ISSN (e): 2250 – 3005 || Volume, 05 || Issue, 10 ||October – 2015 ||
International Journal of Computational Engineering Research (IJCER)

Sub-Graph Finding Information over Nebula Networks

K.Eswara Rao[$1] , A.NagaBhushana Rao[$2]
[$1,$2] Asst. Professor, Dept. Of CSE, AITAM, Tekkali, SKLM, AP, INDIA.

ABSTRACT

Social and information networks have been extensively studied over years. This paper studies a new query on sub graph search on heterogeneous networks. Given an uncertain network of N objects, where each object is associated with a network to an underlying critical problem of discovering, top-k sub graphs of entities with rare and surprising associations returns k objects such that the expected matching sub graph queries efficiently involves, Compute all matching sub graphs which satisfy "Nebula computing requests" and this query is useful in ranking such results based on the rarity and the interestingness of the associations among nebula requests in the sub graphs. "In evaluating Top k-selection queries, "we compute information nebula using a global structural context similarity, and our similarity measure is independent of connection sub graphs". We need to compute the previous work on the matching problem can be harnessed for expected best for a naive ranking after matching for large graphs. Top k-selection sets and search for the optimal selection set with the large graphs; sub graphs may have enormous number of matches. In this paper, we identify several important properties of top-k selection queries, We propose novel top–K mechanisms to exploit these indexes for answering interesting sub graph queries efficiently.

Key words: *Nebula Networks, Top-k sub graph, Indexing*

I. INTRODUCTION

With the ever-increasing popularity of entity-centric applications, it becomes very important to study the interactions between entities, which are captured using edges in the entity relationship (or information) nebula networks. Entity-relationship networks with multiple types of entities are usually referred to as heterogeneous information networks. For example, bibliographic networks capture associations like 'Identification of fingerprints for the serine protease Family'. Similarly, social networks, biological protein-enzyme classification Using SVM ((support vector machine), Wikipedia entity network, etc. also capture a variety of rich associations. In these applications, it is critical to detect novel connections or associations among objects based on some subgraph queries. Two example problems are shown in Figures1 and are described as follows.

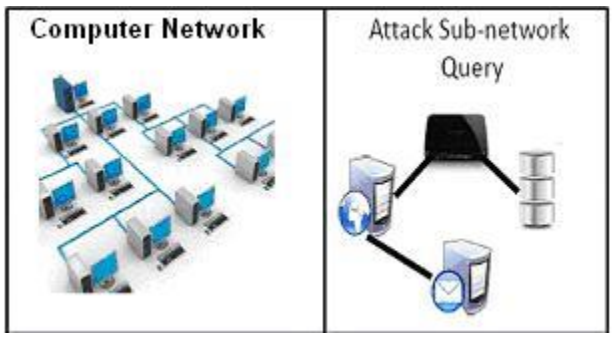

Fig.1 Attack Localization Problem

P1: query Selection over nebula: Organization Nebula networks consist of two subgraphs and object nodes where two graphs are connected if they have worked together on a successful mission in the past, and a subgraph is linked to an nebula if the person has a known expertise in using that subgraph. For example, US army network which consists of 2-3M nebulas and much more linked objects. A manager in such an organization may have a mission which can be defined by a query graph of objects and networks. For example, the right half of Figure 1 shows an organization network while the left half of the figure shows a sample mission

query graph consisting of two subgraphs. The network has edges with weights such that a high weight implies higher compatibility between the nodes connected by the edge. The manager is interested in selecting a subgraph to accomplish the mission with the network to network compatibilities as specified in the mission query. Using the historical compatibility based organization network, how can we find the best query (selection) over the nebula?

P2: Attack Localization: Consider a computer network as shown in the left part of Figure 2. It can consist of a large number of components like database servers, hubs, switches, desktops, routers, VOIP phones, etc. Consider a simple attack on multiple web servers in such a network where the attack script runs on a compromised web server. The script reads a lot of data from the database server (through the network hub) and sends out spam emails through the email server. Such an attack leads to an increase in data transfer rate along the connections in multiple "attack sub-networks" of the form as shown in the right part of the figure. Many such attacks follow the same pattern of increase in data transfer rates. How can a network administrator localize such attacks quickly and find the worst affected sub-networks given the observed data transfer rates across all the links in the network?

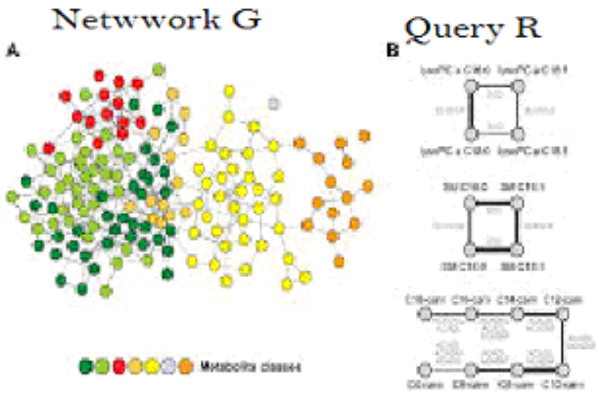

Fig.2 Example of a Network ***G*** and a Query ***Q***

Both of these problems share a common underlying problem: Given a heterogeneous network G, a heterogeneous subgraph query Q, and an edge interestingness measure I which defines the edge weight, find the top-K matching subgraphs S with the highest interestingness. The two problems can be expressed in terms of the underlying problem as follows. P1: G = organization network, Q = mission query, I = historical compatibility, S = team. P2: G = computer network, Q = an "attack sub-network" query, I = data transfer rate, S = critical sub-networks. Besides the two tasks, this proposed problem finds numerous other applications. For example, the interesting subgraph matches can be useful in network bottleneck discovery based on link bandwidth on computer networks, suspicious relationship discovery in social networks, de-noising the data by identifying noisy associations in data integration systems, etc.

Comparison with Previous Work

The proposed problem falls into the category of the subgraph matching problems. Subgraph matching has been studied in the graph query processing literature with respect to approximate matches [4], [25], [26], [30] and exact matches [18], [27], [31]. Subgraph match queries have also been proposed for RDF graphs [15], probabilistic graphs [24] and temporal graphs [1]. The proposed problem can be solved by first finding all matches for the query using the existing graph matching methods and then ranking the matches. The cost of exhaustive enumeration for all the matches can be prohibitive for large graphs. Hence, this paper proposes a more efficient solution to the top-K subgraph matching problem which exploits novel graph indexes. Many different forms of top-K queries on graphs have been studied in the literature [5], [21], [23], [28], [30]. Gou et al. [5] solve the problem only for twig queries while we solve the problem for general subgraphs. Yan et al. [21] deal with the problem of finding top-K highest aggregate values over their h-hop neighbors, in which no subgraph queries are involved. Zhu et al. [28] aim at finding top-K largest frequent patterns from a graph, which does not involve a subgraph query either. Different from existing top-K work, the proposed work deals with a novel definition of top-K general subgraph match queries, which have a large number of practical applications as discussed above.

Brief Overview of Top-K Interesting Subgraph Discovery
Given a heterogeneous network containing entities of various types, and a subgraph query, the aim is to find the top-K matching subgraphs from the network. We study the following two aspects of this problem in a tightly integrated.
way: (1) computing all possible matching subgraphs from the network, and (2) computing interestingness score for each match by aggregating the weights of each of its edges. To solve these problems, we present an efficient solution which exploits two low-cost index structures (a graph topology index and a maximum metapath weight (MMW) index) to perform top-K ranking while matching (RWM). Multiple applications of the top-K heuristic, a smart ordering of edges for query processing, quick pruning of the edge lists using the topology index and the computation of tight upper bound scores using the MMW index contribute to the efficiency of the proposed solution in answering the top-K *interesting subgraph* queries.

Summary
We make the following contributions in this paper. We propose the problem of top-K interesting subgraph discovery in information networks given a heterogeneous edge-weighted network and a heterogeneous unweighted query. To solve this problem, we propose two low-cost indexes (a graph topology index and a maximum metapath weight (MMW) index) which summarize the network topology and provide an upper bound on maximum metapath weights separately. Using these indexes, we provide a ranking while Matching (RWM) algorithm with multiple applications of the top-K heuristic to answer *interesting subgraph* queries on large graphs efficiently. Using extensive experiments on several synthetic datasets, we compare the efficiency of the proposed RWM methodology with the simple ranking after matching (RAM) baseline. We also show effectiveness of RWM on two real datasets with detailed analysis. Our paper is organized as follows. In Section II, we define the *top*-K *interesting subgraph discovery* problem. The proposed approach consists of two phases: an offline index construction phase and an online query processing phase which are detailed in Sections III and IV respectively. In Section V, we discuss various general scenarios in which the proposed approach can be applied. We present results with detailed insights on several synthetic and real datasets in Section VI. We discuss related work and summarize the paper in Sections VII and VIII respectively.

II. PROBLEM DEFINITION

In this section, we formalize the problem definition and present an overview of the proposed system. We start with an introduction to some preliminary concepts.
Definition 1 (A Heterogeneous Network). A heterogeneous network is an undirected graph
$G = < V_G, E_G, type_G, weight\ G >$ where V_G is a finite set of vertices (representing entities) and E_G is a finite set of edges each being an unordered pair of distinct vertices. $type_G$ is a function defined on the vertex set as $type_G : V_G \rightarrow T_G$ where T_G is the set of entity types and $|T_G| = T$. weight G is a function defined on the edge set as $weight_G : E_G \rightarrow R \in [0, 1]$. $Weight_G(e)$ represents the interestingness measure value associated with the edge e.

For example, Figure 3 shows a network G with three types of nodes. $T_G = \{A,B,C\}$. $|V_G|=13$, and $|E_G|=18$.

Definition 2 (Subgraph Query on a Network). *A subgraph query Q on a network G is a graph consisting of node set V_Q and edge set E_Q. Each node could be of any type from T_G.*
For example, Figure 2 shows a query Q with four nodes. $|V_Q|=4$, and $|E_Q|=3$. The network G and the query Q shown in Figure 2 will be used as a running example throughout this paper.

Definition 3 (Subgraph Isomorphism). A graph $g = <V_g, E_g, type_g >$ is subgraph isomorphic to another graph $g' = <V_{g'}, E_{g'}, type_{g'} >$ if there exists a subgraph isomorphism from g to g′. A subgraph isomorphism is an injective function $M : V_g \rightarrow V_{g'}$ such that (1) $\forall v \in Vg$, $M(v) \in V_{g'}$ and $type_g(v)=type_{g'}(M(v))$, (2) $\forall e =(u, v) \in E_g$, $e' =(M(u), M(v)) \in E_{g'}$.
Definition 4 (Match). The query graph Q can be subgraph isomorphic to multiple subgraphs of G. Each such subgraph of G is called a match or a matching subgraph of G. The query Q can be answered by returning all exact matching subgraphs from G. For example, the subgraph of G induced by vertices (8, 9, 5, 6) is a match for the query Q on network G shown in Figure 2. For sake of brevity, we will use the vertex set (tuple notation) to refer to the subgraph induced by the vertex set.

Definition 5 (Interestingness Score). The interestingness score for a match M for a query Q in a graph G is defined as the sum of its edge weights. For example, the interestingness score for the occurrence {8,9, 5, 6} is 2.1. Though we use sum as an aggregation function here, any other monotonic aggregation function could also be used.

Definition 6 (Top–*K* Interesting Subgraph Discovery Problem).
Given*: A heterogeneous information network* G*, a heterogeneous unweighted query* Q*, and an edge interestingness measure.*
Find*: Top-*K *matching subgraphs with highest interestingness scores.*

For example, (3, 4, 5, 6) and (4, 3, 2, 7) are the Top two matching sub-graphs both with the score 2.2 for the query *Q* on network *G* in Figure 2.
For example, the metapath corresponding to the path (5, 4,7) is (A,A,B). There are TD distinct metapaths of length D where T is the number of types. Each column of a topology index corresponds to a metapath.

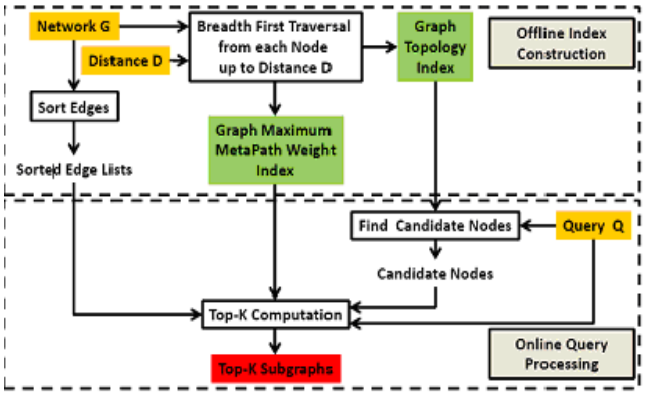

Fig-3 Top-*K* interesting sub graph Discovery System Diagram

Figure 2 shows the graph topology index for the first 4 nodes of the graph shown in Figure 2. For example, for node 2, there are two 2-hop neighbors of type A (4 and 8) reachable via the metapath (B,A). Hence the entry topology(2, (B,A))=2. A blank entry in the index indicates that there is no node of type t at a distance d from node n along the corresponding metapath. As we shall see in Section IV-A, the topology index plays a crucial role in reducing the search space by pruning away candidate graph nodes that cannot be instantiated for a given query node.

III. *TOP-K* INTERESTING SUBGRAPH QUERY PROCESSING

Given a query Q with node set VQ and edge set EQ, top-K matching subgraphs are discovered by traversing the sorted edge lists in the top to bottom order with the following speedup heuristics. First for each node in V_Q, a set of nodes from the graph that could be potential candidates for the query node, is identified using the topology index (Algorithm 1). The edges in the sorted edge lists that contain nodes other than the potential candidate nodes are marked as invalid. This prunes away many edges and speeds up the edge list traversal. The query Q is then processed using these edge lists in a way similar to the *top-K* join query processing (Section IV-B) adapted significantly to handle network queries. The approach discussed in Section IV-B is further made faster by the tighter upper bound scores computed using the MMW index (Algorithm 2). We will discuss these in detail in this section.

Algorithm 1 Candidate Node Filtering Algorithm

Input: (1) Query Node *q*, (2) Graph Node *p*, (3) *topology* [p], (4) *queryTopology*[q], (5) Index Parameter *D*
Output: Is *p* a potential candidate node for query node *q*?
1: **for** $d = 1 \ldots D$ **do**
2: **for** $mp = 1 \ldots T^d$ **do**
3: if *queryTopology*[*q*][*d*][*mp*] > *topology* [*p*][*d*][*mp*] **then**
4: Return False
5: Return True

Candidate Node Filtering Algorithm

The proposed candidate node filtering approach is summarized in Algorithm 1. For each distance value d, all possible metapaths of length d are checked. By comparing the topology for all metapaths with the corresponding query Topology values (Step 3), it can be inferred whether the candidate p is valid to be an instantiation of

query node q for some match. The time complexity is O(DTD+1). Candidate pruning leads to shortening of the edge lists associated with any of the query edges. For example, nodes 2, 8 and 10 get pruned for the query node Q2. Thus, the edge list corresponding to the query edge (Q2,Q3) will have the following AA edges marked as invalid: (2,3), (8,9) and (10,9).

B. Top-K Match Computation

In this sub-section, we describe the top-K algorithm to perform interestingness scoring and matching simultaneously. The algorithm is based on the following key idea. A top-K heap is maintained which stores the best K answers seen so far. The sorted edge lists are traversed from top to bottom. Each time an edge with maximum edge weight from any of the lists is picked and all possible size-1 matches in which that edge can occur are computed. Candidate size-1 matches are grown one edge at a time till they grow to the size of the query. Partially grown candidate matches can be discarded if the upper bound score of these matches falls below the minimum element in the top-K heap. The algorithm terminates when no subgraph using the remaining edges can result into a candidate match with upper bound score greater than the minimum element in the top-K heap. We discuss the details below.

Definition 7 (Valid Edge). A valid edge e with respect to a query edge qE is a graph edge such that both of its endpoints are contained in the potential candidate set for the correspond *ing query nodes in* qE. *Recall that the potential candidate set for each query node is computed using Algorithm 1.*

The sorted edge lists are quite similar to the lists in Fagin's TA [4]. To traverse the edge lists in the top to bottom order, a pointer is maintained with every edge list. The pointers are initialized to point to the topmost graph edge in the sorted edge list, which is valid for at least one query edge. As the pointers move down the lists, they move to the next valid edge rather than moving to the next edge in the list (as in Fagin's TA).

Definition 8 (Size-c candidate match). A size-c candidate match is a partially grown match such that c of its edges have been instantiated using the matching graph edges.

VI. EXPERIMENTS

We perform experiments on multiple synthetic datasets each of which simulates power law graphs. We evaluate the results on the real datasets using case studies. We perform a comprehensive analysis of the objects in the top subgraphs returned by the proposed algorithm to justify their interestingness. Data and code is available at http://dais.cs.uiuc.edu/manish/RWM/.

A. Synthetic Datasets

We construct 4 synthetic graphs using the R-MAT graph generator in GT-Graph software [2]: G1, G2, G3 and G4 with 103, 104, 105, and 106 nodes respectively. Each graph has a number of edges equal to 10 times the number of nodes. Thus, we consider graphs with exponential increase in graph size. Each node is assigned a random type from 1 to 5. Also, each edge is assigned a weight chosen uniformly randomly between 0 and 1. All the experiments were performed on an Intel Xeon CPU X5650 4-Core 2.67GHz machine with 24GB memory running Linux 3.2.0. The code is written in Java. The distance parameter D for the indexes is set to 2 for both the proposed approach RWM (Ranking While Matching) and the baseline RAM (Ranking After Matching), unless specified explicitly. Also unless specified explicitly, we are interested in computing top 10 interesting subgraphs (K=10) and the execution times mentioned in the tables and the plots are obtained by repeating the experiments 10 times.

Baseline: Ranking After Matching (RAM)

The problem of finding the matches of a query Q in a heterogeneous network G has been studied earlier [20], [27]. In [27], the authors present an index structure called SPath. SPath stores for every node,a list of its typed neighbors at a distance d for $1 \leq d \leq D$. SPath index is then used to efficiently find matches for a query in a path-at-a-time way: the query is first decomposed into a set of shortest paths and then the matches are generated one path at a time. This method is used as a baseline.

Index Construction Time

Figure 4 shows the index construction times for the various indexes. Generating the sorted edge lists is very fast. Even for the largest graph with a million nodes, the sorted edge lists creation takes around 40 seconds. The Topology+MMW (D=2) and SPath (D=2) curves show the time required for construction of these indexes, for various graph sizes. The X axis denotes the number of nodes in the synthetic graphs and the Y axis shows the index construction time in seconds. Note the Y axis is plotted using a log scale.

The index construction time rises linearly as the graph size grows. Also, as expected the index construction time rises as D increases.

Index Size

Figure 4 shows the size of each index for different values of D. The X axis plots the number of nodes in the synthetic graphs and the Y axis plots the size of the index (in KBs) using a logarithmic scale. Different curves plot the sizes of various indexes, and the graph. Note that the size of the topology index and the MMW index for D=2 is actually smaller than the size of the graph. Even when the index parameter is increased to D=3, the topology and the MMW indexes remain much smaller than the SPath index for D=2. For D=3, the SPath index grows very fast as the size of the graph increases. As expected as the graph size increases, the size of each index increases. While the increase is manageable for the Edge lists, the MMW index and the topology index, the increase in SPath index size is humongous.

Query Execution Time

We experiment with three types of queries: path, clique and general subgraphs, of sizes from 2 to 5. We present a comparison of different techniques for the graph G2 using the indexes with *D*=2.

	$\|V_Q\|= 2$	$\|V_Q\|= 3$	$\|V_Q\|= 4$	$\|V_Q\|= 5$
RAM	245	2004	14628	169328
RWM0	15	32	43	122
RWM1	19	36	98	178
RWM2	20	40	442	6887
RWM3	218	1733	2337	3933
RWM4	18	34	42	118

Table I
Query Execution time (MESC) for Path queries
(Graph G2 and Indexes with D=2)

	$\|V_Q\|= 2$	$\|V_Q\|= 3$	$\|V_Q\|= 4$	$\|V_Q\|= 5$
RAM	144	8698	34639	174992
RWM0	11	376	14789	229236
RWM1	14	448	16789	200075
RWM2	13	567	19089	201709
RWM3	157	2279	17184	161545
RWM4	12	347	13567	198617

The tables I- II show the average execution times for an average of 10 queries per experimental setting each repeated 10 times. The six different techniques are as follows: RAM (the ranking after matching baseline), RWM0 (without using the candidate node filtering), RWM1 (without using the MMW index), RWM2 (same as RWM1 without the pruning any partially grown candidates), RWM3 (same as RWM1 without the global Top-***K*** quit check), RWM4 (same as RWM1 with the MMW index). Clearly, RAM takes muchlonger execution times for all types of queries. We observed that the larger the number of candidate matches, the more the execution time gap between the RAM method and the RWM methods. An interesting case is $|V_Q|=5$ for the clique queries. Actually there are very few (less than 10) cliques of size 5 of a particular type in the graph. Hence, we can see that almost all the approaches take almost the same time. In this case, the Top-***K*** computation overheads associated with the RWM approaches and lack of pruning result in relatively lower execution time for RAM.

Next, note that RWM4 usually performs faster than RWM1. The time savings are higher for the path queries compared to the subgraph or clique queries. This is expected because the upper bound scores computed in RWM4 are tighter only if most of the query structure can be covered by the non overlapping paths. Also, RWM0 performs slightly better than RWM4 for smaller query sizes, but candidate node filtering helps significantly as query size increases.

Table-III shows the time split between the candidate filtering step and the actual Top–***K*** execution. Note that the candidate filtering takes a very small fraction of the total query execution time.

	$\|V_Q\|=2$	$\|V_Q\|=3$	$\|V_Q\|=4$	$\|V_Q\|=5$
RAM	158	3186	39294	469962
RWM0	10	165	824	4660
RWM1	12	195	1022	5891
RWM2	12	212	3135	27363
RWM3	111	1486	3978	9972
RWM4	12	165	791	4518

Table –III

Query Execution Time (Msec) for Subgraph Queries (Graph G2 and Indexes With D=2)

Query size / Query Type	$\|V_Q\| = 2$		$\|V_Q\| = 3$		$\|V_Q\| = 4$		$\|V_Q\| = 5$	
	CFT	TET	CFT	TET	CFT	TET	CFT	TET
Path	8	10	10	24	10	32	12	106
Clique	5	6	8	##	9	13538	9	2E+05
Subgraph	6	6	9	##	10	781	12	4506

Table –IV

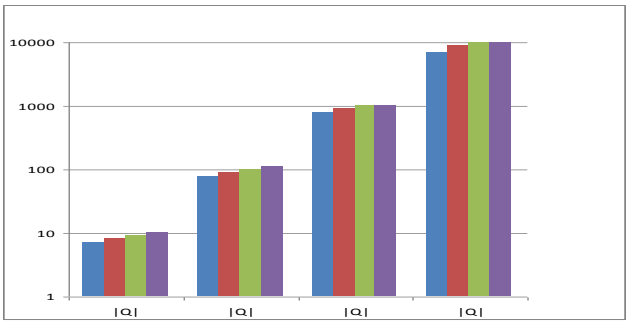

Fig. 5 Query Execution Time for Different Values of ***K***

Scalability Results

We run the 20 path and general subgraph queries (each 10 times) over all the 4 synthetic graphs using RWM4 and present the results in Table IV. The table shows that the execution time increases linearly with the graph size, and exponentially with the query size. Even though the execution time is exponential in query size, (1) that is the case with most subgraph matching algorithms, and (2) intuitive user queries are limited in size by limits of human interpretability for most applications. Effect of Varying the K Figure-5 shows the effect of varying K on 20 path and general subgraph queries on graph G2 using RWM4. As expected, the query execution time increases as K increases. However, the increase in execution time is reasonably small enough making the system usable even for larger values of K.

	$\|V_Q\| = 2$	$\|VQ\| = 3$	$\|VQ\| = 4$	$\|VQ\| = 5$
# SIZE-1 Candidates	9.55	7.87	4.38	1.63
# SIZE-2 Candidates		29.28	19.31	8.94
# SIZE-3 Candidates			24.42	24.5
# SIZE-4 Candidates				14.61

TABLE V

Number of candidates as percentage of total matches for different query sizes and candidates sizes

Table-V shows the percentage of candidates of different sizes with respect to the total number of matches. The results shown in this table are obtained by running the algorithm for the 20 path and subgraph queries on graph G2. We removed the clique queries because the number of cliques of size 5 matching such queries is less than 10 and hence no pruning occurs. Note that on an average, the number of candidates is around 14% of the total number of matches. Clearly, for subgraph queries there are candidates of higher sizes also, but the number of such candidates is much smaller (< 1%) compared to the number of matches, and so we do not show them here.

V. RELATED WORK

The network (graph) query problem can be formulated as a selection operator on graph databases and has been studied first in the theory literature as the subgraph isomorphism problem [3], [14], [20]. One way of answering network queries is to store the underlying graph structure in relational tables and then use join operations. However, joins are expensive, and so fast algorithms have been proposed for approximate

	DBLP	Wikipedia
Nmber of Nodes	138 K	670K
number of edges	1.6M	4.1M
number of types	3	10
Sorted Edge List Index Size	50 MB	261 MB
Topology Index Size	5.8 MB	148 MB
MMW Index Size	11.4 MB	249 MB
Spath Index Size	4.3 GB	13.7 GB
Sorted Edge List Construction Time	12 sec	23 sec
Topology + MMW Construction Time	461 min	1094 min
Average Query Time	100 sec	42 sec

graph matching as well as for exact graph matching. A problem related to the proposed problem is: given a subgraph query, find graphs from a graph database which contain the subgraph [16], [22], [29]. All top-K processing algorithms are based on the Fagin et al.'s classic TA algorithm [4]. Growing a candidate solution edge-by-edge in a network can be considered to be similar to performing a join in relational databases. The candidates are thus grown one edge at a time much like the processing of a top-K join query [11] and as detailed in Section IV-B. However, we make the top-K join processing faster by tighter upper bounds computed using the MMW index and list pruning using the topology index. The top-K joins on networks with the support of such graph indexes is our novel contribution. The proposed problem is also related to the team selection literature. However, most of such literature following the work of Lappas et al. [13] focuses on clique (or set) queries [10], unlike the general subgraph queries handled by the proposed approach. Top- K matching subgraphs can also be considered as statistical outliers. Compared to our previous work on outlier detection from network data [6], [7], [8], [9], we focus on query based outlier detection in this work. For more comparisons with previous work, please refer to Section I.

VI. CONCLUSION

In this paper, we studied the problem of finding top-K interesting subgraphs corresponding to a typed unweighted query aplied on a heterogeneous edge-weighted information network. The problem has many practical applications. The baseline ranking after matching solution is very inefficient for large graphs where the number of matches is humongous. We proposed a solution consisting of an offline index construction phase and an online query processing phase. The low cost indexes built in the offline phase capture the topology and the upper bound on the interestingness of the metapaths in the network. Further, we proposed efficient top-K heuristics that exploit these indexes for answering subgraph queries very efficiently in an online manner. Besides showing the efficiency and scalability of the proposed approach on synthetic datasets, we also showed interesting subgraphs discovered from real datasets like Wikipedia and DBLP. In the future, we plan to study this problem in a temporal setting.

REFERENCES

[1] P. Bogdanov, M. Mongiov`ı, and A. K. Singh. Mining Heavy Subgraphs in Time-Evolving Networks. In *ICDM*, pages 81–90, 2011.

[2] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *SDM*, pages 442–446, 2004.

[3] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (Sub)Graph Is omorphism Algorithm for Matching Large Graphs. *TPAMI*, 26(10):1367– 1372, 2004.

[4] Y. Tian, R. C. Mceachin, C. Santos, D. J. States, and J. M. Patel. SAGA: A Subgraph Matching Tool for Biological Graphs. *Bioinformatics*, 23(2):232–239, Jan 2007.

[4] R. Fagin, R. Kumar, and D. Sivakumar. Comparing Top-K Lists. In *SODA*, pages 28–36, 2003.

[5] G. Gou and R. Chirkova. Efficient Algorithms for Exact Ranked Twigpattern Matching over Graphs. In *SIGMOD*, pages 581–594, 2008.

[6] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. Outlier Detection for Temporal Data. In *SDM*, 2013.

[7] M. Gupta, J. Gao, and J. Han. Community Distribution Outlier Detection in Heterogeneous Information Networks. In *ECML PKDD*, pages 557– 573, 2013.

[8] M. Gupta, J. Gao, Y. Sun, and J. Han. Community Trend Outlier Detection using Soft Temporal Pattern Mining. In *ECML PKDD*, pages692–708, 2012.

[9] M. Gupta, J. Gao, Y. Sun, and J. Han. Integrating Community Matching and Outlier Detection for Mining Evolutionary Community Outliers. In *KDD*, pages 859–867, 2012.
[10] M. Gupta, J. Gao, X. Yan, H. Cam, and J. Han. On Detecting Association-Based Clique Outliers in Heterogeneous Information Networks. In *ASONAM*, 2013.
[11] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting Top-K Join Queries in Relational Databases. *VLDB Journal*, 13(3):207–221, Sep 2004.
[12] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *J. Sci. Comp*, 20(1):359–392, Dec 1998.
[13] T. Lappas, K. L iu, and E. Terzi. Finding a Team of Experts in Social Networks. In *KDD*, pages 467–476, 2009.
[14] B. D. McKay. Practical Graph Isomorphism. *Congressus Numerantiu m*, 30:45–87, 1981.
[15] Y. Qi, K. S. Candan, and M. L. Sapino. Sum-Max Monotonic Ranked Joins for Evaluating Top-K Twig Queries on Weighted Data Graphs. In *VLDB*, pages 507–518, 2007.
[16] S. Ranu and A. K. Singh. GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases. In *ICDE*, pages 844– 855, 2009.
[17] Y. Sun, Y. Yu, and J. Han. Ranking-based Clustering of Heterogeneous Information Networks with Star Network Schema. In *KDD*, pages 797–806, 2009.
[18] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient Subgraph Matching on Billion Node Graphs. *PVLDB*, 5(9):788–799, May 2012.
[20] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM*, 23(1):31–42, Jan 1976.
[21] X. Yan, B. He, F. Zhu, and J. Han. Top-K Aggregation Queries ov er Large Networks. In *ICDE*, pages 377–380, 2010.
[22] X. Yan, P. S. Yu, and J. Han. Substructure Similarity Search in Graph Databases. In *SIGMOD*, pages 766–777, 2005.
[23] J. Yang, W. Su, S. Li, and M. M. Dalkilic¸. WIGM: Discovery of Subgraph Patterns in a Large Weighted Graph. In *SDM*, pages 1083– 1094, 2012.
[24] Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient Subgraph Similarity Search on Large Probabilistic Graph Databases. *PVLDB*, 5(9):800–811, May 2012.
[25] X. Zeng, J. Cheng, J. X. Yu, and S. Feng. Top-K Graph Pattern Matching: A Twig Query Approach. In *WAIM*, pages 284–295, 2012.
[26] S. Zhang, J. Yang, and W. Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *PVLDB*, 3(1):1185–1194, 2010.
[27] P. Zhao and J. Han. On Graph Query Optimization in Large Networks. *PVLDB*, 3(1):340–351, 2010.
[28] F. Zhu, Q. Qu, D. Lo, X. Yan, J. Han, and P. S. Yu. Mining Top-K Large Structural Patterns in a Massive Network. *PVLDB*, 4(11):807–818, 2011.
[29] Y. Zhu, L. Qin, J. X. Yu, and H. Cheng. Finding Top-K Similar Graphs in Graph Databases. In *EDBT*, pages 456–467, 2012.
[30] L. Zou, L. Chen, and Y. Lu. Top-K Subgraph Matching Query in a Large Graph. In *PIKM*, pages 139–146, 2007.
[31] L. Zou, L. Chen, and M. T. ¨Ozsu. Distance-join: Pattern Match Query in a Large Graph Database. *PVLDB*, 2(1):886–897, Aug 2009.