Top-K Aggregation Queries Over Large Networks

Xifeng Yan [#], Bin He [†], Feida Zhu [‡], Jiawei Han [§]

[#] *University of California at Santa Barbara*
xyan@cs.ucsb.edu

[†] *IBM Almaden Research Center*
binhe@us.ibm.com

[‡] *Singapore Management University*
fdzhu@smu.edu.sg

[§] *University of Illinois at Urbana-Champaign*
hanj@cs.uiuc.edu

Abstract— **Searching and mining large graphs today is critical to a variety of application domains, ranging from personalized recommendation in social networks, to searches for functional associations in biological pathways. In these domains, there is a need to perform aggregation operations on large-scale networks. Unfortunately the existing implementation of aggregation operations on relational databases does not guarantee superior performance in network space, especially when it involves edge traversals and joins of gigantic tables.**

In this paper, we investigate the neighborhood aggregation queries: Find nodes that have top-k highest aggregate values over their h-hop neighbors. While these basic queries are common in a wide range of search and recommendation tasks, surprisingly they have not been studied systematically. We developed a Local Neighborhood Aggregation framework, called LONA, to answer them efficiently. LONA exploits two properties unique in network space: First, the aggregate value for the neighboring nodes should be similar in most cases; Second, given the distribution of attribute values, it is possible to estimate the upper-bound value of aggregates. These two properties inspire the development of novel pruning techniques, forward pruning using differential index and backward pruning using partial distribution. Empirical results show that LONA could outperform the baseline algorithm up to 10 times in real-life large networks.

I. Introduction

As witnessed in physical, biological, and social networks, networks are ubiquitous and proliferating, e.g., the Internet, the Web, Facebook, LinkedIn, and thousands of online communities connected by Blogging and Instant Messaging. Managing and mining these large-scale networks is critical to a variety of application domains, ranging from personalized recommendation in social networks, to search for functional associations in biological pathways. Network analysis has attracted great attention in several research communities. Linkage analysis has evolved into powerful and easy-to-use search tools like Google[1]. Furthermore, advanced analysis of social networks might tackle very complicated mining tasks such as evaluating the network value of customers [4], [9] and link prediction [11].

Most of social and biological networks often have a node attribute set, denoted as $\Lambda = \{a_1, a_2, \ldots, a_t\}$. Each node has a value for these attributes, which describe various features and aspects of the entities that the nodes represent. For example, a node representing a Facebook user may have attributes showing if he/she is interested in online RPG games. In LinkedIn, each node represents a professional and the linkage shows a reference between two professionals. In communication networks, the intrusion packets could formulate a large, dynamic intrusion network, where each node corresponds to an IP address and there is an edge between two IP addresses if an intrusion attack takes place between them.

The existing analytical tools usually develop application-specific criteria to gauge the importance of nodes or to discover knowledge hidden in complex networks. However there is a growing need of processing very standard queries efficiently in large-scale networks. For example, for each node, find the aggregate value of an attribute for its neighbors within h-hops. This kind of queries could identify the popularity of a game console in one's social circle, or the number of times a gene is co-expressed with a group of known genes in co-expression networks.

II. Problem Formulations

The aggregation queries are useful for emerging applications in many online social communities such as book recommendation on Amazon, target marketing on Facebook, and gene function finding in biological networks. These applications can be unified by a general aggregation query scenario over a network. A top-k aggregation on graphs needs to solve three problems:

P1. evaluate the individual strength of a node, $f(u)$, for a given query. $f(u)$ could be as simple as 1/0, e.g., if a user recommends a movie or not, or it can be a classification function, e.g., how likely a user is a database expert.

P2. evaluate the collective strength of a node $F(u)$. $F(u)$ is an aggregate function over $f(v_1)$, $f(v_2)$, ..., $f(v_m)$, where $v_1, v_2, \ldots, v_m$ are u's neighbors within h-hops. $F(u)$ can be a simple aggregation function such as SUM, $f(v_1) + f(v_2) + \ldots + f(v_m)$[1], or AVG, $\frac{f(v_1)+\ldots+f(v_m)}{m}$. It can be as complicated as a non-linear function, e.g., learned by a collective classification method [13].

[1] If we introduce edge weights, $F(u)$ could be $w(u, v_1)f(v_1) + w(u, v_2)f(v_2) + \ldots + w(u, v_m)f(v_m)$, where $w(u, v)$ measures the connection strength between u and v, e.g., the inverse of the shortest distance between u and v.

P3. find top-k nodes with the highest scores.

We define the above problems as follows:

Definition 1 (Relevance Function): Given a network G, a relevance function $f : V \rightarrow [0, 1]$ assigns a score [0,1] to each node in G. 0 means the node is not relevant to the query, while 1 means full relevance.

Definition 2 (Neighborhood Aggregate): Given a network G and a relevance function $f : V \rightarrow R$, a sum aggregation of h-hop neighbors is defined as $F(u) = \sum_{v \in S_h(u)} f(v)$. An average aggregation of h-hop neighbors is defined as $F(u) = \frac{\sum_{v \in S_h(u)} f(v)}{|S_h(u)|}$, where $S_h(u)$ is the set of u's neighbors within h-hops.

Definition 3 (Top-k Neighborhood Aggregates): Given a network G, a relevance function f, and a neighborhood aggregation function F, find k nodes in $V(G)$ whose neighbors generate the highest aggregate score.

For a large network, it could be expensive to perform aggregations over the entire network for various kinds of queries. Assume on average each node has m 1-hop neighbors, in order to evaluate the collective strength of all the nodes, the number of edges to be accessed could be around $m^h|V|$ for h-hop queries. This computational cost is not affordable in applications involving large-scale networks and heavy query workloads.

The performance of using a relational query engine to process aggregation queries over networks is often costly. For 2-hop queries, it has to self-join two gigantic edge tables, if one indeed chooses table to store large graphs. In this paper, we introduce a solution to the above question by studying the two basic aggregation functions SUM and AVG. However, the similar ideas could be extended to other more complicated functions.

III. Forward Processing

A naive approach to answer top-k neighborhood aggregation queries is to check each node in the network, find its h-hop neighbors, aggregate their values together and then choose the k nodes with the highest aggregate values. We develop forward pruning techniques to improve the naive approach. Algorithm 1 shows the high-level procedure, named LONA-Forward. Line 11 finds new nodes that can be pruned and thus avoid forward processing in later iterations. The challenge is how to design an efficient and effective pruning algorithm, pruneNodes.

We observed that although we are not able to derive a tight bound for individual nodes, it is possible to derive a differential bound between a pair of nodes if they are connected. Intuitively, if one node has a low aggregate value, very likely its neighbors have low value, thus can be pruned.

Given a node u in a graph, let $S(u)$ denote the set of distinct nodes in u's h-hop neighborhood. For every node u, and its neighbor node v, the *differential index* tells the number of nodes in $S(v)$, but not in $S(u)$, denoted $delta(v-u) = |S(v) \setminus S(u)|$.

Algorithm 1 Pruning based Forward Processing

```
LONA-Forward($G$)
Output: Top $k$ $(node, aggr\_value)$ pairs
1: Add $G$'nodes into a queue $Q$
2: $topklist$ ={}, $topklbound$ = 0, $prunedlist$={}
3: while $Q$ is not empty do
4:     $u$ = get the top node in $Q$
5:     if $u$ not in $prunedlist$ then
6:         $F(u)$ = $u$'s aggregate value in $h$-hops
7:         if $F(u) > topklbound$ then
8:             update $topklist$ with ($u$, $F(u)$)
9:             update $topklbound$
10:        end if
11:        $pnodes$ = pruneNodes($u$, $F(u)$, $G$, $topklbound$)
12:        add nodes from $pnodes$ into $prunedlist$
13:    end if
14: end while
15: return $topklist$
```

To use differential index for node pruning, suppose we have conducted forward processing of a node u and denote its h-hop aggregate value as $F(u)$. We can then compute the upper bound of the aggregate value of any u's neighbor node v. For SUM aggregates, we can derive

$$\overline{F}_{sum}(v) = min(F(u) + delta(v-u), N(v) - 1 + f(v)). \quad (1)$$

We use $\overline{F}(v)$ to denote the upper bound value of $F(v)$. Given a node v, its possible upper bound is $N(v) - 1 + f(v)$ (i.e., node v itself has score $f(v)$ and all the other nodes in its h-hops have score 1). Given $F(u)$ and $delta(v-u)$, it can be concluded that $F(v)$ will be at most $F(u) + delta(v-u)$ (i.e., $S(v)$ contains all of u's nodes that have values and have score 1s for all the nodes that are not in $S(u)$). Therefore, we take the smaller one as the upper bound of $F(v)$. Actually, the upper bound of $F(v)$ is the minimum value of the bounds derived from v's friends. That is,

$$\overline{F}_{sum}(v) = \min_{u \in S(v)}\{F(u)+delta(v-u), N(v)-1+f(v)\}.$$

The upper bound of AVG aggregates is simply dividing the $\overline{F}_{sum}(v)$ value by the number of nodes in v's h-hops, i.e.,

$$\overline{F}_{avg}(v) = \frac{\overline{F}_{sum}(v)}{N(v)}. \quad (2)$$

For the forward pruning, we build the differential index for all the edges in a graph. After we did a forward processing on node u, for any u's neighbor node v that satisfies $\overline{F}(v) <$ *topklbound*, we can prune v and put it into the pruned node list.

The differential index adopted by forward processing needs to be pre-computed and stored. While it is more advanced than the naive approach, we are looking for new pruning techniques that do not need any pre-computed index. In the next section, we will introduce these techniques.

IV. Backward Processing

An alternative to forward processing is to apply a backward distribution method. For each node u, rather than aggregating the scores of its neighbors, the distribution process sends its

Algorithm 2 Backward Processing

```
BackwardNaive ($G$)
Output: Top $k$ $(node, aggr\_value)$ pairs
 1: for each non-zero node $u$ in $G$ do
 2:     for each node $v$ in $u$'s $h$-hops do
 3:         $F_{sum}(v) = F_{sum}(v) + f(u)$
 4:     end for
 5: end for
 6: if SUM function then
 7:     $topklist$ = pick $k$ nodes with the highest $F_{sum}(u)$ values.
 8: else if AVG function then
 9:     $topklist$ = pick $k$ nodes with the highest $\frac{F_{sum}(u)}{N(u)}$ values.
10: end if
11: return $topklist$
```

score to all of its neighbor nodes. When all the scores are backward distributed, we can calculate the aggregate value of all the nodes and then select the top k nodes.

The backward process has to wait until all of nodes are distributed; its cost is equal to the naive forward approach. However, there is one exception when the relevance function is 0-1 binary: It can skip nodes with 0 score, since by default these zero nodes have no contribution to the aggregate values.

In particular, we distribute nodes according to their scores in a descending order. In this order, when we do backward processing of a node u, we are able to compute the upper bound of the aggregate value for u's h-hop neighbors:

Given a node v, suppose it has been scanned by l nodes u_1, ..., u_l using backward processing, and u_l is the latest one. That is, $f(u_l)$ is the lowest one among $f(u_1)$, ..., $f(u_l)$. For the SUM function, we can compute $F_{sum}(v)$'s upper bound as

$$\overline{F}_{sum}(v) = \sum_{i=1}^{l} f(u_i) + f(u_l) * (N(v) - l) + f(v). \qquad (3)$$

This is because v has $N(v)$ neighbors, among which l neighbors' score is known and $N(v) - l$ neighbors' score is unknown. Since we distribute the score in a descending order, we could bound $N(v) - l$ neighbors using the lowest one that has been distributed, i.e., $f(u_l)$.

The backward processing does partial distribution on a subset of nodes whose score is higher than a given threshold γ. Then it orders all the nodes according to their aggregate upper bound values, and performs a naive forward processing, where the unpromising nodes are discarded.

V. Experimental Results

In this section, we report our experiments that validate the efficiency and effectiveness of LONA algorithms and illustrate its properties on a series of real-life graph datasets. In particular, we experimented with the LONA-Forward algorithm and the LONA-Backward algorithm. These two algorithms are compared with the baseline algorithm, "Base", which performs naive forward processing, without pruning. We assume memory-resident large networks, as having them on disk would not be practical in terms of graph traversal. We are currently developing an infrastructure to partition large networks into subnetworks and distribute them into multiple machines. Our experiments demonstrated that LONA outperforms Base by up to 10 times in three real graph datasets tested in this study: (1) Condensed Matter Collaboration Network (cond-mat 2005)[14], available at http: www-personal.umich.edu/~mejn/netdata/cond-mat-2005.zip. It has around 40k nodes and 180k edges. (2) Citation Network (cite75_99) [5]. It has 3M nodes and 16M edges. (3) Intrusion Network (IPsec) IP traffic data from a security company. It has 2.5M nodes and 4.3M edges. All our experiments were performed on a 2.5GHZ Xeon quad core, 8GB memory Server.

Relevance Functions. Given a graph, we need to generate functions that assign relevance scores to each node (i.e., the relevance function f in Definition 1). We designed a mixture function to mimic the setting of relevance functions in real-life applications. Our relevance function consists of two components: random assignment function, f_r whose value has an exponential distribution, and a random walk procedure f_w. f_r assigns a score whose range is between 0 and 1. It has a blacking ratio parameter r, which controls the percentage of nodes to be assigned "1".

We tested 2-hop queries since they are much harder than 1-hop queries and more popular than 3+ hop queries.

Results. Figures 1, 2 and 3 show the runtime performance of the three algorithms for varying top-k SUM queries with blacking ratio $r = 1\%$. The experiments are conducted on the Collaboration network, Citation network, and the Intrusion network. As shown in each figure, both algorithms LONA-Forward and LONA-Backward outperform Base by a significantly large margin, up to 10 times.

We then run the same set of experiments using the AVG aggregation function. Figures 4, 5 and 6 show the runtime performance for varying top-k AVG queries with blacking ratio $r = 1\%$. As shown in each figure, LONA-Backward outperforms Base by a significantly large margin, up to 10 times. LONA-Forward is also better than Base; however its performance deteriorates in the citation network when k increases. In general, AVG queries are more difficult to process for LONA-Forward when the blacking ratio r is low. The upper bound derived from the differential index (Equation 1) in the forward approach is not tight any more since it assumes all the neighbors of v, but not u, have relevance score 1.

VI. Conclusions

Top k query processing has been extensively studied in RDBMS. Some works tackle the problem outside the core of query processing engines using indexes, views, or application-level query rewriting [3], [6], [16], [18]. Recent research mainly focused on supporting top k queries inside relational query engine [2], [7], [8], [10], because it has been proved to be more efficient. Supporting top k query in SQL is first proposed by [2]. Its syntax or similar variations have been adopted in most of the current RDBMSs. However, the existing top k optimization techniques cannot be directly applied to graph data.

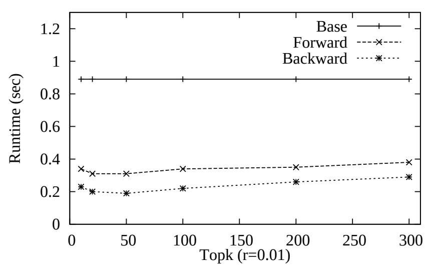

Fig. 1. Collaboration (SUM)

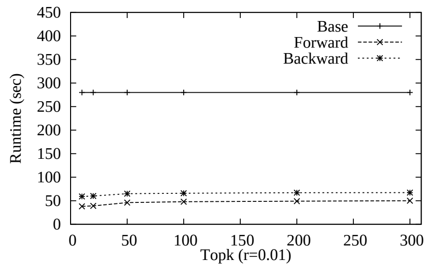

Fig. 2. Citation (SUM)

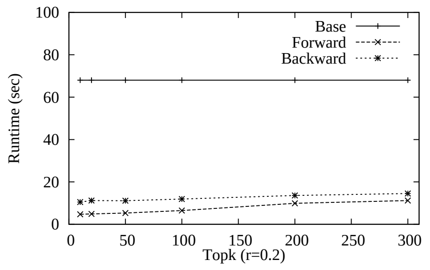

Fig. 3. Intrusion (SUM)

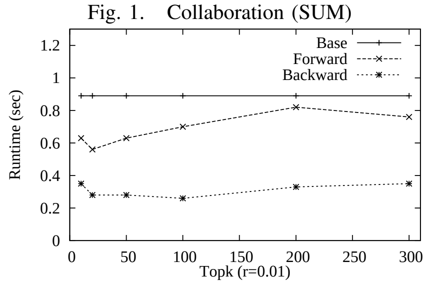

Fig. 4. Collaboration (AVG)

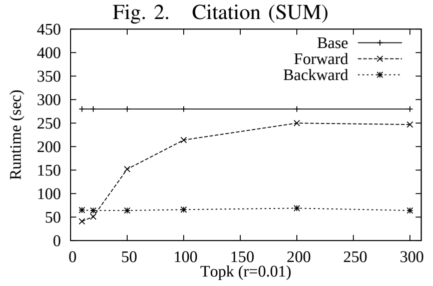

Fig. 5. Citation (AVG)

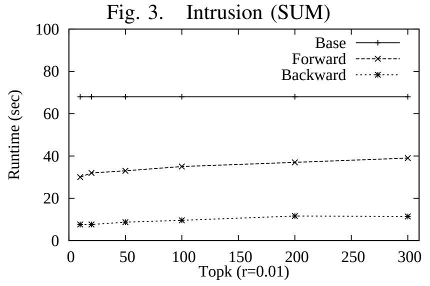

Fig. 6. Intrusion (AVG)

Graph summarization and aggregation methods have been developed recently. Tian et al. [15] introduced SNAP operations to consistently merge nodes and edges with regard to a predefined hierarchy. When such hierarchy is not available, graph summarization can still be achieved by various kinds of clustering methods. For example, Wu et. al. adopted geodesic clustering [17] to achieve multi-level geodesic aggregation. Navlakha et al. [12] applied the MDL principle to yield highly intuitive coarse-level aggregate graphs for a large input graph. The aggregation problem discussed in these studies is about summarizing networks, different from the neighborhood aggregation problem studied in our work.

In this paper, we investigated the techniques of answering local aggregation queries efficiently over large networks. These query forms have become increasingly important for search and mining tasks over social networks and biological networks. Surprisingly they have not been studied systematically. We developed a **Lo**cal **N**eighborhood **A**ggregation framework, called LONA, to answer these queries. LONA exploits two properties of aggregation functions over a network: First, the aggregates of neighboring nodes should be similar in many cases; Second, it is possible to estimate the average value of aggregates, which is likely less than top-k values. Empirical results show that LONA could outperform baseline algorithms up to 10 times in real-life datasets.

Acknowledgements: The work was supported by NSF Grant IIS-0842769, IIS-0847925, and IIS-0905215.

References

[1] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. 7th Int. Conf. on World Wide Web (WWW'98)*, pages 107–117, 1998.

[2] M. Carey and D. Kossmann. On saying "enough already!" in SQL. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 219–230, 1997.

[3] S. Chaudhuri. Evaluating top-k selection queries. In *VLDB*, pages 397–410, 1999.

[4] P. Domingos and M. Richardson. Mining the network value of customers. In *Proc. 2001 ACM Int. Conf. on Knowledge Discovery in Databases (KDD'01)*, pages 57–66, 2001.

[5] B. Hall, A. Jaffe, and M. Trajtenberg. The NBER patent citation data file: Lessons, insights and methodological tools. NBER Working Papers 8498, National Bureau of Economic Research, Inc, 2001.

[6] V. Hristidis, N. Koudas, Y. Papakonstantinou, and L. Ca. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, pages 259–270, 2001.

[7] I. Ilyas, W. Aref, and A. Elmagarmid. Supporting top-k join queries in relational databases. In *VLDB*, pages 754–765, 2003.

[8] I. Ilyas, R. Shah, W. Aref, J. Vitter, and A. Elmagarmid. Rank-aware query optimization. In *SIGMOD*, pages 203–214, 2004.

[9] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proc. 2003 ACM Int. Conf. on Knowledge Discovery in Databases (KDD'03)*, pages 137 – 146, 2003.

[10] C. Li, K. C.-C. Chang, I. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In *SIGMOD*, pages 131–142. ACM Press, 2005.

[11] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proc. of the 12th Int. Conf. on Information and Knowledge Management (CIKM'03)*, pages 556 – 559, 2003.

[12] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *Proc. 2008 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'08)*, pages 419–432, 2008.

[13] J. Neville and D. Jensen. Iterative classification in relational data. In *Proc. of the Workshop on Statistical Relational Learning, 17th National Conf. on Artificial Intelligence*, page 4249, 2000.

[14] M. Newman. The structure of scientific collaboration networks. *Proc. Natl. Acad. Sci.*, 98:404–409, 2001.

[15] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *Proc. 2008 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'08)*, pages 567–580, 2008.

[16] P. Tsaparas, N. Koudas, and T. Palpanas. Ranked join indices. In *ICDE*, pages 277–288, 2003.

[17] A. Wu, M. Garland, and J. Han. Mining scale-free networks using geodesic clustering. In *Proc. 2004 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'04)*, pages 719–724, 2004.

[18] K. Yi, H. Yu, J. Yang, G. Xia, and Y. Chen. Efficient maintenance of materialized top-k views. In *ICDE*, pages 189–200, 2003.