i.e. with a length shorter than $|\tilde{p}| + d_LCA$, the search will be discarded.

C. Prune LCA computation

A major part of the computation overhead of decentralized search is large number of LCA computations. It is possible to prune the number of LCA computation required at each step for decentralized search to reduce the overall computation overhead. Suppose the search is visiting vertex u, then $d_{LCA}(u,t)$ has already been calculated in previous steps. If a neighbor vertex 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 carried on as $d_{LCA_l}(v,t) > d_{LCA_l}(u,t)$. In practice, this principle can prune almost half of the total number of LCA computations of a search on average.

TABLE I
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 the number of vertices and edges in the largest WCC, and the average shortest distance $\overline{\sigma}$ of 100,000 vertex pairs.

VII. 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 in two aspects, the distance accuracy and the path diversity, in VII-C and VII-D respectively. We then show both overhead of index in VII-E. The throughput of our query processing system is shown in VII-F. Finally, we show the scalability of our system in VII-G.

A. Datasets

We evaluate our algorithm on 8 complex networks from different disciplines collected from Snap [15] and NetworkRepository [23] as shown in table I. To simplify our experiments, we treat them as undirected, un-weighted graphs and only use the largest weakly connected component of each graph.

B. Experiment settings

We evaluate our algorithms in both distributed setting, 20 Amazon EC2 m4.xlarge nodes, and centralized setting, one Cloudlab [21] c8220 server. Powergraph [10] and Snap [15] are used as platforms respectively. All algorithms are implemented in C++.

We use $DEGREE/h$ proposed in [18] as our landmark selection strategy. We randomly generate 100,000 queries by randomly choosing 1,000 vertices as source vertices and 100 target vertices for each source vertex as BFS is extremely slow for large graphs.

C. 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 under various landmark set size of 4 variations of bidirectional decentralized search with mixture of different tie breaking strategies and index construction approach. We also list the results of a state-of-the-art online search approach, TreeSketch [11]. Note that the online search in [20] is similar to TreeSketch with only different stop condition so we do not include it to the comparison. We run TreeSketch with random index construction strategy.

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 the 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 shown in Fig. 4 of full branch decentralized search with regular index is 17.4% to 45.7% lower than single branch decentralized search 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 shown in Fig. 4 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.

TABLE II
PATH DIVERSITY ($K = 2$)

Graph	Path cnt Decetralized Search	Path cnt TreeSketch	$\overline{r_p}$
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

TABLE III
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

D. 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 II shows the average number of 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.

Moreover, decentralized search is not restricted by labels of the source and the 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 II, the average r_p for full branch decentralized search on various graphs ranges from 0.301 to 0.501.

E. Overhead

The index and query overhead is shown in table III. In our implementation, both the label for each vertex and approximated paths are stored as vectors. And each vertex id is represented by 8-byte unsigned long. The size shown in table III is the sum of vector size of each vertex.

F. Throughput

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

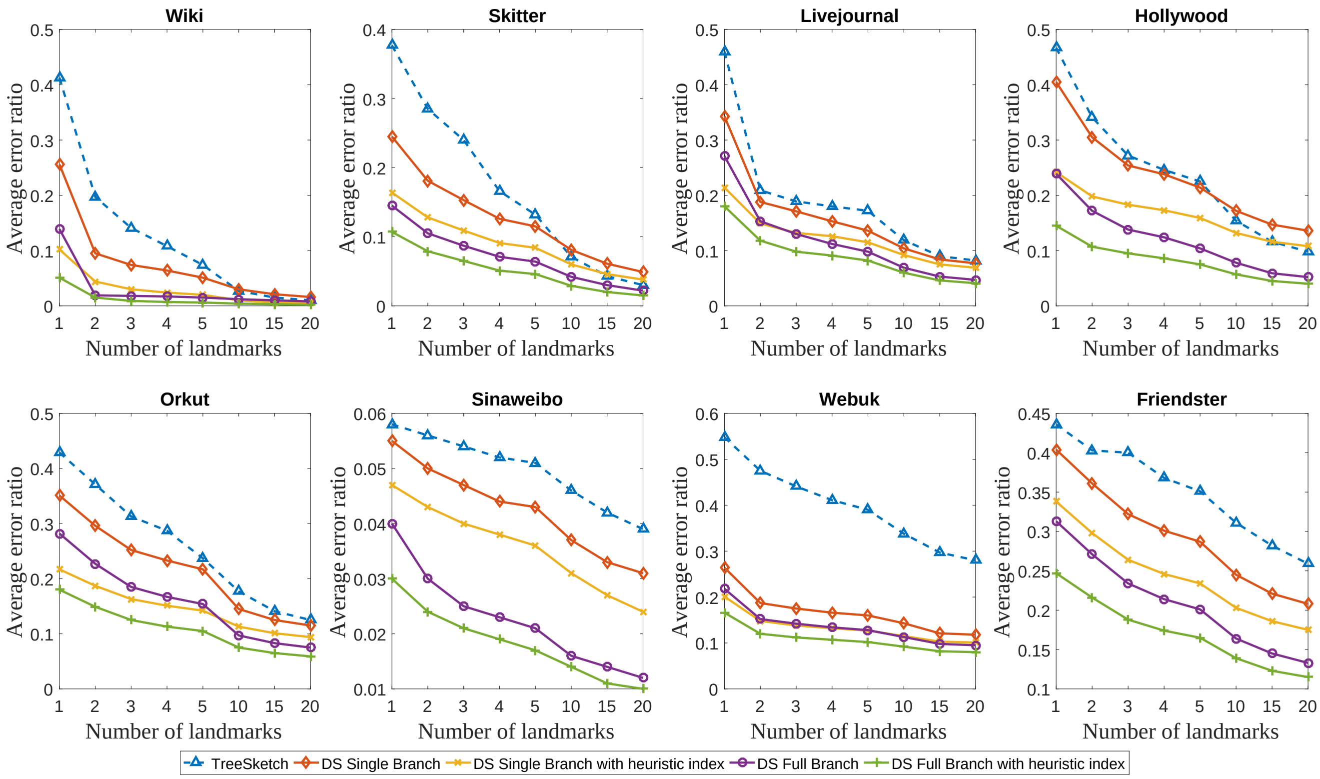


Fig. 4. The accuracy of the approximated distance of decentralized search compared to TreeSketch.

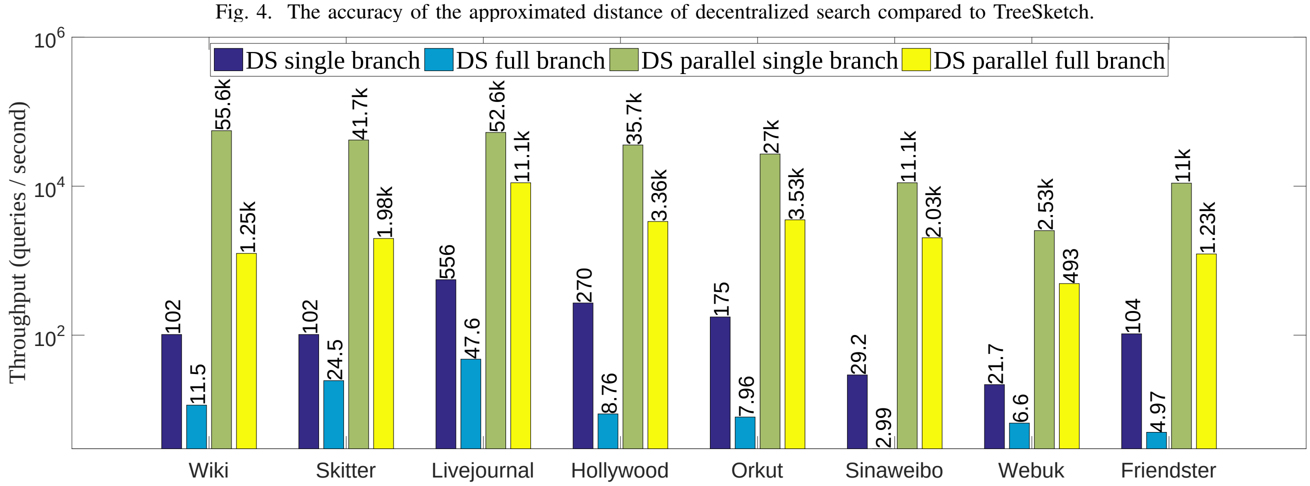


Fig. 5. The throughput of sequential and parallel version decentralized search.

single branch decentralized search and 74.7 to 679.1 times for full branch decentralized search.

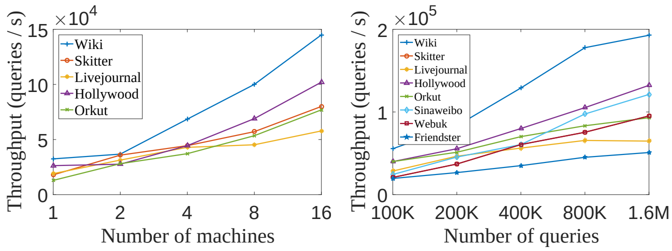
We can also see that due to the search duplicates itself for full branch decentralized search, the throughput is 3.2% to 30.4% of single branch for the sequential mode and 2.3% to 21.1% of single branch for the parallel mode.

G. Scalability

Since our algorithm is designed for large-scale networks, scalability is another major concern of our algorithm. We show how our algorithm performs as number of machines and queries increases. 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 throughput as number of machines increases. Results shown in Fig. 6(a) 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. Note that we do not have results for the 3 largest graphs as they are not able to fit into fewer than 8 machines.

We also show the trend of the throughput as number of queries running simultaneously increases. All the experiments



(a) Number of machines increases (b) Number of queries increases

Fig. 6. Throughput of decentralized search

are carried on 20 machines. We can see in Fig. 6(b) the constant growth of throughput as the number of queries running simultaneously increases. The growth of throughput slows 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 it for 100,000 queries.

VIII. 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 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 the number of machines and the number of parallel processed queries increases.

References

[1] 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, 2013.

[2] 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, 2012.

[3] M. A. Bender and M. Farach-Colton. The lca problem revisited. In *LATIN 2000: Theoretical Informatics*. Springer, 2000.

[4] M. Christoforaki and T. Suel. Estimating pairwise distances in large graphs. In *Big Data (Big Data), 2014 IEEE International Conference on*, 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, 2002.

[6] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the third ACM international conference on Web search and data mining*, 2010.

[7] 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.

[8] A. Frchette, F. B. Shepherd, M. K. Thottan, and P. J. Winzer. Shortest path versus multi-hub routing in networks with uncertain demand. In *INFOCOM, 2013 Proceedings IEEE*, pages 710–718, April 2013.

[9] 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, 2005.

[10] 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)*, 2012.

[11] 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, 2010.

[12] 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, 2012.

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

[14] J. Kleinberg. Complex networks and decentralized search algorithms. In *Proceedings of the International Congress of Mathematicians (ICM)*, 2006.

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

[16] M. Maier, M. Rattigan, and D. Jensen. Indexing network structure with shortest-path trees. *ACM Trans. Knowl. Discov. Data*, 2011.

[17] M. Newman. *Networks: an introduction*. OUP Oxford, 2010.

[18] 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, 2009.

[19] Z. Qi, Y. Xiao, B. Shao, and H. Wang. Toward a distance oracle for billion-node graphs. *Proceedings of the VLDB Endowment*, 7(1):61–72, 2013.

[20] 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*, 2014.

[21] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:*, 2014.

[22] 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*.

[23] 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.

[24] M. Shafiee and J. Ghaderi. A simple congestion-aware algorithm for load balancing in datacenter networks. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[25] 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*, 2014.

[26] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, 2003.

[27] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 2005.

[28] 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*, 2011.

[29] 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, 2007.

[30] J. Wang, L. Li, S. H. Low, and J. C. Doyle. Can shortest-path routing and tcp maximize utility. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 2049–2056. IEEE, 2003.

[31] 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, 2010.

[32] Y. Yang, J. X. Yu, H. Gao, and J. Li. Finding the optimal path over multi-cost graphs. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012.

[33] 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, 2010.