Proxies for Shortest Path and Distance Queries

(Extended Abstract)

Shuai Ma*†, Kaiyu Feng‡, Jianxin Li*†, Haixun Wang§, Gao Cong‡, Jinpeng Huai*†

* Beijing Advanced Innovation Center for Big Data and Brain Computing, Beijing, China

† SKLSDE Lab, Beihang University, Beijing, China

‡ School of Computer Engineering, Nanyang Technological University, Singapore

§ Facebook Inc., USA

{mashuai, lijx, huaijp}@buaa.edu.cn {kfeng002@e., gaocong@}ntu.edu.sg haixun@gmail.com

***Abstract*—Computing shortest paths and distances is one of the fundamental problems on graphs, and it remains a *challenging* task today. This article investigates a light-weight data reduction technique for speeding-up shortest path and distance queries on large graphs. To do this, 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 first show that routing proxies hold good properties for speeding-up shortest path and distance queries. Then we design a *linear-time* algorithm to compute routing proxies and their corresponding DRAs. Finally, we experimentally 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.**

1. 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. We allow the usage of auxiliary structures generated by preprocessing, but restrict them to have a moderate size (compared with the input graph). In this work, we are only interested in shortest paths and *exact* shortest distances on *large* graphs.

Finding shortest paths and distances is one of the fundamental problems on graphs, and has found its usage as a building block in various applications, *e.g.,* measuring the closeness of nodes in social networks and Web graphs [7], [12], [15], finding the distances between physical locations in road networks [18], and drivers' routing services [8].

Algorithms for computing shortest paths and distances have been studied since 1950's and still remain an *active* area of research. The classical one is Dijkstra's algorithm [3] due to Edsger Dijkstra. Dijkstra's original algorithm runs in $O(n^2)$ [4], and the enhanced implementation with Fibonacci heaps runs in $O(n \log n + m)$ due to Fredman & Tarjan [5], where n and m denote the numbers of nodes and edges in a graph, respectively. The latter remains asymptotically the fastest known solution on arbitrary undirected graphs with non-negative edge weights [16].

However, computing shortest paths and distances remains a challenging problem, in terms of both time and space cost, for large-scale graphs such as Web graphs, social networks and road networks. The Dijkstra's algorithm [5] is not acceptable on large graphs (*e.g.,* with tens of millions of nodes and edges) for online applications [12]. Therefore, a lot of optimization techniques have been recently developed to speed up the computation [2], [6], [9], [12], [13], [14], [15], [17], [18].

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, however, *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.

Contributions & Roadmap. We develop a light-weight data reduction technique for speeding-up shortest path and distance queries on large weighted undirected graphs.

(1) We first propose a notion of *routing proxies* (or simply proxies), each of which represents a small subgraph, referred to as *deterministic routing areas* (DRAs) (Section **??**). We also give an analysis of routing proxies and DRAs, which shows that they hold good properties for speeding-up shortest path and distance queries.

(2) We then develop a *linear-time* algorithm for computing deterministic routing areas along with their maximal routing proxies (Section **??**). This makes our solution a light-weight technique that is scalable to large graphs.

(3) Using real-life large road and co-authorship (sparse) graphs, we finally conduct an extensive experimental study (Section **??**). We find that (a) on average 1/3 nodes in these graphs are captured by routing proxies, leaving the reduced

graph about 2/3 of the original input graph for both road and co-authorship graphs, (b) proxies can be found efficiently, *e.g.,* they can be found in at most 6 seconds for co-authorship graphs and less than half an hour for the largest road graph, and (c) the reduced graph incurs only about 70% space overhead of the original input graph on average. Moreover, (d) using proxies benefits existing shortest path and distance algorithms in terms of time cost, *e.g.,* it reduces around 30%, 20% and 1% time for bidirectional Dijkstra [9], ARCFLAG [11] and AH [20] on road graphs, respectively, and 49%, 4% and 49% time for bidirectional Dijkstra, ARCFLAG and TNR [1] on co-authorship graphs, respectively, and (e) existing shortest path and distance algorithms also benefit from using proxies in terms of space overhead, *e.g.,* it reduces about 62%, 28% and 18% space overhead for ARCFLAG, TNR and AH on road graphs, respectively, and about 86% and 2% space overhead for ARCFLAG and TNR on co-authorship graphs, respectively. (f) Using synthetic data, we further find that proxies and DRAs are sensitive to the density and degree distribution of graphs and perform well on graphs that follow the power law distribution, and, moreover, for a given degree distribution, less nodes are captured when the average degree is higher.

2. Query Answering with Routing Proxies

In this section, we show how to answer shortest path and distance queries using routing proxies.

Based on the previous analyses, we present a framework for speeding-up shortest path and distance query answering, which consists of two modules: *preprocessing* and *query answering*. We next introduce the details of the framework.

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

(1) It first computes all DRAs and their maximal proxies with algorithm computeDRAs (as discussed in Section **??**).

(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, moreover, 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 $A_u^+ = A_u^1 \cup \ldots A_u^h$.

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.

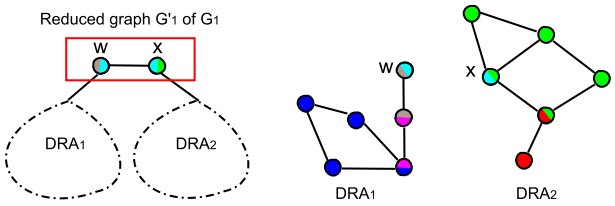

Figure 1. Example query answering

(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.

By the analyses in Section **??**, 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.,* bidirectional Dijkstra [9], ARCFLAG [11], CH [6], TNR [**?**], AH [20]) on the reduced graph G' for computing $path(u_s, u_t)$.

Similarly, the shortest distance $dist(s, t) = dist(s, u_s) + dist(u_s, u_t) + dist(u_t, t)$ can be computed.

We next illustrate how shortest path and distance queries are processed with an example below.

Example 1: Consider graph G_1 and its BCCs in Fig. **??**(1), in which the DRAs and their maximal proxies are computed by algorithm computeDRAs, *i.e.,* $dra_1 = \{u, v, BC_1, BC_2, BC_3\}$ with its proxy w and $dra_2 = \{y, BC_5, BC_6\}$ with its proxy x, as shown in Fig. 1. Note that here both A_w^+ and A_x^+ have single components. Moreover, the reduced graph G_1' of G_1 is the subgraph with nodes w and x only.

(1) Consider nodes s in dra_1 and t in dra_2, we first compute $dist(w, x)$ or $path(w, x)$ on G_1', and then let $dist(s, t) = dist(s, w) + dist(w, x) + dist(x, t)$ or $path(s, t) = path(s, w)/path(w, x)/path(x, t)$. Note that here $dist(s, w)$, $dist(x, t)$, $path(s, w)$ and $path(x, t)$ have all been computed in the precessing stage.

(2) If nodes s and t are both in dra_1 or dra_2, we directly compute their shortest path or distance on dra_1 or dra_2, as they have single components. □

Remarks. (1) As shown above, we need $O(d)$ extra space to store the routing information to compute shortest paths and distances, where d is the total number of nodes in all DRAs.

(2) It is easy to see that the main computation cost is reduced from graph G to its reduced graph G'. As shown in the experimental study (Section **??**), on average about 1/3 nodes of sparse graphs are captured by non-trivial proxies and their DRAs, *i.e.,* d is about $|V|/3$. That is, the reduced graph G' is about 2/3 size of the original graph G, and hence our data reduction technique could reduce graph sizes and speed up shortest path and distance computations.

3. Conclusion

We have studied how to speed-up (exact) shortest path and distance queries on large weighted undirected graphs. To do this, we propose a light-weight data reduction technique,

a notion of proxies such that each proxy represents a small subgraph, referred to as DRAs. We have shown that proxies and DRAs can be computed efficiently in linear time, and incur only a very small amount of extra space. We have also verified, both analytically and experimentally, that proxies significantly reduce graph sizes and improve efficiency of existing methods, such as bidirectional Dijkstra, ARCFLAG, TNR and AH for shortest path and distance queries.

A couple of topics are targeted for future work. We are to extend our techniques for dynamic graphs, as real-life networks are often dynamic [10], [19]. We are also to explore the possibility of revising routing proxies for directed graphs and other classes of graph queries, *e.g.,* reachability.

Acknowledgments

This work is supported in part by 973 Program (No. 2014CB 340300), NSFC (No. 61322207&61421003), Special Funds of Beijing Municipal Science & Technology Commission, and MSRA Collaborative Research Program. We also thank the anonymous reviewers for their valuable comments and suggestions that help improve the quality of this manuscript.

References

[1] J. Arz, D. Luxen, and P. Sanders. Transit node routing reconsidered. In *SEA*, 2013.

[2] J. Cheng, Y. Ke, S. Chu, and C. Cheng. Efficient processing of distance queries in large graphs: a vertex cover approach. In *SIGMOD*, 2012.

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

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *FOCS*, 1984.

[6] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, 2008.

[7] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, 2009.

[8] S. Liu, Y. Yue, and R. Krishnan. Adaptive collective routing using gaussian process dynamic congestion models. In *KDD*, 2013.

[9] M. Luby and P. Ragde. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4(4):551–567, 1989.

[10] S. Ma, J. Li, C. Hu, X. Lin, and J. Huai. Big graph search: challenges and techniques. *FCS*, 10(3):387–397, 2016.

[11] 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.

[12] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, 2009.

[13] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *ESA*, 2005.

[14] J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. *PVLDB*, 2(1):1210–1221, 2009.

[15] A. D. Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *WSDM*, 2010.

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

[17] F. Wei. Tedi: efficient shortest path query answering on graphs. In *SIGMOD*, 2010.

[18] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *PVLDB*, 5(5):406–417, 2012.

[19] W. Yu, C. C. Aggarwal, S. Ma, and H. Wang. On anomalous hotspot discovery in graph streams. In *ICDM*, 2013.

[20] 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.