A review of Point-to-Point Shortest Path/Distance Algorithms in Large-scale Complex Networks

Zheng Lu
Department of Electrical Engineering & Computer Science
The University of Tennessee
zlu12@utk.edu

ABSTRACT

Point-to-point shortest path/distance problem is building blocks for numerous applications. Solving this problem efficiently for large-scale complex networks is challenging. For complex networks with at least millions of vertices, classical BFS/Dijkstra traversal is too expensive to use. In this review, we provide a comprehensive state-of-the-art study in which we thoroughly expose the main focus of shortest path/distance algorithms in large-scale complex networks and network structures that inspired these algorithms as well as limitations of existing works.

1. INTRODUCTION

Introduced by the famous experiment conducted by Stanley Milgram, the small-world phenomenon, that most vertices can be reached from each other by a small number of hops, is found to be an fundamental characteristic in most real world networks. Although this property assure us the existence of such short paths between vertices in small world networks, due to the ever increasing number of vertices, this seemingly straightforward operation has become challenging. For example, for graph derived from LiveJournal social networks with 4 million vertices and 50 million edges [**?**], online BFS/Dijkstra traversals will take around a minute to finish on a single commercial computer, even for A* search which can significantly reduce the number of vertices visited, it still require to access an average of 20K vertices for each query [**?**].

Various networks which are commonly used in modern information technologies to model real-world phenomenon, such as online social networks (e.g., LinkedIn or Facebook), biological networks, road networks, etc have millions of vertices and billions of edges that even a cluster of machines can hardly deal with them. For example, in Figure 1, we plot the runtime of BFS running on a Amazon EC2 cluster with 10 machines for networks of various size. Networks are derived from Facebook, Twitter and LiveJournal respectively [**?**], [**?**]. We can see that even for a cluster of machines, a single BFS traversal takes around 3.5 seconds, and the time required for the traversal increases quickly as the scale of the network increases.

In this review, we are focusing on one of the most common variants of the shortest path/distance problem, to find a point-to-point shortest path/distance in a graph. This operation serves as the building block for many other tasks, and has many applications. For example, a natural application for road network is providing driving directions [**?**]. In social networks, such applications include social sensitive search [**?**], analyze influential people [**?**]. Estimating minimum round trip time between hosts without direct measurement is another application in technology networks [**?**].

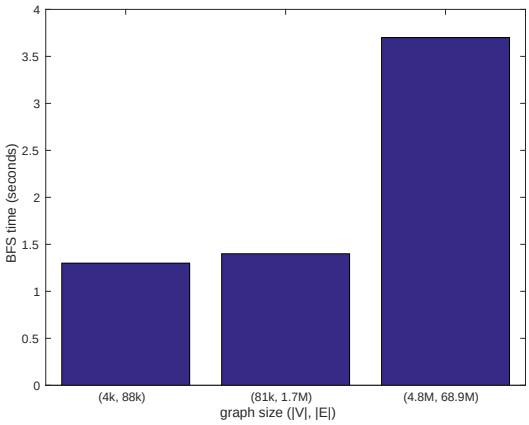

Figure 1: This figure shows the runtime of a single BFS on a Amazon EC2 cluster with 10 machines for graphs of various size.

Usually in these applications, point-to-point shortest path/distance problem will be solved repeatedly for different source/target vertex pairs. A straight forward solution to this problem is to apply online traversal like BFS/Dijkstra for each query. But clearly, this approach is very inefficient due to the scalability issue discussed above. Due to the nature of applications' needs, most point-to-point shortest path/distance algorithms will use preprocessing in order to speed up the following online searching operations. One obvious way for preprocessing is to perform all-pair shortest paths operation such as Floyd-Warshall and store the results for each pair of vertices. When a query arrives, only constance time is required to return the results. However, this approach is also unacceptable due to the $O(V^3)$ runtime and $O(V^2)$ storage space. To balance between preprocessing and online query has become a key challenge in this research area [**?**], [**?**], [**?**], [**?**], [**?**].

The point-to-point shortest path/distance problem has already been well studied in road networks, distance queries can be answered in less than a microsecond for the com-

plete USA road network [?]. Most of the methods on road networks take advantage of the spatial and planar-like properties a road network has [?]. Especially, Abraham et al. discovered that several fastest distance computation algorithms actually rely on a graph measure called highway dimension. However, these existing algorithms for road networks can hardly be applied to complex networks, due to the fact that the structure of most complex networks is nowhere near planar, and they usually do not have low highway dimensions like road networks. Actually, the locality of real-world complex networks is very poor, since the number of edges crossing clusters of the network is very large [?].

To further increase the scalability to handle large-scale complex networks, approximate methods have also been studied. Instead of returning an exact shortest path/distance between query vertex pairs, a near optimal path/distance is returned. Compared to online traversals for finding exact shortest path in seconds, it usually takes milliseconds for an algorithm to return an approximated shortest path [?], [?], and microseconds to approximate shortest path distance [?].

1.1 Problem Formulation

Our problem formulation is as follows: we investigate how to find exact or approximate point-to-point shortest paths/distance between any two vertices in extremely large networks. By extremely large, we are concerned with networks of million vertices, but in practice, such networks may even contain hundreds of millions of vertices. We are focusing on a specific kind of networks which is very common in real-world networks called complex network. They usually do not exhibit the same properties as simple ones such as road networks. For example, it is well known that these networks have their vertices' degrees conforming to the power law, and that vertices have relatively short distances between themselves. In most cases, the locality of these networks are very poor.

1.2 Contributions

In this review, we provide a comprehensive state-of-the-art study in algorithms on point-to-point shortest path/distance problems in large-scale complex networks. Existing algorithms usually contain two phases, distance calculation and online searching. After preprocessing, algorithms can return exact or estimated shortest path distance of two given vertices that can be used in applications where the underlying path is not important. The later one return an actual exact or approximate shortest path, and is essential for several applications require path information. Usually, online searching needs distance estimates to avoid large number of visited vertices, which is the main benchmark of these kind of algorithms.

1.3 Paper organization

The remaining of this review is organized as follows. In Section 2, we present several fundamental preprocessing methods being used by most of the algorithms and describes algorithms that calculate exact or estimated shortest path distance for any given vertex pair. We discuss online searching algorithms that finding actual paths in Section 3. We conclude the whole review in Section 4.

2. DISTANCE CALCULATIONS

As we discussed in previous section, applications of point-to-point shortest path problem usually solve large amount of queries repeatedly on same underlying network. Thus most of algorithms in this area allow preprocessing to speed up following online queries. A distance oracle (centralized) or distance labels (distributed) will be generated after preprocessing that can quickly answer shortest distance queries. Time and space for preprocessing usually should be limited to linear in the graph size with small constant [?]. For example, for a graph with several millions of vertices, typical preprocessing take seconds to thousands of seconds to finish, and consume hundreds of Megabytes to several Gigabytes of storage. With the help of data structures generated by preprocessing, shortest distance queries can be answered in microseconds and shortest path queries can be answered in milliseconds for network with millions of vertices compared to several seconds for online traversals without preprocessing. In this section, we discuss three fundamental preprocessing based methods that have been used by most of point-to-point shortest path algorithms for complex networks. We also introduce recent improvements and variants of these algorithms.

2.1 Preliminaries

In our problem, we consider a graph $G = (V, E)$, which represents a graph of vertex set V and edge set E. For a source vertex s and target vertex t, we are interested in finding a path p in the graph, which is an ordered sequence of n vertices, such that,

$$p = (v_0, v_1, ..., v_{n-1})$$

where $(v_i, v_{i+1}) \in E$. We can write that $|p| = n$ to denote that length of p is n. Given a pair of vertices, there may be many paths connecting them, which we denote as P. We can then define the distance between s and t as:

$$d_G(s,t) = min_{p \in P(s,t)} |p|$$

2.2 Landmarks

2.2.1 Basic Algorithm

Landmark based preprocessing algorithms are widely used for approximate shortest distance between vertices L. The preprocessing first choosing a small (constant) number of vertices as landmarks. Then for each landmark, the algorithm perform a BFS/Dijkstra traversal. After this operation, each vertex in the graph will be assigned a label with distance to each vertex in the landmark set. Upper and Lower bounds can be computed in constant time using these labels according to triangle inequality as follows:

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

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

where $d_G(s,l)$ and $d_G(l,t)$ are stored in the labels of s and t.

On the theoretic side, Thorup and Zwick proved that for any integer $k \geq 1$, with $O(kmn^{1/k})$ expected preprocessing time and $O(kn^{1+1/k})$ storage size, and a point-to-point shortest path query can be answered in $O(k)$ time [?]. The ratio of estimated distance and the exact should lie within $[1, 2k-1]$. For k = 1, we are actually computing APSP and store the results, which is very expensive as discussed in previous section. For k = 2, the estimate may be three times larger, that is unacceptable due to most complex networks has the small-world property. Although theoretical bounds of landmark based algorithms is not very promising, they works well in practice for large-scale complex networks [?]. Intuitively, labels of vertices encapsulates shortest path distance to the landmarks, which is part of the actual topology of the network, leading to accurate distance estimates.

2.2.2 Variance of Landmark based preprocessing

The choice of the landmark sets are important to give the distance estimations, M. Potamias et al. studied various strategies for choosing landmarks [?]. It turned out that the optimal landmark selection problem is NP-hard. Several landmark selection heuristics and different ways to use triangle inequality bound as distance estimates were evaluated in their paper. Although several advanced landmark selection strategies have better performances, simple heuristics such as choosing landmarks based on degree achieve high performance and low computation overhead at same time, that make them popular in this field.

The distance estimates of landmark approach heavily depends on the quality of label on each vertex, the more information label carries the better estimation results will be. In order to maximize information stored in each label, Akiba et al. proposed to doing a BFS from every vertex in the network [?]. The key contribution is that to reduce the large time and storage overhead brought by the huge number of BFS to an acceptable level, they prune during each BFS. The algorithm doing BFS in an iterative way, when previous BFS already explored a shortest path from current root to the visiting vertex, then this vertex will not be pruned. For example, suppose the algorithm already have an index L_{k-1} and is doing a BFS from v_k. Assuming this BFS reach a vertex u with distance δ. IF $d_{L_{k-1}}(v_k, u) = \delta$, the algorithm prune u, that is, (v_k, δ) will not be added to L_k as well as it will not be added to the priority queue of current BFS. Without all edge information incorporated into labels, this algorithm can calculate exact shortest distance instead of approximations. Even with pruning during BFS, the algorithm still take much longer time to preprocessing and have a relatively large label size.

Instead of only storing shortest path distance to each landmark as labels, Maier et al. stored shortest paths to landmarks for each vertex, they called it network structure indexes or NSI [?]. They use lowest common ancestor to further lower the upper bound given by triangle inequality by finding the smallest triangle along the path. In this way, not only the NSI improves the distance estimates, it can also return the related path without performing any online searching. The obvious drawback of this algorithm is that the size of the label is increased.

Landmark based algorithms have a well known drawback, that is, it is not good at approximating distance for two vertices near each other. To solve this problem, Qiao et al. proposed a local landmark scheme that choose landmarks based on each query [?].

Graph Embedding Methods: The embedding methods, which aim to transform graphs into alternative representations. By doing so, the algorithm can obtain significant speed-ups by embedding vertices into a different space and using a more efficient distance functions. Zhao et al. take a quiet different approach to estimates distance between vertices, they tried to map vertices in high dimensional graphs to positions in low-dimension coordinate spaces so that distance estimates between vertices can be easily calculated [?], [?]. To avoid large number of pairwise distance calculations, the algorithm first choose a small set of landmarks, and use global optimization algorithm to fix their coordinates. Then the algorithm add vertices to the coordinate space according to their distance to these landmarks in an iterative way. Note that although this algorithm use a different way than regular landmark based algorithms, it still use the same information, so the quality of distance is still the same as landmark based algorithms.

2.3 2-hop Cover

2.3.1 Basic Algorithm

While landmark based algorithms have been widely used to estimate shortest path distance, 2-hop cover is a different preprocessing algorithm that aim to finding exact shortest path distance.

The idea of 2-hop cover is pretty simple. For each vertex u, we assign a set $C(u)$ of vertices so that every pair of vertices (u, v) has at least one vertex $w \in C(u) \cap C(v)$ on a shortest path between the pair. The distance from each vertex to each vertices in its cover composed its label $L(u) = (w, d_G(u, w))_{w \in C(u)}$. It is easy to calculate distance from s to t with their labels $L(s)$, $L(t)$ using following formula:

$$d_G(s,t) = min\{\delta + \delta' | (s,\delta) \in L(u), (w,\delta') \in L(v)\}$$

The labels $L(u)$ is called a 2-hop cover. Although the idea of 2-hop cover is straight forward, but finding optimal 2-hop cover is a very challenging problem [?].

2.3.2 Variances of 2-hop Cover

Agarwal et al. use a similar approach as 2 hop cover [?]. In the preprocessing period, a subset of neighbor vertices was stored as vicinity of each vertex. During online searching period, algorithms searching intersects between vicinities of two vertices. Each intersect vicinity lies in a path between source vertex and target vertex. The algorithm will compare each path's length and return the shortest one as the estimation of shortest path between the vertex pair.

Highway structure has been widely used to facilitate finding shortest path in road networks. Intuitively for road networks any shortest path longer than certain hops contain a highway vertex. Jin et al. explore this highway structure in large sparse graphs, the algorithm build a tree structured highway

to connect source vertex and target vertex together [?]. This algorithm can be seen as a generalization of 2 hop cover, since it use a intermediate tree instead of a single vertex.

2.4 Tree Decomposition

2.4.1 Basic Algorithm

Tree decomposition based algorithms are another family of algorithms that finding exact shortest path distances. A tree decomposition of G is a pair (T, X), where T is a tree and $X = \{X_t | t \in V(T)\}$ is a family of subsets of $V(G)$, which are called bags, with the following properties: $(i) U_{t \in V(T)} X_t = V(G)$. (ii) For every $(u, v) \in E(G)$, there exists $t \in V(T)$ such that $u, v \in X_t$. (iii) For all $v \in V(G)$, the set $\{t | v \in X_t\}$ induces a subtree of T. The number of bags is denoted as b. The width of a tree decomposition (T, X) is $max_{t \in V(T)}\{|X_t| - 1\}$ which is refer to as w.

During preprocessing, the distance between any two vertices u, v contained in the same bag has been computed and stored as labels.

Let T_s and T_t be the subtrees of T induced by the bags including s and t. Let t_s and t_t be the vertices of T_s and T_t that is closest to the root of T. Let u be the lowest common ancestor of t_s and t_t. Any path from s to t must contain at least one vertex in the bag X_u. So the distance between s and t can be computed from the following equation:

$$d_G(s, t) = min_{v \in X_u}\{d_G(s, v) + d_G(v, t)\}$$

The $d_G(s, t)$ can be derived from this fact based on dynamic programming based on the labels created during preprocessing.

Since only the distance of every vertex pairs in same bags is computed and stored during preprocessing, so the overhead depends on the tree width of the tree decomposition of the network.

2.4.2 Tree decomposition based on core-fringe structure

Emerging in the area of finding shortest path in road networks, original tree decomposition based algorithms does not perform well on complex networks since the tree width is quite large which lead to unacceptable storage overheads. Akiba et al. leverage the core-fringe structure that most complex network shared [?], that is, most complex network has a relatively dense core and leaves a tree-like fringe. With this structure property, the tree width of complex networks outside the dense core is actually very small thus suitable for tree decomposition based algorithms.

2.5 Comparison

Table 1 compared the three algorithm families and highlight their limitations on large-scale complex networks. Although landmark based algorithm only return estimated results, due to its simplicity and lower computation and storage overhead, it is the most popular preprocessing algorithm in the literature. On the contrary, 2-hop cover and tree decomposition have larger overhead but can return exact results. Since the optimal size of 2-hop cover has not yet been solved, so there are no theoretical bound on the overheads. For tree decomposition, even some algorithms take advantage of the core-fringe structure that most complex networks have, the non-constant query time still make it less practical for applications with huge number of shortest path/distance queries.

3. PATH ONLINE SEARCHING

Online traversal algorithms such as BFS/Dijkstra require to exam $O(n)$ number of vertices in order to find the point-to-point shortest path for any two vertices, they are too expensive for large-scale complex networks. The main focus of the research body of online search algorithm is to achieve lower number of visited vertices for each query.

3.1 A* Search

Classical A* search has been used in Artificial Intelligence to find a solution in a huge search space by only searching a small subspace. The reason A* search avoid large number of visited vertices is that it uses estimates on distances to the destination to guide vertex selection in a search. Feasible distance estimates such as Euclidean distance usually do not exist for complex networks. So preprocessing is required to generate such distance estimates before online searching. Suppose the estimates from each vertex to target vertex following a function π called a potential function, then the reduced cost of an edge can be defined as following:

$$l_\pi(v, w) = l_G(v, w) - \pi(v) + \pi(w)$$

where the $l_G(v, w)$ is the original edge cost of the network. We say π is feasible if l_π is nonnegative for all edges. Only when π is feasible, we can use A* search to find the exact shortest path. Due to this constraint, when using landmark based preprocessing, only the lower bound of triangle inequality can be used as the distance estimates [?]. Since the $max_{l \in L}|d_G(s, l) - d_G(l, t)|$ was used as a feasible potential function, the choice of landmark set has a large compact on the performance of the algorithm in terms of numbers of vertices being visited.

Note that Although A* search which can significantly reduce the number of vertices visited, it still visit a relatively large number of vertices before finding a shortest path. For example, A* search require to access an average of 20K vertices in a graph of 4 million vertices for each point-to-point shortest path query [?].

3.2 Decentralized Search

J. Kleinberg studied the fundamental reason in small world experiments, why people find the path via a very simple "greedy" heuristic: each person forwards the message to a neighbor who is closest to the target [?]. Such "greedy" heuristic was called decentralized search in the paper, since only local information was required to forward the parcel. It turns out that only some of small world networks have such property that will allow efficient navigation through decentralized search. In decentralized search, landmark based preprocessing is used to provide knowledge of distance between vertices, and the algorithm use it to guide the searching.

Algorithm family	return type	computation overhead	storage overhead	query time
Landmark	approximate	$O(\|L\|\|E\|)$	$O(\|L\|\|V\|)$	$O(1)$
2-hop Cover	exact	no theoretical bound	$O(\|V\|\sqrt{\|E\|})\star$	$O(1)$
Tree Decomposition	exact	$O(\|V\|\|E\| + bw^2)$	$O(\|E\|)$	$(w^5 log^3 \|V\|)$

$\star$ This is an approximated optimal value from [**?**].

Table 1: Comparison of preprocessing algorithms

The quality of the path found by decentralized search, that is, the number of visited vertices, depends on the quality of this distance information, more accurate distance information can lead to shorter path being found.

One characteristic of decentralized search is that the algorithm does not need to maintain a priority queue, so fewer memory overhead is required for each searching which allow more queries to be running at the same time. Further more, if paths to landmarks are stored instead of only distances, then the number of visited vertices for any query of decentralized search is bounded by two times of diameter of the graph. Considering that most complex networks have relatively short diameter, decentralized algorithm has much smaller overhead than other online searchings.

3.3 Subgraph BFS/Dijkstra Traversal

Although online BFS/Dijkstra traversals are extremely expensive in large-scale complex networks, several algorithms have been proposed to use them in a much smaller subgraph generated from the original graph.

Instead of only storing distance to each landmark as labels, Tretyakov et al. store shortest path to each landmark, which is called sketch of a vertex [**?**]. When query about the shortest path between each pair, sketches of two vertices are pulled, and they perform a BFS on the subgraph induced by the union of two sketches, to find the shortest path on this subgraph and use it as the approximation of the shortest path of two vertices. Gubichev et al. adopt a very similar way, they use bidirectional BFS to speed up the algorithm and returning a list of paths in increasing length order [**?**].

4. CONCLUSIONS

In this paper, we thoroughly studied state-of-the-art algorithms of point-to-point shortest path/distance algorithms in large-scale complex networks. Algorithms in this field usually consist of preprocessing and online searching.

Preprocessing are used to generate data structures that can be used to speed up the following online searching. We classify previous works into three preprocessing categories and studied their differences. Among three preprocessing algorithms, landmark based algorithms have the best scalability but can only return estimated results. 2-hop cover and tree decomposition do not work well for large-scale complex network due to large computation and storage overhead. Tree decomposition algorithms also have non-constant query time.

For online searching, we studied several different algorithms and evaluate them in terms of quality of returned path and number of visited vertices. A* search can give exact shortest paths, but need to visit large number of vertices for each query. Decentralized search achieve higher level of scalability due to it only require local information and does not need to maintain a priority queue, but usually only return estimated results. Subgraph BFS/Dijkstra traversal also has good scalability due to the size of subgraph is usually very small.

5. ACKNOWLEDGMENTS

The author would like to thank Dr. Cao for his comments and suggestions which have significantly improved the readability of this review. The author also would like to thank Yunhe and Lipeng for their inspiring talks.