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

ABSTRACT

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

CCS Concepts

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

Keywords

Graphs, Shortest Paths, Decentralized Search

1. INTRODUCTION

Various types of graphs are commonly used as models for real-world phenomenon, such as online social networks, biological networks, the world wide web, among others. As their sizes keep increasing, scaling up algorithms to handle extreme size graphs with billions of vertices and edges remains a challenge that has drawn increased attention in recent years. Specifically, straightforward operations in graph theory are usually too slow or costly when they are applied to graphs at this scale. One problem that has drawn a lot of attention in the past is finding shortest paths in the network. This operation serves as the building block for many other tasks. For example, a natural application for road network is providing driving directions [1]. In social networks, such applications include social sensitive search [24], analyzing influential people [13], among others. Estimating minimum round trip time between hosts without direct measurement is another application in technology networks [21].

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

Our design is motivated by recent studies that combine both offline processing and online queries [17], [23], [3], [18], [12]. 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 [22], [9], [17], [11], [23], [18]are widely used for approximate shortest path/distance between vertices. Usually such algorithms select a small set of landmarks, and construct an index that consists of labels for each vertex, that store distances or shortest paths to landmarks. The approximation accuracy of landmark based algorithms heavily depends on the number of landmarks. Usually to achieve high accuracy, a relatively large set of landmarks is required, which lead to large preprocessing overheads. Indexes that can answer path queries usually have much larger space overhead than indexes that can only answer distance queries. One goal of our design, therefore, is to provide accurate results while still maintain low overhead for indexing.

Previous work also combined online search with index, but most efforts are solely based on the label of source and target vertices, which result in estimated paths only containing vertices belonging to the either label. This problem limits both the accuracy of the estimated path length compare to the exact ones, and the path diversity. So instead of estimating shortest path solely by examining vertices contained in labels of source and target vertices, our algorithm performs

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

DOI: 10.1145/1235

a heuristic search on each vertex it examined and pick the next vertex to examine by the distance information gathered during the examine. In this way, our approach explore edges that have not been indexed and examine vertices that does not include3d in labels of source and target vertices to achieve higher accuracy with limited index size. The heuristic search that we use is called decentralized search which was introduced by [14]. Here the "decentralized" means that the decision at each step is made based solely on local information which, in our context, is the labels of neighbor vertices.

We also introduce several optimizations to control the expansion of search space of decentralized search to balance between different level of accuracies and resources required for each search. With these optimizations, our algorithm becomes more versatile to meet various application needs without redoing the preprocessing.

On the core of decentralized search, all decisions made at each step relies on information stored in the index, not all the edges are equally important for estimating the shortest paths. To find out which edges should be stored in the index, we introduce a heuristic index construction algorithm that can increase the approximation accuracy by admitting only those most valuable edges. Experimental results shows that our approach not only increase accuracy of decentralized search under same number of landmarks, but also increase the accuracy of other methods based on landmark methods.

[talk about how the distributed search can help in reduce average query time and can deal with very large scale networks] 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 a short diameter, decentralized search can finish in very limited steps. Also, the algorithm does not need to mark which vertices have been visited like BFS or A* search, so only small space overhead is required for each search. This property makes it possible for large number of searches running in parallel without reaching the memory limit of machines. For example, in our experiments, we showed that millions of decentralized search can run in parallel. Furthermore, the average search time can be controlled at tens of microseconds or even shorter.

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

1.1 Contributions

Our contributions can be summarized as follows:

First, as an approximation algorithm, we combine decentralized search with landmark based indexes to achieve a high level of accuracy. We optimize the search space of decentralized search to achieve low online search overheads. By controlling the search space, our algorithm is able to achieve differentiated levels of accuracies.

Second, we propose a more effective heuristic approach for constructing index of the network that can improve the accuracy of the decentralized search without increasing preprocessing and online search overheads.

Third, we implement our algorithm in a distributed manner to handle extremely large graphs and perform the decentralized search in a parallel way to further reduce the average online query time for better scalability.

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

2. RELATED WORKS

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

Exact Approaches: Majority of the exact approaches are based on either 2-hop cover [6], [2] or tree decomposition [3], [25]. For the former one, finding optimal 2-hop covers is a challenging problem. [2] takes a different approach that solving 2-hop cover problem with graph traversals which has better scalability. [12] borrowed the highway concept from shortest path algorithms on road networks and construct a spanning tree as a "highway". [8] introduced an effective disk-based label indexing method based on independent set.

Approximate Approaches: Since the exact approaches do not scale very well, approximate algorithms are also well studied for large-scale complex networks. Landmark based algorithms are extensively studied for approximate shortest path/distance [22], [9], [17], [7], [16]. Although theoretical study of such algorithms does not reveal promising results [22], they usually work well in practice. [17], [20] 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 [11], [23], [18] which index shortest paths are proposed to alleviate this problem. Beside landmark based approaches, there are several other approximate approaches. [5] forms the shortest path problem as a learning problem to predicting pairwise distance. [26] maps vertices to low-dimension Euclidean coordinate spaces to answer distance queries in constant time.

Combining online search with indexes: [9] uses A* search for online query based on indexes constructed by landmark based algorithms. However, the cost of each A* search is still very high for large scale networks. [11] perform BFS on a sub-graph generated by the labels of source and target vertex. Although search space is greatly reduced, it still needs around seconds to handle graphs with millions of vertices.

3. PRELIMINARIES

3.1 Notations

In our problem, we consider a network modeled as a graph

$G = (V, E)$, which represents a vertex set V and edge set E. For a source node s and a target node t, we are interested in finding a path $p(s,t) = (s, v_1, v_2, ..., v_{l-1}, t)$ with a length of $|p(s,t)|$ close to the exact distance between s and t in the graph. Let $P(s,t)$ be the set of all paths from s to t. We can then define the distance between s and t as $d_G(s,t) = min_{p(s,t) \in P(s,t)} |p(s,t)|$.

We will focus on unweighted, undirected graphs in this paper. But all the ideas presented in this paper can be extended for weighted and/or directed graphs.

3.2 Landmark based indexes

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

3.3 The least common ancestor distance

When shortest paths have been indexed, the common ancestors closest to the source and target can be derived from the labels. This common ancestor is called the least common ancestor $c_l(s,t)$ of the source and target vertex. Since the shortest-path distance satisfies the triangle inequality, for an arbitrary pair of vertices s and t, we have the following bounds:

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

$$d_G(s,t) \geq max_{l \in L}|d_G(s, c_l(s,t)) - d_G(c_l(s,t), t)| \quad (2)$$

The upper bound, which we refer to LCA distance denoted by $d_{LCA}(s,t)$, can be used as an approximation of the distance of s and t. Note that if s and t are not connected with each other, there will not be a common l from $L(s)$ and $L(t)$.

4. DECENTRALIZED SEARCH FOR SHORTEST PATH APPROXIMATION

In this section, we are going to discuss how to perform decentralized searches based on the index structures to achieve a higher accuracy. We will also discuss how to optimize and control the search space of decentralized search to maintain low online search overheads.

4.1 Index guided decentralized search

We perform decentralized search on an indexed graph as follows. For a given pair of source and target vertex, the search start at the source vertex and set it as the currently visited vertex and append it to the approximated path. The search examines each neighbor of currently visited vertex, for each neighbor, the LCA distance to the target vertex is calculated. The neighbor with least LCA distance will be picked as the vertex to visit next and append it to the approximated path. This step continues until the search reach the target vertex. The algorithm is depicted in 1.

By calculating the LCA distance to the target from each neighbor, the search can explore edges that do not contained in the indexes. By choosing the neighbor with least LCA distance, the search does not constrained itself to vertices in the label of source and target vertices. For example in Fig. 1 from 15 to 11, by following the procedure of decentralized search, instead of finding a edge with both ends in labels of vertex 15: $(0, 3, 6, 9, 15)$ and 11: $(0, 2, 11)$, our approach find a edge $(9, 4)$ that can lead to a shorter path which is denoted by solid curved line.

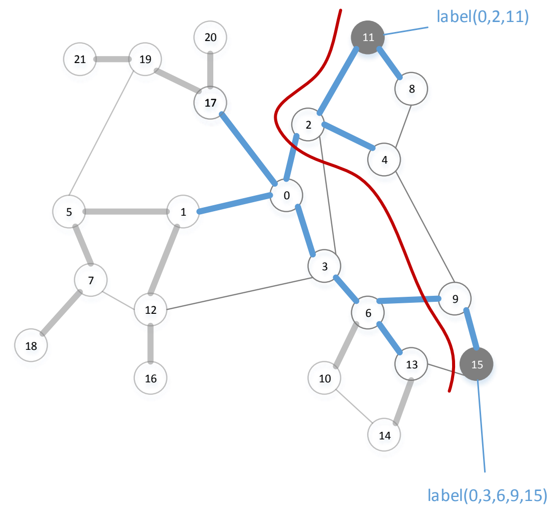


Figure 1: Decentralized search explores the edges that are not directly connected to the indexed shortest path. Edge $(4, 9)$ cannot be found by solely searching circles in the path by LCA distance.

Algorithm 1 Algorithm decentralized search

```
function DecentralizedSearch(G, s, t)
    p_appr. ← ∅
    u ← s
    p_appr. = p_appr. ∪ u
    while u ≠ t do
        d_min ← ∞
        w ← u
        for eachvadjecenttou do
            if d_LCA(v, t) < d_min then
                d_min ← d_LCA(V, t)
                w ← v
        u ← w
        p_appr. = p_appr. ∪ u
    return p_appr.
```

As long as the source vertex s and target vertex t are reachable from each other, decentralized search will terminate in as much as $2 * max_{u,v} d_G(u,v)$ steps, where $max_{u,v} d_G(u,v)$ is the diameter of the network. Too see this, for the LCA distance of an arbitrary pair of source and target vertex s and t, the following bound holds:

$$\begin{aligned} d_{LCA}(s,t) &\leq max_{l \in L}\{d_G(s, c_l(s,t)) + d_G(c_l(s,t), t)\} \\ &\leq max_{u,v} d_G(u,v) + max_{u,v} d_G(u,v) \end{aligned} \quad (3)$$

And at each step, suppose decentralized search is visiting vertex p, there is a neighbor of vertex q that is on the path indicated by LCA computation of p and t we denoted as q. We have $d_{LCA}(q,t) \leq d_{LCA}(p,t) - 1$. Since decentralized search always pick the neighbor with least LCA distance to the target, the LCA distance to the target at each step will decrease at least by 1. Therefore, decentralized search will terminate in at most $d_{LCA}(s,t)$ steps. According to equation 3, decentralized search for arbitrary pairs of reachable vertices will terminate in at most $2 * max_{u,v} d_G(u,v)$ steps.

However, terminating when the search reaches the target vertex is a valid but not an ideal stopping criterion. Since the label of each vertex stores the shortest path of each vertex to each landmark. And shortest path follows the optimal substructure, i.e. the path between any two vertices along the shortest path is also the shortest path of them. So a better terminate condition is to stop the search when reaching any vertex in the label of the target vertex. Since the path contained in the label of target vertex is already the the shortest path from this vertex to the target vertex due to the optimal substructure, we can directly concatenate it to the visited vertices to form a approximated path. With this terminate condition, required step for decentralized search is reduced from at most $2 * max_{u,v} d_G(u,v)$ to at most $max_{u,v} d_G(u,v)$.

The time complexity of Decentralized search depends on the several parameters of the graph. Decentralized search take $O(max_{u,v} d_G(u,v))$ steps to finish. For each step, the search need to check $O(max_u Degree(u))$ neighbors. For each neighbor, k LCA computations are required where k is the number of landmarks. Each LCA computations only takes constant times $O(1)$. So the time complexity of decentralized search depends on the average degree and the diameter of the graph. For Space complexity, first the landmark embedded in every node takes $O(kn)$ space. For each query, only $O(k)$ space is required to store the labels of target vertex and the vertex that is being examined.

4.2 Bi-directional search

In this section we show how to combine the bidirectional search and decentralized search together. Unlike bidirectional BFS, the goal for performing bidirectional search is not to reduce search space but to increase accuracy. A reversed search starting at target vertex and aim at source vertex may explore a different set of vertices and edges which may lead to a different path, sometimes with shorter length, to be found compared to the original search. For example in Fig. 2, the search starts from 20 to 7 can find a shorter path $p = (20, 17, 19, 5, 7)$ than the search starts at 7. Due to that 0 has a smaller LCA distance to 7 than 19, the edge $(19, 5)$ cannot be found by the search starts from 20.

In BFS, we expect two search will meet in some intermedia vertices that can be used as a new stop criterion which lead to reduced search space. But in decentrailzed search, there is no guarantee that two search will meet at any vertex except the source/target vertex. Since two search are driven by two distinct goals: finding next hop with least LCA distance to source/target vertex.

[explained by an example]

4.3 Handle ties

Tie happens frequently in decentralized search especially when the number of landmark is small. The tie here means during a step when decentralized search examining neighbors of currently visited vertex, there is not sufficient information in the index that can separate several neighbors, i.e. they have the same LCA distance to the target. For example in Fig. 3, to find path from 8 to 6, when traversing neighbors of vertex 8, both vertex 11 and 4 have the same LCA distance to vertex 6, but their actual distances to vertex 6 are different due to edges currently invisible to the decentralized search. Labels of each neighbor, on the other hand, provide no clue which one can lead to a shorter path.

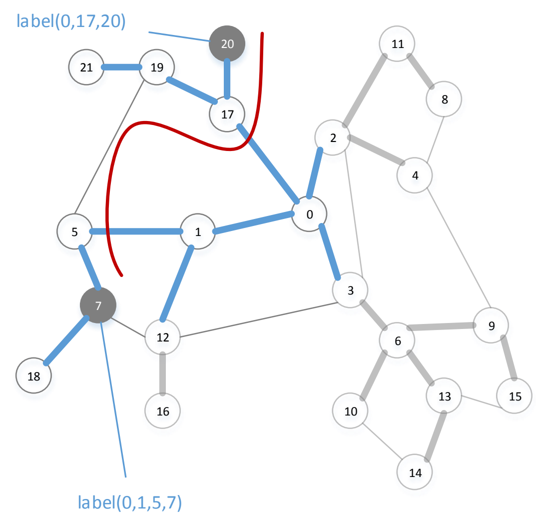


Figure 2: Bidirectional decentralized search can explore different set of edges which may found path at different length. Searches start at vertex 7 will lead to a path shorter than starting at 20 by taking advantage of the edge $(5, 19)$.

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

[the lca heuristic and effort/shorter path ratio.]

5. INDEX CONSTRUCTION

As the decentralized search relies on the index structure to find paths, the constructed indexes serve as a crucial factor for accurate approximations. Previous works have studied various landmark selection strategies which have a significant impact on the accuracy of online query. In our study, we observe that even with the same landmark set, choosing which shortest path from each vertex to landmark to be indexed also plays an important role for the accuracy of online search. In this section, we propose a heuristic index construction algorithm that can generate an index that can lead to higher accuracy for decentralized search.

5.1 Heuristic index construction algorithm

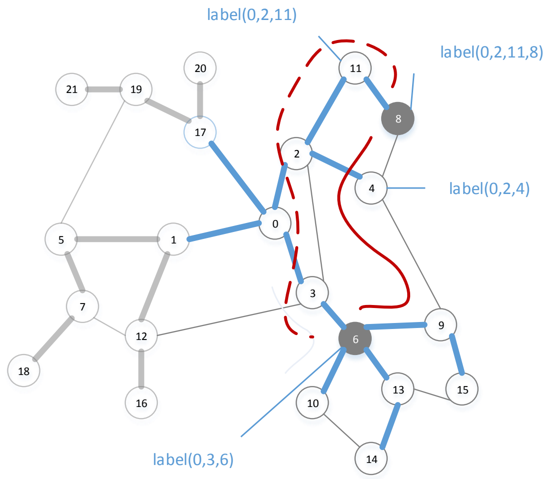


Figure 3: Tie happens during decentralized search. Although LCA distances are the same, selecting different neighbor will search different part of the graph which may lead to paths at different length.

[need improvement of the writing here] On the core of decentralized search is to find neighbor vertices that share LCA with target vertex at a higher level of the indexed shortest path tree. From the point of view of a vertex, a good shortest path from each landmark to be indexed should be the one that intersects with most of other shortest paths. So that the average chance to find a shorter path from other vertices to it is higher. For each vertex, we want to find a shortest path that has common ancestors with the largest number of other vertices to be indexed. So we are actually looking for the shortest path which intersects with most other paths to increase the chance for decentralized search to find a shorter path from other vertices to this one.

We propose a greedy heuristic to construct an index which can lead to better accuracy in online queries. For each vertex u, when generating a label for it to store a shortest path to landmark l, we want to store the one that has vertices with highest betweeness centrality, i.e. intersects with most other shortest paths. Since it is quite costly to calculate the betweeness centrality, we use the degree of each vertex along the shortest path as a approximation which is much easier to compute. We call this value the path degree Pd. Path degree can be easily calculated by summing up the degree of vertices along the path.

The calculation of path degree can be easily embedded into the BFS during index construction. The path degree of shortest path follows optimal substructure, i.e. if $(u, .., w, ..., v)$ has the highest path degree among all the shortest path from u to v, then the path degree of $(u, ..., w)$ is also the highest among all the shortest path from u to w. For each vertex, the shortest path with highest degrees can be calculated easily by comparing path degree of indexed shortest path of its parents and selecting the highest one. Such calculation can be done during breadth first search with little overhead by caching the path degree of the label of each vertex. Fig. 4 shows an example of how to greedily select shortest path

Algorithm 2 Algorithm heuristic index construction vertex program running on u

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

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.

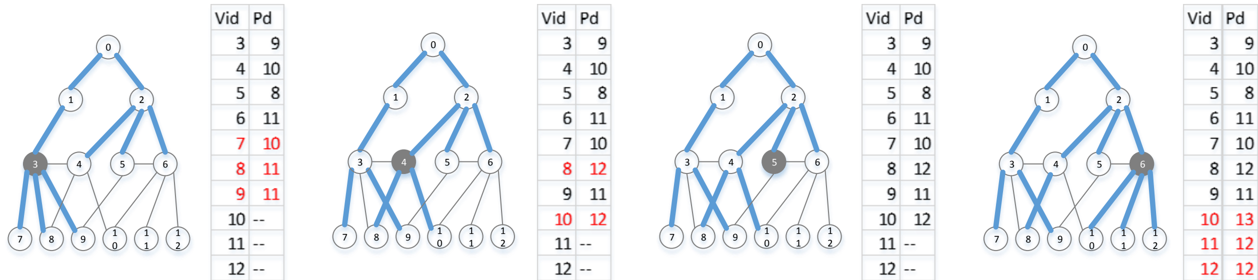
6. DISTRIBUTED IMPLEMENTATIONS

To handle extremely large graphs, we implement our algorithm in distributed settings. Due to that decentralized search does not require large volume of data to be cached during the search, it is well suitable for running in a parallel way. Our implementations can handle graphs with billions of edges and running millions of independent queries in parallel.

We build our algorithm on a distributed general graph processing platform - Powergraph[10]. An overview of our system is shown in Fig. **??**. Powergraph is designed to handle large-scale graphs with power law degree distributions efficiently by taking advantage of vertex-programs to factor computation over edges. Computation in Powergraph consists of a vertex-program running on a set of vertices. Each vertex-program consist of three phases: Gather, Apply and Scatter. During Gather phase, the *gather* and *sum* functions are used to collect and accumulate data from neighbors. The vertex data is updated in Apply phase by *apply* function through analysis on data collected from Gather phase. In the Scatter phase, *scatter* function is used to spread the new value to the graph and signal neighbors to start new vertex-programs. In this section, we are going to talk about details on how we implement our algorithm as vertex-programs.

6.1 Decentralized search vertex-program

Algorithm 3 shows the vertex-program of decentralized search. In Gather phase, for each query, LCA distance d_L calculated from labels L of each neighbor and target vertex is collected and accumulated by finding the neighbor with 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 4: Heuristic algorithm that index shortest path with highest path degree during breadth first search

smallest LCA distance. If a tie happens, according to our tie strategy, multiple neighbors may be returned as candidates. Since the decentralized search is a state-less search itself, vertex data does not need to be updated during the Apply phase. Instead, the program will append the candidate(s) to the approximated path $p_{appr.}$ and check whether the stop criterion is satisfied for each query. If so, the results will be recorded and this query will be terminated. Otherwise, program will proceed to Scatter phase to start new vertex-programs on next hop candidate(s) of each query and pass query information to them.

Algorithm 3 Algorithm decentralized search vertex program running on u

function GATHER($L(v)$, $L(t)$)
  return $d_L(v,t)$
function SUM($d_L(v_1,t)$, $d_L(v_2,t)$)
  return $min(d_L(v_1,t), d_L(v_2,t)$
function APPLY($p_{appr.}(s,t)$, $d_L(v,t)$)
  $p_{appr.}(s,t) += u$
  if termination condition meets **then**
    store $p_{appr.}(s,t)$
    $term$ = true
  else
    $term$ = false
function SCATTER($p_{appr.}(s,t)$, $term$)
  if $\neg term$ **then**
    Activate(v, $p_{appr.}(s,t)$)

6.2 Distributed tie breaking strategy

In the centralized version of our search algorithm, ties can be handled very easily. If a candidate cannot lead to the shortest approximate path at the next step among all candidates, it will be simply discarded. The situation becomes more complicated when implementing decentralized search in a distributed setting. In the distributed version of decentralized search, each candidate will start a new vertex program to approximate shortest path independently. There are no communications among different searches as this would be too costly to implement in a distributed setting. Hence, a candidate does not know the length of the path found by other candidates. Even if a candidate finds a shorter path, it does not have the ability to terminate searches at other candidates.

With this limitation, the search space are quite difficult to control and the search may end up with excessive overhead. A naive to alleviate this problem is to set a hard limit on the number of neighbors selected as candidates for next hop. But the algorithm needs to first decide which ones to select, and there is no available information to help making this decision. Another approach used in our implementation is that at each step, only one candidate will be chosen as a "main" candidate. For candidates that are not "main" candidates, an extra stop criterion will be applied. If it cannot find a shorter path than the one currently found, then the search will stop, and no result will be recorded. This will certainly reduce the chance a shorter path to be found. But it has a much smaller overhead and treats candidates more adaptively than the hard limit approach.

7. EVALUATIONS

We evaluate our algorithms in distributed settings with 8 Cloudlab [?] r320 nodes from Apt cluster. Each node has a 8-core 2.1GHz Xeon E5-2450 processor and 16 GB memory. All the experimental traffics are carried on a 10Gbps Ethernet. We use Powergraph [10] as the platform to implement the distributed version. We also implemented a centralized version with Snap [15] on a c8220 node with two 10-core 2.2GHz E5-2660 processors and 256GB memory. All algorithms are implemented in C++.

7.1 Datasets

We evaluate our algorithm on 9 graphs from different disciplines as shown in table 1 in ascending order on the number of edges. All graphs are complex networks that have power-law degree distribution and relatively small diameter. All datasets have at least millions of edges, the largest one has almost two billion edges. To simplify our experiments, we treat each graph as undirected, un-weighted graphs. We only use the largest weakly connected component of each graph that consist of more than 90% of vertices for all graphs. All datasets are collected from [15] and [19].

7.2 Approximation Accuracy

Dataset	Type	$\lvert V_{wcc}\rvert$	$\lvert E_{wcc}\rvert$	σ
Wiki-talk	Communication	2.4M	4.7M	
Skitter	Internet	1.7M	11.1M	
Baidu	Web	2.1M	17.0M	
Livejournal	Social	4.8M	43.4M	
Hollywood	Collaboration	1.1M	56.3M	
Orkut	Social	3M	117M	
Sinaweibo	Social	58.7M	261.3M	
Webuk	Web	39.3M	796.4M	
Friendster	Social	65M	1.8B	

Table 1: Datasets

We use the average approximation error as the measure of accuracy to evaluate the quality of estimated path. The approximation error of an estimated path is defined as follows:

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

To control the number of exact pairs of shortest path distance we need to find using BFS, we randomly choose 1,000 vertices of each graph as source vertices. For each source vertex, we randomly choose 100 vertices as target vertices. All the results in Fig. 5 and Fig. 6 are averaged with 100,000 queries.

We first compare the accuracy of decentralized search with LCA distance and tree-sketch [11]. For the decentralized search, we also compare accuracy with various tie breaking strategy to achieve different level of accuracies. The average approximation error for each approach is shown in Fig. 5. All experiments are carried on with 2 landmarks. From the figure, we can see that decentralized search achieves better accuracy for all graphs, and the performance gain varies from 8% to 64%. Various optimizations also have great impact on the accuracy. When all three optimizations have been used, the accuracy are even better than LCA distance computed from 20 landmarks for 7 out of 8 graphs (not including the graph friendster).

We then evaluate our heuristic index construction algorithm compared to the random index construction algorithm. We compared the results of decentralized search and LCA distance on both kinds of indexes. Fig. 6 shows the performance gain for a single landmark. As we can see that indexes created by greedy heuristic have a much smaller average error rate for both decentralized search and LCA computation. Due to that decentralized search examine more pairs of vertices than LCA computation, it has a better performance gain on 8 out of 9 graphs compared to LCA computation.

7.3 Path Diversity

With various tie breaking strategy, the algorithm can return various Fig. 3 shows average number paths returned by two tie breaking strategies. The first one is used in our centralized implementation, where all neighbors with same LCA distance are selected. The second one is used in our distributed implementation, where only one neighbor is marked as "main" candidate. We can see that the tie strategy we used in centralized settings can return far more paths than the strategy we used in distributed settings because of the larger search space. We don't have centralized version for graph friendster because it is too large to fit in one machine.

Graph	Path count(DS)	Path count(TS)	Ratio
Wiki	28.9	1.9	0.538
Skitter	24.1	2.4	0.603
Livejournal	30.8	1.9	0.566
Hollywood	9.9	2.6	0.618
Orkut	19.2	3.2	0.589
Sinaweibo	32.0	3.0	0.527
Webuk	704.1	2.0	0.664
Friendster	16.8	2.8	0.599

Table 2: Number of paths returned by decentralized search with different tie breaking strategies

7.4 Overhead

We study the preprocessing and online search overhead of decentralized search in this section. First we compare the preprocessing overheads of greedy index construction with random index construction. Then for online search overheads, to show the impact of increased search space, we compare the average search time with various optimizations enabled.

Fig. 7 shows the online search overhead in log scale with different optimizations. Note that results for bidirectional search and tie breaking are both with early termination on. We can see that early termination greatly reduces the search overhead from 71.6% to 99.7% by reducing the number of vertices being examined. The time overhead of bidirectional search shown in Fig. 7 is 2.42 to 4.38 times more than one direction search due to it has to found two paths instead of one path. The situation is a little complicated for tie breaking, since it is more related to the structure of each graph. For most graphs, the time overhead is 2.74 to 16.71 times more than one direction search. The worst case is 109.53 for graph mouse-gene which is the densest graph in our datasets, where too many candidates with same LCA distance exist at each step.

7.5 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 changes. Moreover, as the size of graph is expected to further increasing in the future, we also studies the impact of the size of the graph to our algorithm.

We first evaluate the search time when number of machines increasing. Results shown in Fig. 8 are averaged of 1,000,000 queries. We can see the trend is that the average run-time decreases as the number of machines increase. The average search time decreases fast when small number of machines are deployed and slow down when large number of machines are used. However there are some spikes in the curve, which might be caused by the hot-spot problem we mentioned before.

We then examine average search time as number of queries running in parallel increases. All experiments are carried on 20 machines in this part. We can see in Fig. 9 that the average search time quickly goes down when the number of queries is small. The reason is that there are some fixed overheads to start and stop the engine for a batch of queries. When the number of queries increases, the average

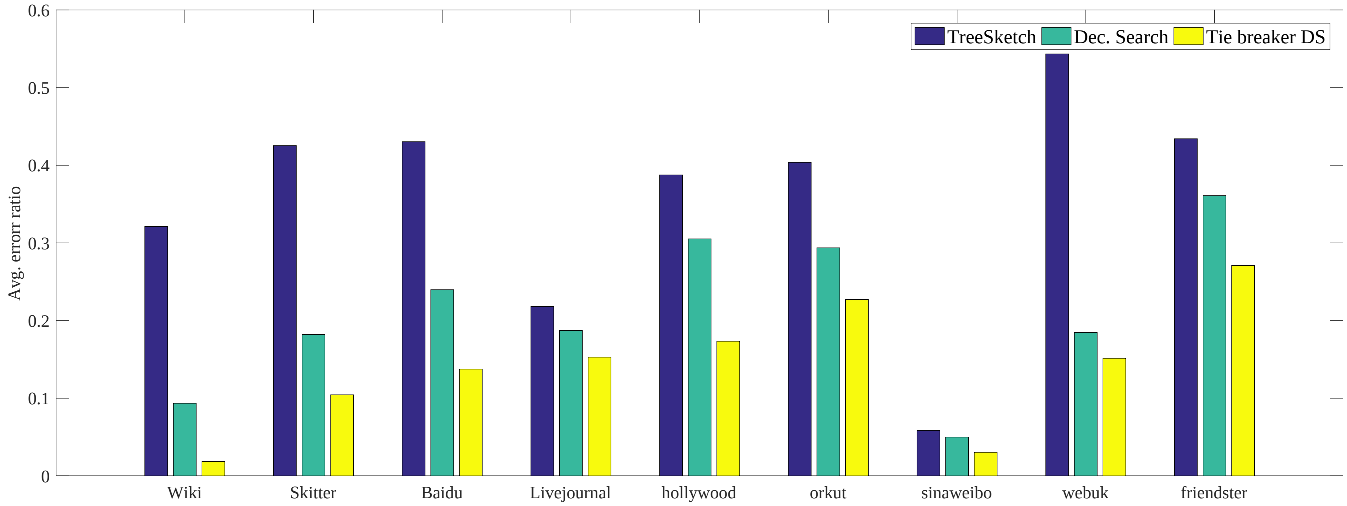


Figure 5: The accuracy of the approximated distance of decentralized search and various optimizations compared to LCA distance.

Graph	Path count(DS)	Path count(TS)	Ratio
Wiki	28.9	1.9	0.538
Skitter	24.1	2.4	0.603
Livejournal	30.8	1.9	0.566
Hollywood	9.9	2.6	0.618
Orkut	19.2	3.2	0.589
Sinaweibo	32.0	3.0	0.527
Webuk	704.1	2.0	0.664
Friendster	16.8	2.8	0.599

Table 3: Number of paths returned by decentralized search with different tie breaking strategies

overhead for each query will decrease. But when the number of queries becomes larger, the average search time begins to go up. Because too many searches will reach the memory and communication limits, that slow down the search speed.

In the last experiment, we evaluate the search time as the size of graph increases. Unlike BFS which needs to traverse neighbors of $O(n)$ vertices, decentralized search only needs to exam neighbors of $O(log(n))$ vertices for complex networks due to the small world property. We can clearly see in Fig. 10 that as the number of vertices increases, average search time of decentralized search does not follow the same trend. The graph friendster which have sixty-five millions vertices have average search time only 2.05 times more than graph mouse-gene which only have forty-three thousand vertices.

8. CONCLUSION

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

9. REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *Proceedings of the 10th International Conference on Experimental Algorithms*, SEA'11, pages 230–241, Berlin, Heidelberg, 2011. Springer-Verlag.

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

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

[4] D. S. Callaway, M. E. Newman, S. H. Strogatz, and D. J. Watts. Network robustness and fragility: Percolation on random graphs. *Physical review letters*, 85(25):5468, 2000.

[5] M. Christoforaki and T. Suel. Estimating pairwise distances in large graphs. In *Big Data (Big Data),*

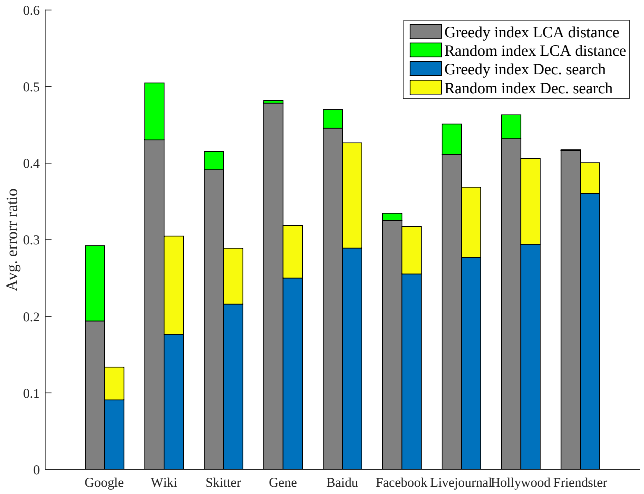


Figure 6: Impact of index quality on the accuracy of both decentralized search and label comparison.

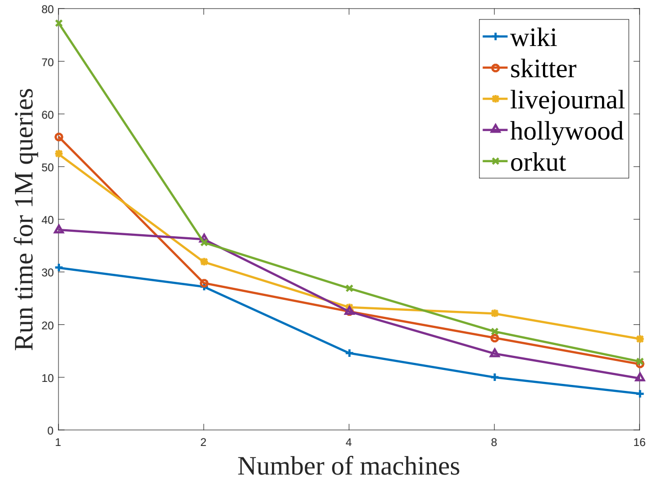


Figure 8: The average search time as the number of machines increase

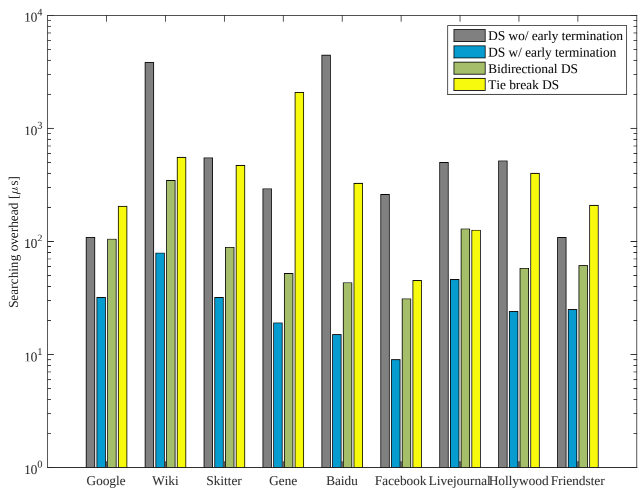


Figure 7: The average search time of decentralized search with various optimizations.

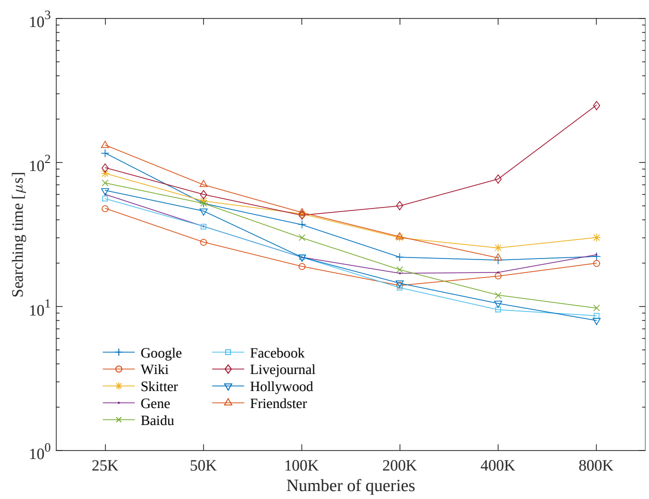


Figure 9: The average search time as the number of queries increase

2014 IEEE International Conference on, pages 335–344, Oct 2014.

[6] 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.

[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. 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.

[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, pages 156–165, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[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)*, pages 17–30, Hollywood, CA, 2012. USENIX.

[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, pages 499–508, New York, NY, USA, 2010. ACM.

[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*

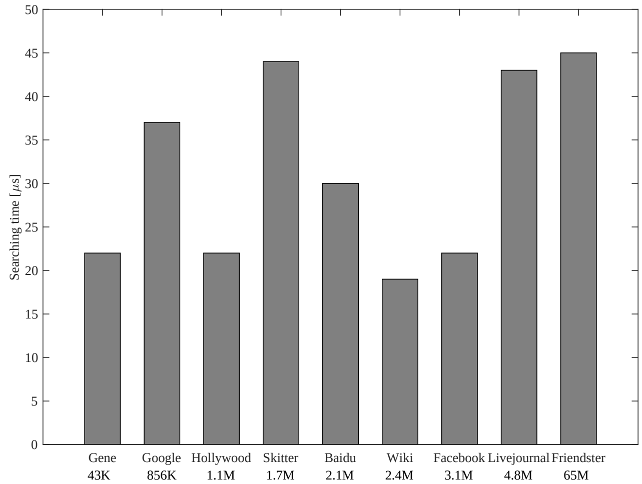


Figure 10: The average search time for different size of graphs

on Management of Data, SIGMOD '12, pages 445–456, New York, NY, USA, 2012. ACM.

[13] 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.

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

[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*, 5(3):15:1–15:25, Aug. 2011.

[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. 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.