An Efficient Reachability Queries Approach for Large Graph based on Cluster Structure

Yale Chai, Yao Ge
Chunyao Song and Peng Nie

College of Computer and Control Engineering, Nankai University
38 Tongyan Road, Tianjin 300350, P.R.China
Email: {chaiyl, geyao}@dbis.nankai.edu.cn, {chunyao.song, niepeng}@nankai.edu.cn

***Abstract*—Reachability query is a fundamental operation on graphs that finds the connection between vertices. Although plenty of techniques have been proposed for reachability queries, most of them are designed for directed graphs. Existing techniques for undirected graphs cannot handle large volumes of data. In this paper, we propose an undirected graph reachability (UGR) query algorithm by integrating graph clustering algorithm with traditional search methods. We first find core vertices by partitioning them into clusters, and then cluster non-core vertices according to their adjacent core vertices. After clustering, we take each cluster as a new vertex and compute transitive closure. Experimental results demonstrate the effectiveness and scalability of the proposed methods for the reachability query problem.**

Keywords–Reachability queries; Graph cluster; Search method.

I. INTRODUCTION

Many real-world networks can be modeled as a graph *G* = (*V*, *E*), where vertices in *V* represent entities and edges in *E* represent relationships between entities. Given two vertices u and v, a reachability query asks whether there exists a path between u and v in *G*. Nowadays, there are lots of techniques for reachability queries on a directed graph. Specifically, for any two vertices u and v in directed graph *G*, if $u \rightarrow v$ then there exists an order o and $o(u) < o(v)$ [1]. This order will become a vital part to construct the reachability index. As a result, these methods cannot be applied to undirected graphs, due to the non-ordering of undirected vertices. However, many networks can be modeled as undirected graphs. Take the social network for an example, we can treat each user as a vertex and consider the communicating between users as an edge. In this paper, we present a novel study on reachability queries for undirected graphs.

As explained in [2], to answer reachability queries in O(1) time, an extreme practice is to pre-compute and store the full transitive closure of edges, which requires a quadratic space complexity, making it infeasible to handle very large graphs. Our target is to improve the ability to scale to large graphs, as well as reduce the processing time. At present, a trend in dealing with big data is parallel processing: dividing the original data into small pieces, then, separately processing each piece and merging at the end. By clustering graph, we can divide the original graph into "pieces". Among all the graph clustering techniques, the structural graph clustering method can not only cluster graph rather quickly, but also ensure that vertices are reachable to each other within the same cluster. As far as we know, pSCAN [3] is a state-of-art graph clustering approach. Therefore, we propose a method which combing pSCAN with traditional search methods.

The rest of the paper is organized as following: in Section II, we introduce the undirected graph reachability (UGR) query algorithm. In Section III, we conduct complexity analysis, and present the evaluation results. We show the conclusion of our work in Section IV.

II. OUR APPROACH

Based on structural graph clustering, we present UGR approach for reachability querying. Our goal is to scale down the graph before using traditional, full search method. The pseudocode of UGR is shown in Algorithm 1.

Algorithm 1 UGR

```
Input: A graph $G$ = ($V$, $E$)), and parameters add $0 < \epsilon < 1$
    and $\mu \geq 2$
Output: A new graph $\widetilde{G}$ = ($\widetilde{V}$, $\widetilde{E}$)
 1: Initialize a disjoint-set data structure with vertices in $V$;
 2: for each vertex $u \in V$ do
 3:   core($u$) $\leftarrow$ false;
 4:   sd($u$) $\leftarrow$ 0 and ed($u$) $\leftarrow$ d[$u$];
 5: end for
 6: for each vertex $u \in V$ in no-increasing order do
 7:   Check if $u$ is a core vertex;
 8:   if sd($u$)$\geq \mu$ then
 9:     core($u$) $\leftarrow$ true;
10:     for each core vertex $v \in N[u]$ do
11:       union($u$,$v$); /* Make u,v in the same cluster */
12:     end for
13:   end if
14: end for
15: $\mathcal{C}_c \leftarrow$ set of subsets of core vertices;
16: $\mathcal{C} \leftarrow$ ClusterNoncore(); /* Cluster non-core vertices */
17: for each cluster $C \Subset \mathcal{C}$ do
18:   Add a vertex to $\widetilde{V}$;
19:   Add an edge to $\widetilde{E}$ for each neighbor of $C$;
20: end for
21: Depth-First-Search($\widetilde{G}$);
22: return $\widetilde{G}$;
```

Our algorithm mainly contains three steps. Firstly, we divide the graph G into clusters. Secondly, we generate a new graph $\widetilde{G}$ in which each vertex was a cluster in G. Finally, we compute the transitive closures of all vertices in $\widetilde{G}$. As shown in Algorithm 1, we use a disjoint-set data structure which maintains disjoint dynamic subsets. Initially, each vertex forms a singleton subset (Line 1); and the subsets union when vertices in the same cluster. For each vertex $u \in V$, we set u as non-core vertex, at the meantime, we incrementally maintain an effective-degree $ed(u)$ and a similar-degree $sd(u)$ for u (Line 2-5).

We structural cluster graph through a Two-Step Paradigm [3]: cluster core vertices (Line 6-15) and then cluster non-core vertices (line 16). For each vertex v adjacent to u, we compute the structural similarity $\sigma(u, v)$ between u and v as equation 1. If $\sigma(u, v) \geq \epsilon$, then we decide vertex u and v are structural similarity, and increase $sd(u)$ by one. Otherwise, we decrease $ed(u)$ by one. Once $sd(u) \geq \mu$, we determine u as a core vertex. On the contrary, once $ed(u) < \mu$, we determine u as a non-core vertex. In this way, the algorithm can terminate early without visiting all the neighbor of u. Furthermore, if v is a core vertex then we assign u and v to be in the same cluster (line 11).

$$\sigma(u, v) = \frac{|N[u] \bigcap N[v]|}{\sqrt{d[u] \cdot d[v]}} \quad (1)$$

where $N[u]$ is the structural neighborhood of a vertex u, and $d[u]$ is the degree of u.

Algorithm 2 ClusterNoncore

```
1: visited($u$) ← false for every vertex;
2: $\mathcal{C} \leftarrow \emptyset$
3: for each cluster $\widetilde{C} \in \mathcal{C}_{\widetilde{c}}$ do
4:   for each vertex $u \in \widetilde{C}$ do
5:     visited($u$) ← true;
6:     for each vertex $v \in N[u]$ do
7:       if sd($u$)<$\mu$ and $v \notin \widetilde{C}$ then
8:         $\widetilde{C} \leftarrow \widetilde{C} \cup \{v\}$ /* Add non-core to its neighbor*/
9:         visited($v$) ← true;
10:      end if
11:    end for
12:  end for
13: end for
14: /* Handle vertices that belong to no cluster */
15: for each vertex $u \in G$ do
16:   if !visited($u$) then
17:     Depth-First-Search($u$);
18:   end if
19: end for
```

Then, we cluster the rest of the vertices as shown in Algorithm 2. Initially, for each vertex u, we set $visited(u)$ to be false. Given the core cluster $\widetilde{C}$ in $\mathcal{C}_c$, for every vertex $u \in \widetilde{C}$, we include all neighbor of u into the same cluster as $\widetilde{C}$, and mark u,v visited (line 3-13). Obviously, $visited(u)$ equals true means vertex u has been assigned to cluster and no need to be visited in depth-first search (DFS). Afterwards, we iterate through every vertex in G that has not been included to any cluster, and execute DFS to merge it with its neighbor (line 15-19). So far, graph G has been divided into self-connectivity clusters.

Theorem 1: (Internal Connectivity) *Let $\mathcal{C}$ be any of clusters of graph G, for any two vertices $v1$, $v2 \in \mathcal{C}$, $v1$, $v2$ can reach to each other.*

Proof: As proved in [3], for any two vertices $v1$, $v2 \in \mathcal{C}$, there is a vertex $u \in \mathcal{C}$ such that both $v1$ and $v2$ are structure-reachable from u. In other words, $v1$ and $v2$ can reach each other within $\mathcal{C}$. ■

In order to compute the transitive closures between clusters, we consider each cluster as a vertex for convenience. The pseudocode to generate a new graph $\widetilde{G}$ is (Algorithm 1 line 17-20): given an empty graph $\widetilde{G}$, we add every cluster $\mathcal{C}$ into $\widetilde{G}$ as a vertex. Given two clusters $\mathcal{C}1$, $\mathcal{C}2$ (new vertex $V1, V2 \in \widetilde{G}$), if there exist an edge between vertex $u \in \mathcal{C}1$ and $v \in \mathcal{C}2$, then add an new edge between $V1$ and $V2$.

Finally, we use DFS on new graph $\widetilde{G}$ to compute the transitive closures. After that, for any vertex $V \in \widetilde{G}$, we can be aware of the set of vertices that V can reach. At query time, given two vertex $v1, v2 \in G$, we first trace back to their clusters and check the connectivity. The whole query time complexity is $O(1)$.

III. Result Discussion

Theorem 2: (Complexity Analysis) *Given graph G, let $Es \subseteq E$ be the set of adjacent vertex-pairs whose structural similarities have been computed, let N_c be the number of clusters in G. And n, m respectively are the number of vertices and edges in G. The time complexity of our UGR approach is $O(a(n) \cdot m + \sum_{(u,v)\in Es} min(d[u], d[v]) + N_c^2)$*, the space complexity is the $O(m + n)$.

Proof: The first part of the time complexity is related to disjoint-set data structure operations, where $a(n)$ is the extremely slowly growing inverse of the single-valued Ackermann function and is less than 5 for practical values of n. The second part of the time complexity is related to structural similarity computations. As proved in [4], $O(\sum_{(u,v)\in E} min(d[u], d[v])) \leq m^{1.5}$. So, the worst case time complexity of this part is $O(m^{1.5})$. Moreover, we execute DFS on new graph $\widetilde{G}$ who has N_c vertices and at most N_c edges, the second part of the time complexity is $O(N_c^2)$. Normally, the time of this part is so little that can be ignored. Thus, the time complexity of our approach is approximately equal to $O(a(n) \cdot m + O(m^{1.5})$. Besides, the space complexity of UGR is the same as pSCAN [3]. ■

In order to demonstrate our analysis, we implemented our UGR method, and compared it with traditional search methods: *DFS* [5] and *Warshall* [6]. We ran all experiments on a computer with an Intel 3.4 GHz CPU, 16GB RAM, and Windows10 OS. We evaluated the algorithms on five real datasets from the Stanford Network Analysis Platform[1], Table 1 lists the number of vertices and edges in the graphs. Table 2 reports the construction time of process (in ms). We only evaluated the performance of Warshall on first three datasets because it obviously slower than others. What's more, when the scale of dataset is small, DFS had better performance. However, it ran into stack overflow on the last two large graphs. Experiments showed that UGR is more scalable and stable than traditional search methods, especially on sparse graph.

IV. Conclusion and Future Work

This paper presents a study on reachability queries on large undirect graphs, moreover, the thought is easy to extend to the

[1]http://snap.stanford.edu/

TABLE I. DATASETS

DataSet	$\|V\|$	$\|E\|$
CA-GrQc	5242	14496
Enron	13220	111467
Cit-HepTh	27770	352,807
Email-EuAll	265,214	420,045
DBLP	317,080	1,049,866

TABLE II. COMPARISON OF CONSTRUCTION TIME

DataSet	*Warshall*	*DFS*	*UGR*
Enron(10000 mails)	4663	2	11
Enron(30000 mails)	73277	18	38
Enron(50000 mails)	651981	17	74
Enron(all)	–	23	107
CA-GrQc	–	28	7
Cit-HepTh	–	132	380
Email-EuAll	–	–	388
dblp	–	–	1064

directed graph as long as we change the clustering method. Based on the experiments, we show that our algorithms is more scalable and stable than traditional search methods, especially on sparse graph. For future work, this work can be extended in several interesting directions. First, we will study the evaluation of graph shortest-path search queries. Second, we will improve the clustering method and make our approach applied to the weighted graph. Third, we will exploit the distributed database to achieve higher scalability in terms of graph sizes.

ACKNOWLEDGMENT

This work was supported in part by Natural Science Foundation of Tianjin, under Grant No. 17JCYBJC23800; National 863 Program of China, under Grant No. 2015AA015401; and Research Foundation of The Ministry of Education and China Mobile, under Grant No. MCM20150507.

REFERENCES

[1] A. D. Zhu, W. Lin, S. Wang, and X. Xiao, “Reachability Queries on Large Dynamic Graphs: A Total Order Approach,” in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data June 22-27, 2014, Snowbird, Utah, USA*, pp. 1323–1334, ISBN: 978-1-4503-2376-5.

[2] Y. Hilmi, V. Chaoji, and M. J. Zaki, “GRAIL: scalable reachability index for large graphs,” *Proceedings of the VLDB Endowment*, vol. 3(1), pp. 276–284, 2010, ISSN: 2150-8097.

[3] L. Chang, W. Li, X. Lin, L. Qin, and W. Zhang, “pSCAN: Fast and Exact Structural Graph Clustering,” in *Proceedings of the 32th International Conference on Data Engineering (ICDE) May 16–20, 2016, Helsinki, Finland*, pp. 387–401, ISBN: 978-1-5090-2020-1.

[4] N. Chiba and T. Nishizeki, “Arboricity and Subgraph Listing Algorithms,” *Society for Industrial and Applied Mathematics*, vol. 14(1), pp. 210–223, 1994, ISSN: 0097-5397.

[5] S. Even, *Graph Algorithms (2nd ed.)*. Cambridge University Press, 2011, ISBN: 978-0-521-73653-4, pp. 46–48.

[6] T. H. Cormen, C. E. Leiserson, and R. Rivest, *Introduction to Algorithms (1st ed.)*. MIT Press and McGraw-Hill, 1990, ISBN: 0-262-03141-8, See in particular Section 26.2, ”The FloydWarshall algorithm”, pp. 558–565.