Query-based Graph Cuboid Outlier Detection

Ayushi Dalmia*, Manish Gupta† and Vasudeva Varma‡
*†‡ International Institute of Information Technology, Hyderabad
†Microsoft
Email: *ayushi.dalmia@research.iiit.ac.in, †gmanish@microsoft.com, ‡vv@iiit.ac.in

***Abstract*—Various projections or views of a heterogeneous information network can be modeled using the graph OLAP (On-line Analytical Processing) framework for effective decision making. Detecting anomalous projections of the network can help the analysts identify regions of interest from the graph specific to the projection attribute. While most previous studies on outlier detection in graphs deal with outlier nodes, edges or subgraphs, we are the first to propose detection of graph cuboid outliers. Further we perform this detection in a query sensitive way. Given a general subgraph query on a heterogeneous network, we study the problem of finding outlier cuboids from the graph OLAP lattice. A Graph Cuboid Outlier (GCOutlier) is a cuboid with exceptionally high density of matches for the query. The GCOutlier detection task is clearly challenging because: (1) finding matches for the query (subgraph isomorphism) is NP-hard; (2) number of matches for the query can be very high; and (3) number of cuboids can be large. We provide an approximate solution to the problem by computing only a fraction of the total matches originating from a select set of candidate nodes and including a select set of edges, chosen smartly. We perform extensive experiments on synthetic datasets to showcase the execution time versus accuracy trade-off. Experiments on real datasets like Four Area and Delicious containing thousands of nodes reveal interesting GCOutliers.**

***Keywords*—Graph OLAP, Outlier Detection, Graph Projection Outliers, Graph Cuboid Outliers, Information Networks**

I. Introduction

In a large variety of applications, heterogeneous information networks (HINs) are used extensively to represent real world data. HINs are graphs with nodes of multiple types. To support data analytics on HINs, Chen et al. [1] proposed a graph OLAP framework. Graph OLAP allows the user to navigate through an information network across multiple levels and multiple dimensions. Each cuboid in such a graph OLAP lattice can be considered as a projection or a view of the network. Each cuboid represents a subgraph of the original graph induced by the nodes belonging to the particular value of the dimension. For example, consider a bibliographic network of authors, papers, conferences and keywords. One can construct a graph OLAP for such a network with respect to the research area dimension. Then the (DM+IR) projected network is a graph which consists of only those authors, papers, conferences and keywords which belong to either DM (Data Mining) or IR (Information Retrieval) areas or both.

GCOutliers: Given a general subgraph query on a heterogeneous network, one can discover all the matches for the query. Further, every match can be assigned to one or more cuboids depending on the value of the OLAP dimension for the nodes in the match. A Graph Cuboid Outlier (GCOutlier) is a cuboid with exceptionally high density of matches for the query. We study the problem of finding outlier cuboids from the graph OLAP lattice.

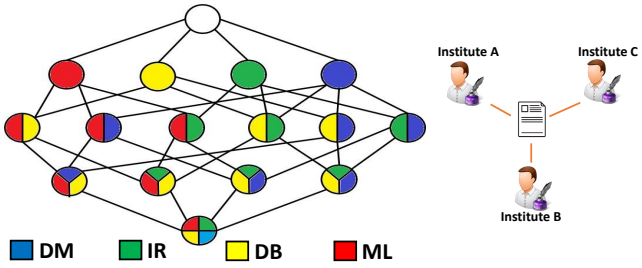

Fig. 1. Left: Bibliographic Network OLAP Lattice of Cuboids for Four Research Areas: Data Mining(DM), Machine Learning (ML), Information Retrieval (IR), Database (DB). Right: A Sample Query

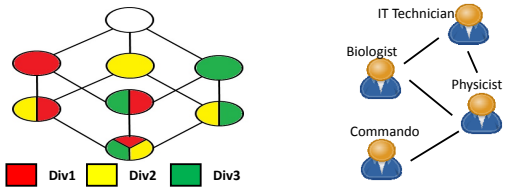

Fig. 2. Left: Organization Network OLAP Lattice of Cuboids for Three Divisions: Div1, Div2, Div3. Right: A Sample Mission Query

GCOutlier Case Studies: GCOutliers are interesting because they provide insights about the information network and support in decision making process, especially in identifying semantic regions of interest within the network. Interesting examples of GCOutliers can be commonly observed in various real-world scenarios. Here are two specific cases in detail.

Research Area Cuboid Outliers: Consider a bibliographic network lattice as shown in Figure 1 (left) on the research area dimension with four research areas: Data Mining(DM), Database (DB), Information Retrieval (IR) and Machine Learning (ML). Similarly, the lattice can be drawn for a combination of other dimensions like "year" along with "research areas". Using the query described in Figure 1 (right), GCOutlier detection can help satisfy the following user information need: "*In which research areas and in which set of years, there were exceptionally high collaborations between authors of institutes A, B and C?*" The user can expect to discover a particular (or group of) year and a research area for which some large funding was made by multi-institution projects leading to papers with authors from institutes A, B and C (e.g., "big data" in recent few years).

Organization Division Cuboid Outliers: Consider an organiza-

tion network lattice as shown in Figure 2 (left). For simplicity we show only three divisions but in general there could be a large number of divisions in an organization. Employees are connected to each other in the network if they have a history of working together. Using a mission query as shown in Figure 2 (right), one can answer this question: "*In which divisions should I try to find teams of people who can help me accomplish my mission?*" The user can expect to find divisions of the organization which have handled similar missions earlier.

Besides these examples, GCOutliers find numerous other applications, e.g., finding regions in a spatial/sensor network showing high concentration of a specific behavior, finding parts of an electronic circuit that need to be redesigned because they contain a high density of faulty design patterns, etc.

Relationship with Previous Work: Our work is most related to a recent hot sub-area of outlier detection: outlier detection in graphs. While general outlier detection in graphs has been studied for quite some time, query based outlier detection has gained attention very recently. A large number of algorithms have been proposed for unexpected, missing or anomalous nodes [2], [3], edges [4], [5], subgraphs [6], [7], [8], [9] and graphs [10]. To the best of our knowledge, we propose a first work on finding cuboid outliers. Note that a cuboid represents a semantic sub-network defined by a particular set of values of the OLAP dimension. Thus unlike general subgraphs, a cuboid is a semantically related collection of nodes.

Brief Overview of GCOutlier Detection: Given a heterogeneous network having high dimensional data and consisting of nodes of different types, represented as a Graph OLAP, the goal is to find interesting outlier cuboids. This is challenging because: (1) finding matches for the query (subgraph isomorphism) is NP-hard; (2) number of matches for the query can be very high; and (3) number of cuboids can be large. Given a query, we need to first find matching subgraphs from the graph but we may not need to find all subgraphs. The proposed approach finds matches originating from only a carefully-chosen small sample of nodes and including edges from a select set of edges. This is done by learning query-specific node and edge regression models and assigning a probability to each node and edge respectively, of being a part of some match. While this leads to a reduced set of matches from the graph, it reduces the execution time drastically while maintaining the accuracy of the outlier cuboid computation. The discovered sample of matches is further used to compute approximate outlier scores for each cuboid. Cuboids with exceptionally high scores are output as GCOutliers.

Summary: We make the following contributions in this paper.

- We propose the problem of graph cuboid outlier discovery given a heterogeneous network in the form of a graph OLAP lattice and a subgraph query.
- To find a sample of all matches and hence outlier cuboids from large graphs, we propose methods to grow matches originating from only those nodes with high probability of being a part of some match and including only high probability edges. The probabilities are learned using query-specific node and edge regression models.
- Using extensive experiments on synthetic datasets, we compare the effectiveness and efficiency of the proposed methods. Our experiments result in a good time versus accuracy trade-off on synthetic datasets, and interesting GCOutliers from real datasets.

Our paper is organized as follows. In Section II, we formalize the graph cuboid outlier detection problem. The proposed approach consists of two phases: an offline index construction phase and an online query processing phase which are detailed in Section III. We present results with detailed insights on several synthetic and real datasets in Section IV. We discuss related work in Section V. The paper is summarized in Section VI.

II. GCOutlier Detection Problem Definition

In this section we formally define the GCOutlier Detection problem. In order to aid the understanding of the problem statement, we present the following preliminary concepts.

Definition 1 (Heterogeneous Network): A heterogeneous network is an undirected graph $G = \langle V_G, E_G, type_G, attr_G \rangle$ 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 $type_G : V_G \rightarrow \mathcal{T}_G$ where $\mathcal{T}_G$ is the set of node types and $|\mathcal{T}_G| = T$. Each type has a fixed set of attributes. $attr_G$ is a function defined on the vertex set which associates each node with a vector of size D_{type} of attribute values. At least one of these attributes is used as an OLAP dimension when creating the graph OLAP.

For example, in the bibliographic networks (Figure 1), $\mathcal{T}_G = \{author, conference, paper, keywords\}$. Edges represent relationships like "author" wrote a "paper", etc. Nodes have attributes like research area, affiliation and year. In the organization network (Figure 2), nodes could have attributes like name, gender, organizational division of work, profession, etc. Figure 3 shows a heterogeneous network G with three types of nodes, $T_G = \{A, B, C\}$, $|V_G| = 13$ and $|E_G| = 18$.

Definition 2 (OLAP Dimension Attributes): It is a subset of attributes which are used for generating graph cuboids (or projections). At least one of node attributes ($attr_G$) is used as an OLAP dimension when creating the graph OLAP.

For example, in the bibliographic networks (Figure 1), research area could be used as an OLAP dimension. In the organization network, division could be used as an OLAP dimension. In Figure 3, the network is projected over two dimensions, X and Y. Both X and Y can take 2 values, $\{X_1, X_2\}$ and $\{Y_1, Y_2\}$ respectively. Each vertex takes some value for both the attributes X and Y.

Definition 3 (Subgraph Query on a Network): A sub-graph query Q on a heterogeneous network G is a graph consisting of node set V_Q and edge set E_Q. Each node could be of any type from $\mathcal{T}_G$ and could be associated with a predicate on its attribute values.

For example, the query in Figure 1 consists of authors and a paper where the authors have predicates on the "affiliation"

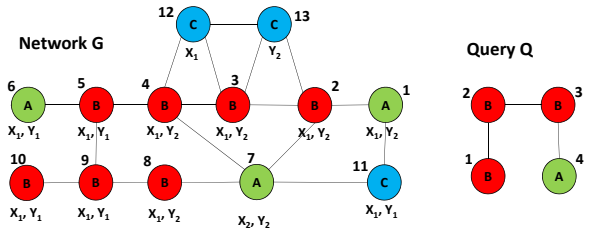

Fig. 3. Example of a Network G and a Sample Query Q

attribute. Also, Figure 3 shows a query Q with four nodes, $|V_Q| = 4$, and $|E_Q| = 3$.

Definition 4 (Graph OLAP): The heterogeneous network G can be projected along particular node attributes (called OLAP dimensions) to obtain a lattice of projections or views. Each projection is called a cuboid and corresponds to a set of values for the dimensions. Thus, a cuboid represents a sub-graph consisting of nodes with the corresponding set of projection attribute values for that cuboid, along with the edges induced by those nodes in G.

For example, in Figure 1, the projection attribute is "research area". Thus, the DM cuboid represents a DM-projected network consisting of authors, conferences, keywords, papers in DM and edges connecting them. In Figure 3, (X_1+Y_1) cuboid consists of nodes 5, 6, 9, 10, 11 and 12, and edges (5, 6), (5, 9) and (9, 10).

Definition 5 (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. Let S represent the set of values for the projection attribute for the nodes in a particular match. The match belongs to those R projected cuboids in the graph OLAP where $S \subset R$.

The query Q can be answered by returning all exact matching subgraphs from G. For example, in Figure 3, match $(8, 9, 5, 6)$ for Q from G belongs to $(X_1 + Y_1 + Y_2)$ and $(X_1 + X_2 + Y_1 + Y_2)$ cuboids. Here S=(X_1, Y_1, Y_2). Other matches for Q are $(3, 4, 5, 6)$, $(10, 9, 5, 6)$, $(4, 3, 2, 7)$, $(2, 3, 4, 7)$, $(9, 5, 4, 7)$, $(10, 9, 8, 7)$, $(5, 9, 8, 7)$ and $(4, 3, 2, 1)$.

Definition 6 (GCOutlier Score): Let a cuboid c contain d edges. Let n be the number of edges covered by matches of query Q from c. We define the GCOutlier Score of cuboid c as the ratio $\frac{n}{d}$ when d is non-zero. If d is 0, we define the outlier score to be 0.

There could be many other ways of defining the outlier score. But we chose this definition because (1) it captures the cuboid with unexpectedly high frequency of matches, and (2) normalizes with respect to the size of the cuboid. In Figure 3, the cuboid $(X_1 + Y_2)$ consists of the following edges: (1, 2), (2, 3), (3, 4), (4, 12), (3, 12), (2, 13), (3, 13), and (12, 13). Out of the matches for the query Q, only the match (4,3,2,1) belongs to the cuboid (X_1+Y_2) and covers three edges. Thus the GCOutlier score for cuboid $(X_1 + Y_2)$ is 3/8.

Definition 7 (Graph Cuboid Outlier Detection Problem):
Given: A heterogeneous information network G represented as a graph OLAP, a heterogeneous query Q.

Find: Top K cuboids from the graph OLAP with highest outlier scores which are not a super-set of higher or equally scoring cuboids.

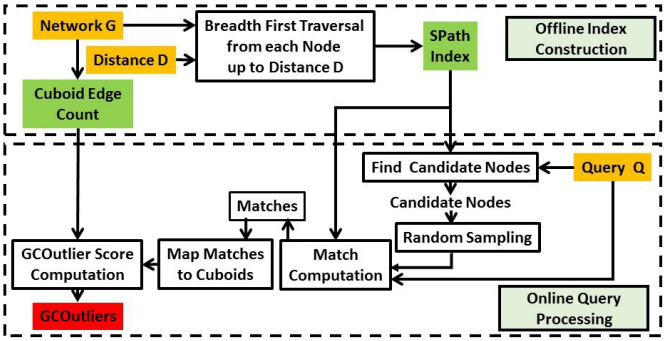

Fig. 4. Graph Cuboid Outlier Detection (GCOD-RS) System Diagram

Table I shows the GCOutlier scores for each cuboid for graph G in Figure 3. Cuboids not shown in the table have a score of 0.

Table I. GCOutlier Score of Cuboids for Graph OLAP in Figure 3

Cuboid	# Edges in Cuboid	# Matching Edges	GCOutlier Score
$X_1 + Y_1$	3	3	1.00
$X_1 + Y_2$	8	3	0.38
$X_1 + Y_1 + X_2$	3	3	1.00
$X_1 + X_2 + Y_2$	11	5	0.45
$X_1 + Y_1 + Y_2$	14	7	0.50
$X_1 + Y_1 + X_2 + Y_2$	18	11	0.61

In this example, if K is set to 1, the cuboid $(X_1 + Y_1)$ will be returned as a GCOutlier. Both the cuboids $(X_1 + Y_1)$ and $(X_1 + Y_1 + X_2)$ have a score of 1, but since $(X_1 + Y_1 + X_2)$ contains $(X_1 + Y_1)$, we ignore $(X_1 + Y_1 + X_2)$.

III. GCOutlier Detection Approach

In this section, we discuss two proposed approaches for the GCOutlier detection problem: (a) Graph Cuboid Outlier Detection using Random Sampling (GCOD-RS), and (b) Graph Cuboid Outlier Detection using Regression Models (GCOD-RM).

A. GCOutlier Detection System Overview

Figures 4 and 5 show a broad overview of the proposed systems using random sampling (GCOD-RS) and using regression models (GCOD-RM) respectively. For a given information network G we first pre-process it offline as shown in the top half of the figure. The lower half denotes the online processing phase in which the user query Q is processed using the indexes generated in the offline phase. Note that the online processing of the subgraph queries is approximate, i.e., approximate results are obtained by exploiting the indexes generated in the offline phase. We will explore the time-accuracy trade-off of the approximation in Section IV.

B. Offline Index Construction for the GCOD System

The offline part of the GCOD system is run once for a graph. It consists of building two indexes: an SPath index and a cuboid

edge count list. We detail the method of constructing these indexes in the following.

SPath Index: SPath [11] is a previously proposed high performance graph indexing mechanism to find matching subgraphs for a general subgraph query on large networks. SPath leverages decomposed shortest paths around vertex neighborhood as basic indexing units, which prove to be both effective in graph search space pruning and highly scalable in index construction and deployment. Via SPath, a graph query is processed and optimized beyond the traditional vertex-at-a-time fashion to a more efficient path-at-a-time way: the query is first decomposed to a set of shortest paths, among which a subset of candidates with good selectivity is picked by a query plan optimizer; candidate paths are further joined together to help recover the query graph to finalize the graph query processing.

For every vertex $v \in V_G$, its Neighborhood Signature $NS_G(v)$ is constructed. For a given distance D, a breadth-first traversal from v upto a distance D is required in order to obtain the shortest path information in the D-hop neighborhood subgraph of v. The SPath index is later used in the online phase to compute the matching subgraphs. The signature $NS_G(v)$ stores the set of nodes of a particular type at a particular distance $d \leq D$ from node v. For example, for node 4, there are two 1-hop neighbors of type B viz. $(3, 5)$, three 2-hop neighbors of type B viz. $(2, 8, 9)$.

Cuboid Edge Count: Besides the SPath Index, we also maintain a list which captures the number of edges covered by each cuboid in the entire graph. We assign an edge to all the cuboids which correspond to the projection attribute values of both the edge endpoints. This list is later used in computing the outlier score of the cuboids.

The second column of Table I indicates the number of edges covered by each cuboid for the graph shown in Figure 3. Edges $(5, 6)$, $(5, 9)$ and $(9, 10)$ contribute to the edge list of cuboid $\{X_1, Y_1\}$. Thus the number of edges covered by cuboid $\{X_1, Y_1\}$ is 3.

Time and Space Complexity: The SPath index construction and building the cuboid edge count needs to be done just once and is an offline task.

The worst case time complexity for SPath index construction is $O(|V_G| \times |E_G|)$ while the worst case space complexity is $O(|V_G|^2)$. In case of the edges covered by the cuboid list, it needs to scan the edges of the graph and compare the projection attribute values covered by each edge with that of each cuboid. Thus, if the projection attribute can take n unique values, the number of cuboids is 2^n and the time complexity for computing the list is $O(|E_G| \times 2^n)$. Since we store only the edge count per cuboid, space complexity is clearly $O(2^n)$.

Although the worst case complexities seem very large, in practical cases, the graphs and the indexes are quite sparse and hence index construction times and space are relatively much smaller.

C. Online Query Processing for the GCOD System

Given a graph G and a query Q, outlier cuboids are discovered by first extracting the exact matches using the SPath index. The number of matches can be very high for some queries. After obtaining the exact matches we map each edge of the match to all possible cuboids. An edge can belong to multiple cuboids. Thus, we obtain the number of matching edges belonging to each cuboid. Next, we compute the outlier score of each cuboid using the cuboid edge count list. As defined in Section II, the outlier score for a cuboid is the ratio of the number of matching edges covered by the cuboid to the total number of edges covered by the cuboid. Based on the outlier density the cuboids are ranked and the top K cuboids are returned as GCOutliers. We call this as the baseline online processing approach which provides exact GCOutliers. Table I shows outlier scores for all cuboids for Graph OLAP in Figure 3, except the cuboids with a 0 score.

The baseline online processing approach though accurate incurs high execution time especially for large queries. To compute the matches for query Q, the baseline algorithm first computes a list of candidate graph nodes for every query node. A graph node g is marked as a candidate node for a query node Q if the graph node g's type-wise d-hop neighborhood is a superset of the type-wise d-hop neighborhood expected by the query node Q. Further, computations are run from each of these candidate nodes in the smallest list among various query nodes, and matches are "grown" around them path-at-a-time.

We propose two different online processing approaches for approximate computation of GCOutliers : GCOD-RS and GCOD-RM. Both of these approaches reduce execution time by computing only a small fraction of the total matches while ensuring high accuracy for GCOutlier computation. While GCOD-RS depends on random sampling to achieve a good time-accuracy trade-off in computation of Graph Cuboid Outliers, GCOD-RM learns two regression models for every query to further improve the trade-off. We discuss both these approaches in detail in the following.

1) Graph Cuboid Outlier Detection using Random Sampling (GCOD-RS):

In GCOD-RS, we adapt the baseline algorithm by random sampling on the pool of candidate nodes. Thus, we use a small percentage of candidate nodes to obtain the matches rather than all the candidate nodes. This implies that the GCOD-RS approach returns a sample of matches rather than all the matches. The hope is that the random sample affects all the cuboids equivalently and so the relative cuboid outlier rankings do not change. We experiment by varying the percentage of candidate nodes selected and show the time-accuracy trade-off in Section IV.

The overall GCOutlier Detection Algorithm using Random Sampling (GCOD-RS) is illustrated in Algorithm 1.

2) Graph Cuboid Outlier Detection using Regression Models (GCOD-RM):

Similar to GCOD-RS, GCOD-RM first obtains a set of candidate nodes using the neighborhood topology checks of the graph node and the query node. Random sampling of the candidate nodes in GCOD-RS results in a very small number of matches to be discovered if the rate of sampling is small. This

Algorithm 1 Graph Cuboid Outlier Detection Algorithm using Random Sampling (GCOD-RS)

Input: (1) Graph G, (2) Query Q, (3) Index Parameter D, (4) Number of Outliers K (5) Sampling rate τ
Output: K GCOutliers.
1: Compute SPath Index using D, and List of Cuboid Edge Count for G once.
2: Compute node signatures NS for each node in Q.
3: Use NS-Containment neighborhood topology checks (as described in [11]) to compute candidate nodes for every query node.
4: $q_v \leftarrow$ Query node with smallest sized candidate list.
5: $CandidateList \leftarrow$ Randomly sample with rate τ from the candidate list for q_v.
6: Compute matches using candidates in $CandidateList$ only for q_v using SPath [11].
7: Map the matches to cuboids.
8: Compute outlier score for each cuboid and rank.
9: Return top K cuboids.

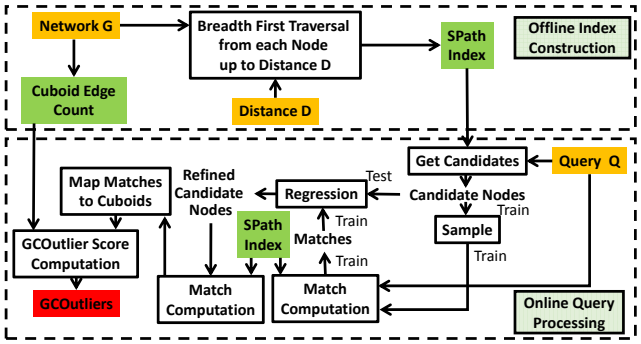

Fig. 5. Graph Cuboid Outlier Detection (GCOD-RM) System Diagram

in turn can lead to poor accuracy for the GCOutlier Detection. Therefore, in GCOD-RM, we perform a more principled sampling using a node regression model. The node regression model is used to predict the probability of a node belonging to some match for the input query versus not. Further, we also learn an edge regression model to predict the probability of an edge being a part of some match or not. Edges with very low probability are ignored when "growing" the matches from selected candidate nodes.

The regression models are trained as follows. Given a query, we first sample a few (say $\tau\%$) candidate nodes and then perform computations to grow matches from those nodes. Each of these candidate nodes are associated with a label that denotes the number of matches of which this node is a part of. Thus, nodes which are not a part of any match get a label of zero. Similarly, edges incident on the candidate node are assigned a score corresponding to the number of matches that they are a part of, considering the computations with the candidate node as the source. Note that only the originating node and the edges incident on this node are used to define the instances for training the node and edge regression models respectively, because the true regression scores can be evaluated only for these node and edges based on the computations from the originating node. Such labeled node and edge instances are then used to learn a node and an edge regression model respectively.

For the edge regression model, the feature vector is a concatenation of the feature vectors for the two nodes on which the edge is incident. The features for the node regression model capture the heterogeneous topological structure associated with each node as follows.

- Graph Topology: Type-wise number of nodes in the d-hop neighborhood of the node ($d \in D$)
- Graph vs Query Topology: Extra type-wise number of nodes in the d-hop neighborhood of the node compared to those expected by the corresponding query node ($d \in D$)
- Total degree of the node
- Type-wise degree of the node
- Total number of d-hop neighbors of the node ($d \in D$)

For our experiments, we use $D = 2$. Note that the candidate node identification also uses the graph versus query topology information to decide whether a node is a candidate node or not. When making this decision, the algorithm needs to ensure that the graph node neighborhood must be a superset of the neighborhood expected by the query node. However, the candidate node list computation fails to consider "how loose a superset the graph node neighborhood is compared to the expected query node neighborhood?" This signal is encoded as the "Graph vs Query Topology" feature in the node regression model as mentioned above.

Note that all these feature values can be computed using the graph SPath index when the candidate list is being constructed for every query node. Thus, feature computation does not involve any extra time. Also, unlike the SPath index, (1) the features store only the count of nodes of a particular type at a particular distance and not the node ids themselves; (2) the features are computed for a small set of graph candidate nodes. Thus, the features can be easily loaded in main memory and regression models can be easily learned.

Using the node regression model, the remaining candidate nodes are ranked by the probability of being a part of some match. The SPath match computation is then run using $\theta\%$ of these candidate nodes sorted in descending order of the probability. Thus for a smaller percentage of candidate nodes, one can now expect to find a large number of matches compared to GCOD-RS and hence higher accuracy. Edges are also sorted in descending order by edge regression model output probability. Edges below a particular threshold are removed from the graph and ignored when running SPath computations. This means that SPath computations will finish faster. We experiment by varying the percentage of candidate nodes selected and show the time-accuracy trade-off in Section IV.

The overall GCOutlier Detection Algorithm using regression models is illustrated in Algorithm 2.

IV. Experiments

Evaluation of outlier detection algorithms is quite difficult due to lack of ground truth. We perform experiments on synthetic graphs generated using the GT-Graph Software [22]. On synthetic datasets we explore the time versus accuracy trade-off. The distance parameter D for the SPath index is set to 2. The execution times are obtained by repeating the experiments 10 times each for 10 queries. We also evaluate the results on real datasets using case studies. We perform a comprehensive analysis of the objects in the top outlier cuboids returned by the proposed algorithm to justify their interestingness. The code

Algorithm 2 Graph Cuboid Outlier Detection Algorithm using Regression Models (GCOD-RM)

Input: (1) Graph G, (2) Query Q, (3) Index Parameter D, (4) Number of Outliers K (5) Sampling rate τ and θ
Output: K GCOutliers.

1: Compute SPath Index using D, and the list of Cuboid Edge Count for G once.
2: Compute node signatures NS_Q for each node in Q.
3: Use NS_Q and NS_G stored in the SPath index to compute candidate nodes for every query node.
4: $q_v \leftarrow$ Query node with smallest sized candidate list.
5: Sample with rate τ from the candidate list for q_v to obtain training data for node and edge regression models.
6: Learn a node regression model based on this training data and obtain the probability score for the remaining candidate nodes.
7: Learn an edge regression model based on this training data and obtain the probability score for the edges.
8: $CandidateList \leftarrow$ Top $\theta\%$ nodes in the descending order of the probability score from the pool of candidate nodes.
9: Compute matches using candidates in $CandidateList$ only for q_v using SPath [11].
10: Map the matches to cuboids.
11: Compute outlier score for each cuboid and rank.
12: Return top K cuboids.

and the data sets are available at: https://www.dropbox.com/s/bzo8z8zx54qdigs/GCOutlier.zip?dl=0.

A. Synthetic Datasets

For most of the results presented in this section, we used a graph with 10^4 nodes and 10^5 edges. Each node is assigned a random type from 1 to 15. Each node is assigned a random projection attribute value between 1 and 5.

Index Construction Time and Space

To verify the scalability of the approach, we used the GT-Graph software to build graphs with $10^3, 10^4, 10^5, 10^6$ nodes respectively. The number of edges is set to 10 times the number of nodes. We observed that the index construction time and the time to compute the cuboid edge count increased linearly with the size of the graph. Cuboid edge count list computation is about an order faster than the SPath index construction. However, the cuboid edge count list requires very little space (in KBs). It is constant and independent of the graph but depends upon the number of unique values for the projection attribute. We omit the plots for lack of space.

Query Execution Time Comparison

We experimented with path queries of sizes from 4 to 10. We vary the percentage of candidate nodes for a given query. Tables II show the number of matches for different percentage of candidate nodes. Note that as the query size increases, the number of matches increase, thus necessitating the use of small number of candidate nodes to compute outliers for large queries.

Figure 6 shows comparison of execution times for GCOD-RS and GCOD-RM for varying percentage of candidate nodes. The execution times are mentioned in milliseconds. Also, the graph is plotted for queries with size 4. The trend is similar for queries of other sizes too. The plot is drawn for $\tau = 0.4$. We varied τ as 0.2, 0.3 and 0.4 and again observed similar trends except that the difference in execution times was smaller for 0.2 and 0.3 compared to this plot. The difference between the two curves denotes the time required to generate the training data (i.e., run SPath for τ fraction of candidate nodes) and for training the regression models.

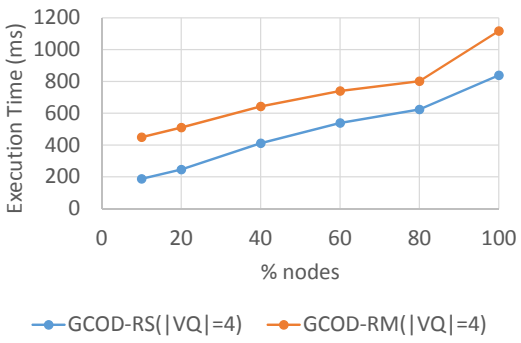

Fig. 6. Execution Time Comparison between GCOD-RS and GCOD-RM

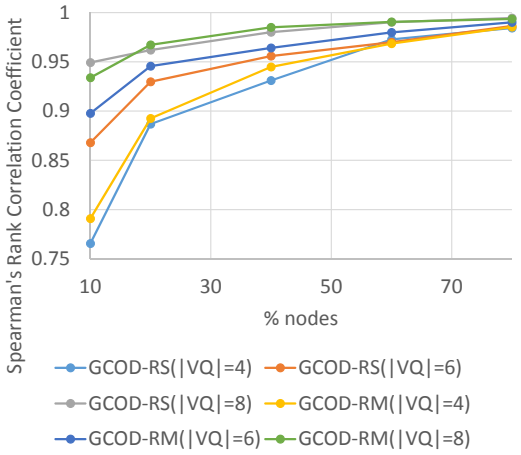

Fig. 7. Spearman's Rank Correlation Coefficient Comparison between GCOD-RS and GCOD-RM

Accuracy Comparisons

We compare the GCOutlier score rankings obtained by varying the percentage of candidate nodes for queries of different sizes. Figures 7, 8 and 9 show the comparison between GCOD-RS and GCOD-RM in terms of accuracy with respect to three metrics: Spearman's Rank Correlation Coefficient, Kendall's Tau Rank Correlation Coefficient and Precision @10.

It is interesting to find that even by taking a smaller percentage of nodes, which helps in reducing our computation time, a good accuracy is achieved. Considering just 10% of the candidate nodes provides more than 60% accuracy across all metrics. Also, in general GCOD-RM provides better accuracy compared to GCOD-RS.

Finally, we combine the execution time and accuracy (Spearman's coefficient) for the two algorithms for various query sizes to present the time-accuracy trade-off in Figure 10. As we can see, the trade-off favors GCOD-RS for smaller query sizes, but the trade-off is better for GCOD-RM for larger query sizes. We observed similar trends using other accuracy metrics.

B. Real Datasets

We experiment with two real datasets: Delicious and Four Area.

Table II. Number of Matches for Various Experimental Settings

% nodes	$\|V_Q\|$=4	$\|V_Q\|$=5	$\|V_Q\|$=6	$\|V_Q\|$=7	$\|V_Q\|$=8	$\|V_Q\|$=9	$\|V_Q\|$=10
10	645	1845	7545	11605	39240	125615	241409
20	1578	3311	16359	21212	91575	348980	469198
40	2932	7042	36787	47798	181020	714290	841570
60	4514	11037	48377	75825	255496	1088511	1255223
80	6018	14257	62922	106550	335870	1511451	1600618
100	7691	18312	79029	128562	447797	1851787	1979479

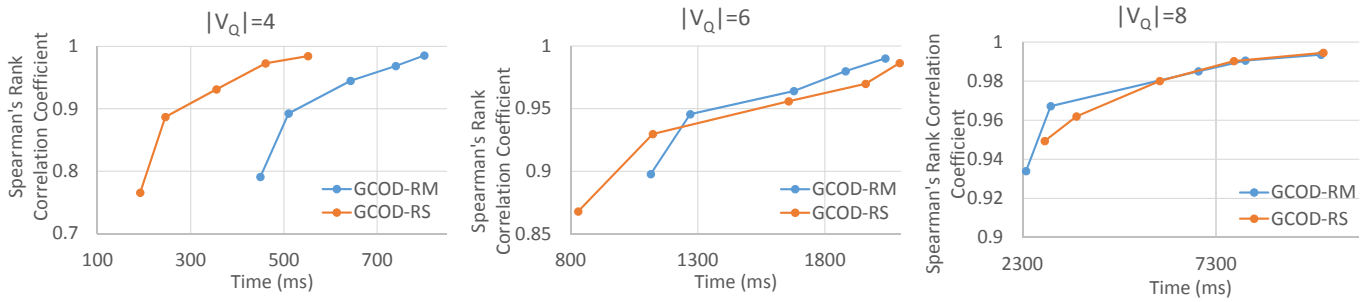

Fig. 10. Time vs Accuracy Trade-off between GCOD-RS and GCOD-RM

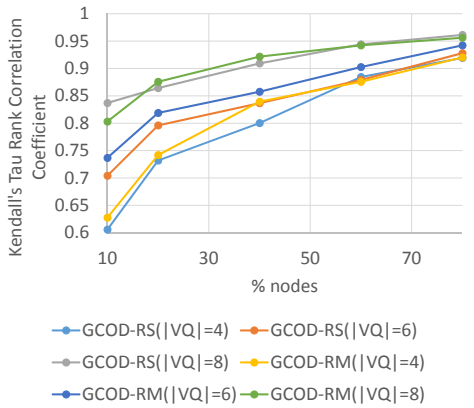

Fig. 8. Kendall's Tau Rank Correlation Coefficient Comparison between GCOD-RS and GCOD-RM

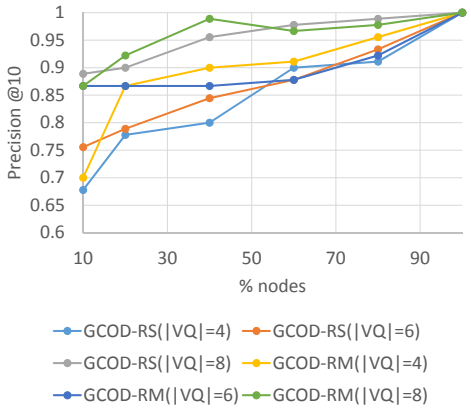

Fig. 9. Precision @10 Comparison between GCOD-RS and GCOD-RM

Delicious: The Delicious network consists of tagging events, users, URLs and tags with ∼83K nodes and ∼588K edges. Delicious provides a basic categorization on the home page which we used to associate a few tags to categories. We consider this category as the projection attribute, and hence project the graph across these 10 dimensions: Arts and design, Education, Fashion, Entertainment, Food, Lifestyle, News and Politics, Sports, Tech and Science, Travel. Personalized PageRank is used to propagate category labels from labeled tags to other nodes. Average number of urls tagged by a user is 2.59, the average number of tags for a url is 3.41 and the average number of authors tagging a particular url is 1.15.

Four Area: This is a subnetwork from DBLP for the four areas of data mining (DM), databases (DB), information retrieval (IR) and machine learning (ML) and consists of papers from 20 conferences (5 per area) till 2008. There are ∼74K nodes and ∼320K edges. Each conference is associated with one of these research areas. Research area labels are propagated to other nodes. We project the data on these research areas and ranges of years. We consider these 4 ranges: 1989-1993, 1994-1998, 1999-2003 and 2004-2008. Average number of authors for a paper is 2.62.

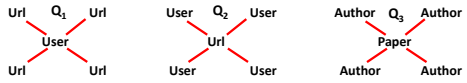

Fig. 11. Queries: Q_1 and Q_2 for Delicious Dataset, Q_3 for 4Area Dataset

SPath index construction took 9 minutes for Delicious and 1.84 hours for 4Area. SPath index sizes are ∼500MB and ∼1.84GB for Delicious and 4Area resp. Cuboid edge count list is a few KBs for both datasets. For experiments on these datasets we perform interesting sub-graph queries as shown in Figure 11.

Results on Delicious Dataset

We projected out graph across 10 dimensions. Thus, given a query, it can have matches across any of the 2^{10} possible cuboids. Based on the nodes containing the matches the possible cuboids are determined. Query Q_1 aims to find categories for which the users have tagged at least four urls. We found

that the maximum number of such queries were found in the 'Sports' category which consists of tags like bike, football, rugby and so on. Query Q_2 aims to find categories for which the url have been tagged by at least four users. We found that the maximum number of such queries were found in the cuboid of three categories viz. 'Arts and Design', 'Lifestyle' and 'Tech and Science' which consists of tags like opensource, mobile, online, makeup, spa, design, music and so on. On average, query execution time is 1.9 minutes for Q_1 and 6.9 minutes for Q_2.

Results on Four Area Dataset

We projected our graph across four areas and four range of years. This gives us a total number of 8 dimensions and hence 2^8 possible cuboids. Thus, given a query, it can have matches across any of 2^8 cuboids. Query Q_3 aims to find research areas and range of years for which the paper has been authored by four authors. It was found that the maximum number of such queries were found across the cuboid of five dimensions viz. Databases and 1989-1993, 1994-1998,1999-2003 and 2004-2008. This means that a high percentage of database papers have been written by 4+ authors across various time ranges. On average, query execution time is 5 minutes for Q_3.

V. Related Work

Outlier detection has been an active area of research since many decades. A number of surveys [12], [13] and books [14], [15] give a thorough analysis on the existing techniques in outlier detection. The proposed work is closely related to outlier analysis in graphs. Several attempts have been made on detecting outliers in graphs. Previous attempts on outlier detection were on static graphs and used techniques like Minimum Description Length [16], egonets [2], and community detection [17]. In case of temporal networks, techniques explored include similarity between graph snapshots [18], spectral methods [19], community outliers [3], [20], [21], etc. As discussed in Section I, recently query based outlier detection have become quite popular where outliers are discovered from a graph in the context of a subgraph query. However, the proposed work is different from the existing ones as it allows to detect cuboid outliers using the graph OLAP framework, from a heterogeneous network.

VI. Conclusion

We proposed the problem of finding outlier cuboids (GCOutliers) from graph OLAP with a query sensitive perspective. The GCOutlier detection task is challenging because of a large number of subgraph matches for a given query and also a large number of projection attribute values (or dimensions). We proposed two methods which performs approximate computation of GCOutliers. Using several synthetic datasets, we found a favorable time versus accuracy trade-off. Also experiments on real datasets showed interesting GCOutlier examples. In the future, we plan to answer questions like: Can we smartly select the initial set of candidate nodes for training the regression models to improve the time-accuracy trade-off further? What could be a more effective set of features for the regression model? Can we estimate outlierness density of cuboids without actually computing all matches? What could be other interesting ways of defining cuboid outlierness?

References

[1] Chen, C., Yan, X., Zhu, F., Han, J., Yu, P.: Graph OLAP: Towards Online Analytical Processing on Graphs. In: ICDM. (2008) 103–112

[2] Akoglu, L., McGlohon, M., Faloutsos, C.: Oddball: Spotting Anomalies in Weighted Graphs. In: PAKDD. (2010) 410–421

[3] Gupta, M., Gao, J., Sun, Y., Han, J.: Integrating Community Matching and Outlier Detection for Mining Evolutionary Community Outliers. In: KDD. (2012) 859–867

[4] Gupta, M., Gao, J., Yan, X., Cam, H., Han, J.: On Detecting Association-based Clique Outliers in Heterogeneous Information Networks. In: ASONAM. (2013) 108–115

[5] Qi, G.J., Aggarwal, C.C., Huang, T.S.: On Clustering Heterogeneous Social Media Objects with Outlier Links. In: WSDM. (2012) 553–562

[6] Henderson, K., Eliassi-Rad, T., Faloutsos, C., Akoglu, L., Li, L., Maruhashi, K., Prakash, B.A., Tong, H.: Metric Forensics: A Multi-level Approach for Mining Volatile Graphs. In: KDD. (2010) 163–172

[7] Gupta, M., Mallya, A., Roy, S., Cho, J.H.D., Han, J.: Local Learning for Mining Outlier Subgraphs from Network Datasets. In: SDM. (2014) 73–81

[8] Gupta, M., Gao, J., Yan, X., Cam, H., Han, J.: Top-K Interesting Subgraph Discovery in Information Networks. In: ICDE. (2014) 820–831

[9] Zhuang, H., Zhang, J., Brova, G., Tang, J., Cam, H., Yan, X., Han, J.: Mining Query-Based Subnetwork Outliers in Heterogeneous Information Networks. In: ICDM. (2014) To appear

[10] Aggarwal, C.C., Zhao, Y., Yu, P.S.: Outlier Detection in Graph Streams. In: ICDE. (2011) 399–409

[11] Zhao, P., Han, J.: On Graph Query Optimization in Large Networks. PVLDB **3** (2010) 340–351

[12] Chandola, V., Banerjee, A., Kumar, V.: Anomaly Detection: A Survey. ACM Comp. Surveys **41** (2009) 15:1–15:58

[13] Hodge, V.J., Austin, J.: A Survey of Outlier Detection Methodologies. AI Review **22** (2004) 85–126

[14] Gupta, M., Gao, J., Aggarwal, C., Han, J.: Outlier Detection for Temporal Data. Morgan & Claypool (2014)

[15] Aggarwal, C.C.: Outlier Analysis. Springer (2013)

[16] Noble, C.C., Cook, D.J.: Graph-Based Anomaly Detection. In: KDD, ACM (2003) 631–636

[17] Gao, J., Liang, F., Fan, W., Wang, C., Sun, Y., Han, J.: On Community Outliers and their Efficient Detection in Information Networks. In: KDD. (2010) 813–822

[18] Pincombe, B.: Anomaly Detection in Time Series of Graphs using ARMA Processes. ASOR Bulletin **24** (2005) 2–10

[19] Idé, T., Kashima, H.: Eigenspace-based Anomaly Detection in Computer Systems. In: KDD. (2004) 440–449

[20] Gupta, M., Gao, J., Sun, Y., Han, J.: Community Trend Outlier Detection using Soft Temporal Pattern Mining. In: ECML-PKDD. (2012) 692–708

[21] Gupta, M., Gao, J., Han, J.: Community Distribution Outlier Detection in Heterogeneous Information Networks. In: ECML-PKDD. (2013) 557–573

[22] Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-mat: A recursive model for graph mining. In: SDM. (2004) 442–446