Proxies for Shortest Path and Distance Queries

(Extended Abstract)

Shuai Ma[*†], Kaiyu Feng[‡], Jianxin Li[*†], Haixun Wang[§], Gao Cong[‡], Jinpeng Huai[*†]
[*]SKLSDE Lab, Beihang University, Beijing, China
[†]Beijing Advanced Innovation Center for Big Data and Brain Computing, Beijing, China
[‡]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[§]Facebook Inc., USA
{mashuai, lijx, huaijp}@buaa.edu.cn, {kfeng002@e., gaocong@}ntu.edu.sg, haixun@gmail.com

***Abstract*—This study investigates a light-weight data reduction technique for speeding-up shortest path and distance queries on large graphs. We propose a notion of *routing proxies* (or simply proxies), each of which represents a small subgraph, referred to as deterministic routing areas (DRAs). We show that routing proxies hold good properties for speeding-up shortest path and distance queries, and there exists a *linear-time* algorithm to compute routing proxies and their corresponding DRAs. Finally, we discuss the experimental results, and verify that our solution is a general technique for reducing graph sizes and speeding-up shortest path and distance queries, using real-life large graphs.**

I. INTRODUCTION

We study the *node-to-node shortest path* (*distance*) problem on large graphs: given a weighted undirected graph $G(V, E)$ with non-negative edge weights, and two nodes of G, the source s and the target t, find the shortest path (distance) from s to t in G. The Dijkstra's algorithm with Fibonacci heaps runs in $O(n \log n + m)$ due to Fredman & Tarjan [2], where n and m denote the numbers of nodes and edges in a graph, respectively, which remains asymptotically the fastest known solution on arbitrary undirected graphs with non-negative edge weights. However, computing shortest paths and distances remains a challenging problem, in terms of both time and space cost, on large-scale graphs. Hence, various optimizations have been developed to speed-up the computation.

To speed-up shortest path and distance queries, our approach is to use *representatives*, each of which captures a set of nodes in a graph. The task of finding a proper form of representatives is *nontrivial*. Intuitively, we expect representatives to have the following properties. (1) A small number of representatives can represent a large number of nodes in a graph; (2) Shortest paths and distances involved within the set of nodes being represented by the same representative can be answered efficiently; And, (3) the representatives and the set of nodes being represented can be computed efficiently.

The full version of this extended abstract appears in [3].

II. ROUTING PROXIES

We first propose *routing proxies* and *deterministic routing areas* (DRAs) to capture the idea of representatives and the set of nodes being represented, respectively.

Proxies. Given a node u in graph $G(V, E)$, we say that u is a *routing proxy* (or simply *proxy*) of a set of nodes, denoted by A_u, if and only if:

(1) node $u \in A_u$ is reachable to any node of A_u in G,

(2) all neighbors of any node $v \in A_u \setminus \{u\}$ are in A_u, and

(3) the size $|A_u|$ of A_u is equal to or less than $c \cdot \lfloor \sqrt{|V|} \rfloor$, where c is a small constant number, such as 2 or 3.

Here condition (1) guarantees the connectivity of subgraph $G[A_u]$, condition (2) implies that not all neighbors of proxy u are necessarily in A_u; and condition (3), called *size restriction*, limits the size of A_u of proxy u. Intuitively, one simply checks the graph by removing u from G and its newly created connected components (CCs), and a proxy of u is a union of such CCs whose total number of nodes is at most $c \cdot \lfloor \sqrt{|V|} \rfloor - 1$.

Deterministic routing areas. A node u may be a proxy of multiple sets of nodes $A_u^1, \ldots, A_u^k$. We denote as A_u^+ the union of all the sets of nodes whose proxy is u , *i.e.,* $A_u^+ = A_u^1 \cup \ldots \cup A_u^k$, and A_u^i $(i \in [1, k])$ is said a component of A_u^+.

We refer to the *subgraph* $G[A_u^+]$ as a deterministic routing area (DRA) of proxy u. Intuitively, DRA $G[A_u^+]$ is a *maximal* connected subgraph, union of a set of CCs, that connects to the rest of graph G through proxy u only. That is, for any node v in $G[A_u^+]$ and any node v in the rest of graph G, u must appear on the shortest path $path(v, v')$.

Maximal proxies. We say that a proxy u is *maximal* if there exist no other proxies u' such that $u' \neq u$ and $A_u^+ \subset A_{u'}^+$.

Trivial proxies. We say that a maximal proxy u is *trivial* if A_u^+ contains itself only, *i.e.,* $A_u^+ = \{u\}$.

Equivalent proxies. We say that two proxies u and u' are *equivalent*, denoted by $u \equiv u'$, if $A_u^+ = A_{u'}^+$.

Trivial proxies only represent themselves. Hence, we only focus on non-trivial maximal proxies (or simply proxies).

We then give an analysis of the properties of DRAs and their routing proxies, and show that they indeed hold the desired properties of representatives discussed in Introduction.

Proposition 1: *Given any two nodes v, v' in the* DRA $G[A_u^+]$ *of proxy u in graph G, (1) the shortest path in $G[A_u^+]$ is exactly the one in the entire graph G, and (2) it can be computed in linear time in the size of G.* □

Corollary 2: *Given any two nodes v, v' in the* DRA $G[A_u^+]$ *of proxy u in graph G, (1) the shortest distance $dist(v, v')$ in $G[A_u^+]$ is exactly the one in the entire graph G, and (2) it can be computed in linear time in the size of G.* □

Proposition 3: *Given two nodes v and u with two distinct proxies x and y, respectively, in graph G, the shortest path from v to u is the concatenation of three paths,* i.e., $path(v, x)/path(x, y)/path(y, u)$. □

Corollary 4: *Given two nodes v and u with two distinct proxies x and y, respectively, in graph G, the shortest distance* $dist(v, u) = dist(v, x) + dist(x, y) + dist(y, u)$. □

Theorem 5: *Finding all* DRA*s, each associated with one maximal proxy, in a graph can be done in linear time.* □

III. Query Answering with Routing Proxies

Based on the previous analyses, we present a framework for speeding-up shortest path and distance queries, consisting of two modules: *preprocessing* and *query answering*.

1. Preprocessing. Given graph $G(V, E)$, the preprocessing module executes the following.

(1) It first computes all DRAs and their maximal proxies with a linear algorithm, referred to as computeDRAs [3].

(2) It then pre-computes and stores all the shortest paths and distances between any node in a DRA and its proxy. To support shortest distance queries, for each node in a DRA, we store its proxy u, its distance to u and the component of A_u^+ to which it belongs, and to support shortest path queries, we further keep the shortest paths from proxy u to all nodes in the DRA.

(3) It finally computes the reduced subgraph G' by removing all DRAs, but keeping their proxies, from graph G.

2. Query answering. Given two nodes s and t in graph $G(V, E)$ and the pre-computed information, the query answering module executes the following.

(1) When nodes s and t belong to the same DRA $G[A_u^+]$ with proxy u such that $A_u^+ = A_u^1 \cup \dots A_u^h$.

If s and t further fall into the same component A_u^i ($i \in [1, h]$), it invokes the Dijkstra's algorithm on the subgraph $G[A_u^i]$ to compute the shortest path and distance between s and t. Otherwise, it simply returns $path(s, u)/path(u, t)$ or $dist(s, u) + dist(u, t)$ in constant time.

(2) When s and t belong to two DRAs $G[A_{u_s}^+]$ and $G[A_{u_t}^+]$ with proxies u_s and u_t, respectively.

As shown in Section II, we know that $path(s, t) = path(s, u_s)/\ path(u_s, u_t)/\ path(u_t, t)$, in which $path(s, u_s)$ and $path(u_t, t)$ are already known. Hence, it simply invokes an algorithm (*e.g.,* ARCFLAG [4], TNR [1], AH [5]) on the reduced graph G' for computing $path(u_s, u_t)$. Similarly, it computes $dist(s, t) = dist(s, u_s) + dist(u_s, u_t) + dist(u_t, t)$.

IV. Experimental Study

We conduct an extensive experimental study, using real-life datasets: (1) *co-authorship networks*, which extracted a co-authorship graph from DBLP (SNAP, http://snap.stanford.edu/data) and (2) *road networks* (DIMACS, http://www.dis.uniroma1.it/challenge9/download.shtml). Note that TNR is designed for road networks and it is very inefficient for TNR to preprocess dense graphs such as DBLP (it took more than 1 week to finish the preprocessing). Hence, we remove all nodes whose degrees are higher than 14, and choose the largest connected component in the remaining graph, referred to as DBLP14. Also AH requires the coordinate information to answer shortest path or distance queries, not available in both DBLP and DBLP14.

Experimental results. (1) In sparse graphs whose average degree is less than 4, about 1/3 nodes in the graph are captured by proxies, leaving the reduced graph about 2/3 of the input graph. In some cases such as DBLP14, about 2/3 nodes in the graph are captured by proxies, leaving the reduced graph about only 1/3 of the input graph.

(2) The performance of proxies and DRAs is sensitive to the density and degree distribution of graphs, and they perform well on graphs following the power law distribution. Meanwhile, for a given degree distribution, DRAs tend to capture less nodes when the average degree is higher.

(3) Proxies and their DRAs benefit existing shortest path and distance algorithms in terms of time cost. They reduce (20%, 1%) time for (ARCFLAG, AH), and have comparable running time for TNR on road networks; They reduce (4%, 49%) time for (ARCFLAG, TNR) on the co-authorship network DBLP14.

(4) Existing shortest path and distance algorithms also benefit from using proxies in terms of space overhead. Proxy+TNR can handle the road network C-US while TNR cannot. Moreover, (Proxy+ARCFLAG, Proxy+TNR, Proxy+AH) incur less space overhead than their counterparts, and are about (38%–68%, 72%–92%, 82%) of (ARCFLAG, TNR, AH), respectively.

V. Conclusion

We have studied how to speed-up (exact) shortest path and distance queries on large weighted undirected graphs. To do this, we have proposed a light-weight data reduction technique, a notion of proxies such that each proxy represents a small subgraph, referred to as DRAs, and proxies and DRAs can be computed efficiently in linear time. We have also verified, both analytically and experimentally, that proxies significantly reduce graph sizes and improve efficiency of existing methods.

Acknowledgments

This work is supported in part by 973 Program 2014CB340300, NSFC U1636210 & 61421003 & 61322207, Special Funds of Beijing Municipal Science & Technology Commission, and MSRA Collaborative Research Program.

References

[1] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. Werneck. Route planning in transportation networks. In *MSR-TR-2014-4*. 2014.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[3] S. Ma, K. Feng, J. Li, H. Wang, G. Cong, and J. Huai. Proxies for shortest path and distance queries. *TKDE*, 28(7):1835–1850, 2016.

[4] R. H. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm. Partitioning graphs to speedup Dijkstra's algorithm. *ACM Journal of EA*, 11:1–29, 2006.

[5] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou. Shortest path and distance queries on road networks: towards bridging theory and practice. In *SIGMOD*, 2013.