On Detecting Association-Based Clique Outliers in Heterogeneous Information Networks

Manish Gupta*, Jing Gao†, Xifeng Yan‡, Hasan Cam§ and Jiawei Han¶
*Microsoft, India. Email: gmanish@microsoft.com
†State University of New York at Buffalo. Email: jing@buffalo.edu
‡University of California, Santa Barbara. Email: xyan@cs.ucsb.edu
§US Army Research Lab. Email: hasan.cam.civ@mail.mil
¶University of Illinois at Urbana-Champaign. Email: hanj@cs.uiuc.edu

***Abstract*—In the real world, various systems can be modeled using heterogeneous networks which consist of entities of different types. People like to discover groups (or cliques) of entities linked to each other with rare and surprising associations from such networks. We define such anomalous cliques as Association-Based Clique Outliers (*ABCOutliers*) for heterogeneous information networks, and design effective approaches to detect them. The need to find such outlier cliques from networks can be formulated as a conjunctive select query consisting of a set of (type, predicate) pairs. Answering such conjunctive queries efficiently involves two main challenges: (1) computing all *matching* cliques which satisfy the query and (2) *ranking* such results based on the rarity and the interestingness of the associations among entities in the cliques. In this paper, we address these two challenges as follows. First, we introduce a new low-cost graph index to assist clique matching. Second, we define the outlierness of an association between two entities based on their attribute values and provide a methodology to efficiently compute such outliers given a conjunctive select query. Experimental results on several synthetic datasets and the Wikipedia dataset containing thousands of entities show the effectiveness of the proposed approach in computing interesting *ABCOutliers*.**

I. Introduction

With the ever-increasing popularity of entity-centric applications, it has become very essential to study the interactions between entities, which are captured using edges in entity-relationship graphs. Entity-relationship graphs can be regarded as heterogeneous information networks due to the heterogeneity in entity types. For example, bibliographic networks capture associations like 'an author wrote a paper' or 'an author attended a conference'. Similarly, social networks, biological protein-enzyme networks, Wikipedia entity network, etc. also capture a variety of rich associations containing entities of different types.

Usually a certain group of the entities of a particular type interact with entities belonging to a particular group of the same or another type. However, certain associations are quite different from such normal association trends. Finding such *rare and interesting* entity associations becomes immediately critical for discovering interesting relationships and/or data de-noising, i.e., removing incorrect data attributes or entity associations. Specifically, if a user is interested in a set of entity types, each following certain predicates, we could discover some unexpected cliques which contain unusual associations amongst its entities, besides satisfying the user query predicates. Given a conjunctive select query, our goal is to detect such unusual clique outliers as Association-Based Clique Outliers (*ABCOutliers*) in heterogeneous information networks. Unusual associations can be helpful in de-noising the data, especially in situations when the data is obtained from unstructured sources and contains significant amount of noise.

Movie Network Example Most of the actors act in movies made in their native language and produced in their own country. Also, there are known associations between countries and languages. For example, Chinese movies in China, Hindi movies in India, English movies in USA or UK, etc. These are all normal associations. However, it will be very surprising to find an actor of American origin acting in a Vietnamese movie being produced in China, for a ⟨(actor, movie, country)⟩ query. These associations contain rare co-occurrences of specific entity attribute values participating in the associations.

While the example above captures a typed query without any predicates, our aim is to build a system which also allows queries with predicates associated with each entity type. This will allow the user to filter the entities and drill down the search for *ABCOutliers* to more interesting subspaces of the dataset.

Brief Overview of *ABCOutlier* Detection Given a heterogeneous network containing entities of various types, and a conjunctive select query which contains (type, predicate) pairs, the aim is to find the top outlier cliques[1] in the network that match the query. The proposed approach consists of two steps. In the first step, we find all possible matching cliques from the network. Next, we compute outlier scores for each match by aggregating the outlierness of each of its edges. We define the outlierness of an edge or an entity association as the average outlierness of attribute associations between the two entities with respect to the entire set of matches. The outlierness of an association between two attribute values is based on (1) degree of correlation of the attribute values with those of the other attribute, and (2) frequency of individual occurrence and

[1]Clique is a subgraph with each node connected to all other nodes.

co-occurrence of values. Further, this computation is done in top-K manner so that we do not need to evaluate the attribute association outlierness across all attributes and all edges.

Summary

- To the best of our knowledge, the proposed notion of Association-Based Clique Outliers, *ABCOutliers* is the first work on *query-based* outlier detection for *heterogeneous* information networks.
- We propose a top-K methodology to rank cliques based on the association outlierness of the attributes for its entities.
- Extensive experiments on both synthetic datasets and Wikipedia network show the effectiveness of the proposed approach.

Organization We define the *ABCOutlier* detection problem in Section II. In Section III, we discuss the algorithm that computes all candidates (matching cliques) satisfying the user query. The methodology to rank these candidates based on outlierness scores is discussed in Section IV. We present results on several synthetic datasets and the Wikipedia dataset with detailed insights in Section V. We discuss related work and summarize the paper in Sections VI and VII respectively.

II. Problem Definition

We start with an introduction to some basic concepts.

Heterogeneous Information Network A heterogeneous information network is an undirected graph $G = (V, E)$ where V is the finite set of vertices (representing entities) and E is the finite set of edges (representing relationships between entities) each being an unordered pair of distinct vertices. Each entity has a type from the set $\mathcal{T}$. Each type has a fixed set of attributes. Thus, each entity is associated with a vector of attribute values of size D_t where t is the type of the entity. These attributes could be numeric, categorical, sets of strings, time durations, etc. G is called heterogeneous because the #types of entities (denoted by T) is >1. Thus $|\mathcal{T}| = T > 1$. Let the set of attributes associated with type T_i be denoted by A_i=($A_{i_1}, A_{i_2}, \ldots A_{i_{D_i}}$) where D_i=$|A_i|$. For example, a Wikipedia entity network can consist of entities like "Barack Obama" or "Taj Mahal" which belong to the types "person" and "place" respectively. The person entity type has attributes like birth date, profession, children, spouse, religion, etc. Figure 2 shows a network with T=3.

Conjunctive Select Query on a Network A conjunctive select query Q of length L on a network G consists of (type, predicate) pairs $\langle(T_1, P_1),(T_2, P_2), \ldots, (T_L, P_L)\rangle$ and aims to obtain all matching cliques containing one entity each of the types T_1, T_2, …, T_L satisfying the predicates P_1, P_2, …, P_L respectively. Each of the predicates P_i is defined only on the attributes of type T_i and consists of conjunctions of atomic conditions, i.e., conditions constructed from attributes and constants using various comparison operators combined using "and." In addition, each of the entities in a particular match should be linked to each other in G. In general, the query may have multiple nodes of the same type. But for simplicity of notation, we will assume that all the nodes in the query are of different types. Figure 2 shows a query with L=3.

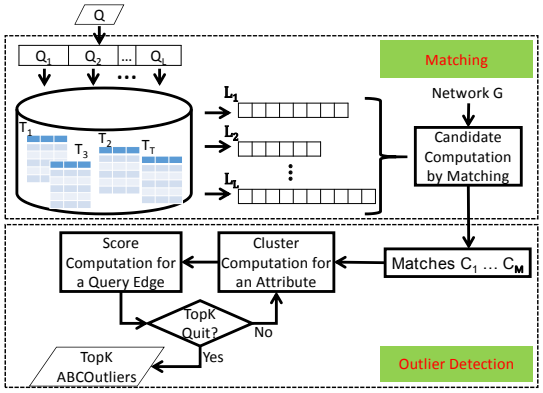


Fig. 1. *ABCOutlier* Detection System Diagram

Match A match C for a query Q is a clique such that it contains the same number and the same type of entities as mentioned in the query. All entities contained in C are connected to each other in G and satisfy the predicates in Q. The i^{th} match C_i for query Q of length L can be expressed as $C_i = \langle e_{i_1}^{D_1 \times 1}, \ldots, e_{i_L}^{D_L \times 1}\rangle$ where $e_{i_j}^{D_j \times 1}$ denotes an entity of type T_j associated with an attribute vector of size D_j.

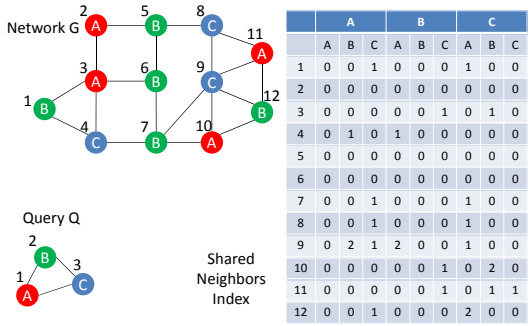
Association-Based Clique Outlier A clique C returned as a result for a conjunctive select query Q on a network G is an *ABCOutlier* if the associations within the pairs of entities in C are highly unusual as expressed in terms of the associations between the attributes of these entities. An association defined by an attribute value pair is unusual if both values (or their cluster representatives) frequently co-occur with some other particular value (or its cluster representative) different from the one in this pair. Consider an association between values v_1 and v_2 for attributes a_1 and a_2. The association is interesting if (1) there is a high correlation between values (or their clusters) of attributes a_1 and a_2, (2) v_1 and v_2 are individually frequent, and (3) co-occurrence of v_1 and v_2 is rare (i.e., low joint probability). Note that this definition is a stronger version of negative associations [11].

For example, consider a size-2 query $\langle$(person, gender = "male"), (movement, true)$\rangle$. "Mahatma Gandhi" and "Civil Rights movement in South Africa" can be marked as an *ABCOutlier* for such a query based on the anomalous association between attributes a_1="nationality of person" and a_2="country of the movement".

Association-Based Clique Outlier Detection Problem The proposed problem can be expressed as follows. Given a network G and a query Q, find the top K cliques that satisfy the query with the highest outlier scores. Figure 1 shows a broad overview of the proposed system. Given a conjunctive select query Q, the top half shows how to compute the matches $C_1, C_2, \ldots, C_M$, while the lower half illustrates how to compute the top-K *ABCOutliers* from these candidates.

III. Candidate Computation by Matching

How to compute matching candidates given a network G and a query Q? A naïve way is to perform an exhaustive search of all possible one-to-one correspondences of query



	A			B			C		
	A	B	C	A	B	C	A	B	C
1	0	0	1	0	0	0	1	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	1	0
4	0	1	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	1	0	0
8	0	0	1	0	0	0	1	0	0
9	0	2	1	2	0	0	1	0	0
10	0	0	0	0	0	1	0	2	0
11	0	0	0	0	0	1	0	1	1
12	0	0	1	0	0	0	2	0	0

Fig. 2. An Example Graph, Query and Shared Neighbors Index

node to graph nodes of the matching type. But as pointed out in [12], such a solution would be quite inefficient. ($O(N^T)$ in the worst case where N is #entities in the network and T=#types.)

A. Indexing the Network and Computing Lists

The attribute information associated with each of the entities in G is stored in a relational DB. For every type T_i, we create a table with D_i columns. Note that the DB is constructed just once offline. The connectivity information of the network is stored separately as entity pairs (along with type information) in memory.

Offline Index Construction The network connectivity information is also stored in a "shared neighbors index." This index stores for each entity, the #shared neighbors of each type which are shared between the entity and its neighbors of a particular type. For example, in Figure 2, consider entity 9 in the network. It has the following neighbors: 2 of type A (10 and 11), 2 of type B (7 and 12) and 1 of type C (8). Now, the #neighbors shared between 9 and 10 is 2 of type B (7 and 12). Also, the #neighbors shared between 9 and 11 is 1 of type B (12) and 1 of type C (8). Thus, overall node 9 shares 2 B and 1 C neighbors, with its type A neighbors (10 and 11). Hence, the entry (9, A, B) is 2 and the entry (9, A, C) is 1.

The shared neighbors index is important for filtering candidates as explained in the next sub-section. For a graph containing N nodes and T types, the shared neighbors index has size $O(NT^2)$ but can be very sparse and hence can be stored in memory.

Online Computation of Lists Given a query Q with L (type, predicate) pairs, it is split into L different relational queries such that each query corresponds to a select on a table of a particular type with the corresponding predicate. Thus, a query Q=$\langle(T_1, P_1), (T_2, P_2), \ldots, (T_L, P_L)\rangle$ is split into L queries $Q_1 = \langle(T_1, P_1)\rangle$, $Q_2 = \langle(T_2, P_2)\rangle$, $\ldots$, $Q_L = \langle(T_L, P_L)\rangle$. This is possible because each predicate P_i is defined over T_i only and is independent of other types. Running each of these L queries on the DB gives a list of the entities that match the user-specified predicate. Each list $L_1, L_2, \ldots, L_L$ could be of a different size.

B. Candidate Filtering

Given multiple lists $L_1, L_2, \ldots, L_L$, we need to find cliques formed out of entity nodes in these lists such that each clique contains exactly one entity from each of the lists and all the entities are connected to each other in G. The pseudo-code for the methodology is presented in Algorithm 1. We start with the most selective type T_{min} (C in Figure 2) with the minimum #entities (L_{min}) satisfying the query predicate (Step 1).

We remove candidates from this list, which cannot be a part of any match for the query. The pruning is done in two ways. First, a node is pruned if its typed neighbors cannot satisfy the requirements of the query. For example, in the query Q, node 3 of type C needs a neighbor of type A and a neighbor of type B. Since all nodes of type C in L_C (4, 8, 9) have at least 1 neighbor of type A and B, none will be pruned. Second form of pruning uses the shared neighbors index. The query needs node 3 and its type A neighbor to share at least 1 neighbor of type B. From the shared neighbors index, we can see that node 8 shares 0 neighbors of type B with its neighbor of type A (11). Hence, node 8 can be pruned from list L_C. Such pruning of lists can be done for all the lists, but pruning only list L_{min} may itself be sufficient to discard a large number of possible candidates (Step 2 of Algorithm 1).

Algorithm 1 Candidate Computation by Matching

Input: (1) Query Q with L (type, predicate) pairs, (2) Network G, and (3) Candidate lists $L_1, L_2, \ldots, L_L$.
Output: Matched Cliques $C_1, C_2, \ldots, C_M$ ($currMatches$).
1: $(T_{min}, L_{min}) \leftarrow$ (Type with smallest list size, Smallest list).
2: $currMatches \leftarrow CandidateFiltering(L_{min})$.
3: **for** Type $t = T_1 \ldots T_L$ **do**
4: **if** $t \neq T_{min}$ **then**
5: $newMatches \leftarrow \phi$.
6: **for** Each clique $C_i \in currMatches$ **do**
7: $C_{i_1} \leftarrow$ The first element of C_i.
8: $N \leftarrow$ Neighbors of C_{i_1} of type t (using G).
9: $N \leftarrow N - \{entity\ e | e \notin list\ L_t\}$.
10: **for** $j = 2 \ldots size(C_i)$ **do**
11: $N \leftarrow N - \{entity\ e | (e, C_{i_j}) \notin E(G)\}$.
12: **for** $n = 1 \ldots size(N)$ **do**
13: $newMatches \leftarrow newMatches \cup \{C_i.n\}$
14: $currMatches \leftarrow newMatches$.

C. Generating Candidates

The elements of the pruned list form the length-1 candidate matches. We keep adding one type of entities to the $currMatches$ of length-l to get the $newMatches$ of length $l+1$ (Steps 3 to 14) as follows. We randomly choose the new entity type t to be added from the set of the entity types remaining to be explored. Consider a clique in the $currMatches$ C_i of length l. The new entities of type t are added to C_i to get the matches of length $l+1$ such that the newly added entity is connected to each of the entities already in C_i (Step 13). Matches are grown one entity at a time until they get pruned out when there is no entity of type t connected to all other entities in C_i or its size equals the size of query.

For example, for the graph and the query in Figure 2, we start with length-1 candidates $\{\{4\}, \{8\}, \{9\}\}$ of type C. Next, we grow each of these candidates as follows. Let the next type be B. Thus, candidate $\{4\}$ grows to $\{\{4,1\}, \{4,7\}\}$. Similarly, we obtain more candidates of length 2 using other length-1 candidates. Finally, when we consider the third type A for the query, the candidate $\{4,1\}$ grows to $\{4,1,3\}$, while some candidates like $\{4,7\}$ get pruned. The algorithm terminates after exploring all candidates of length 3. The

returned result is the list of all matches that satisfy the query: $\{\{4,1,3\},\{9,7,10\},\{9,12,10\},\{9,12,11\}\}$.

Algorithm Complexity The candidate filtering step is $O(N/T \times T^2)$ where N = #entities in the network and T = #types. Overall time complexity of the matching algorithm is $O(N/T(LB + T^2))$ where L = #query nodes and B is the average #neighbors of an entity node of a particular type. In practice, pruning reduces the size of the list L_{min} by a large amount, and hence the execution times are typically very small.

IV. Outlier Score Computation

We will first discuss the outlier score computation for a pair of attribute values, and then use it to define edge and clique outlierness. Further, we will design a method to obtain top K results efficiently.

A. Scoring Attribute Value Pairs

Consider a value v_i for an attribute A_{i_k} of type T_i and the value v_j for an attribute A_{j_l} of type T_j. Let s_i and s_j denote small set of values for attributes A_{i_k} and A_{j_l} respectively. The outlier score of the association (v_i, v_j) should reflect the surprise factor in witnessing such an association. The outlier score should be high if (1) v_i and v_j co-occur rarely, (2) v_i and v_j are individually frequent, (3) $\exists s_j$ such that the co-occurrence frequency of v_i and $s_j > \gamma\times$ the total co-occurrence frequency of v_i wrt any value of A_{j_l}, and $v_j \notin s_j$, and (4) $\exists s_i$ such that the co-occurrence frequency of v_j and $s_i > \gamma\times$ the total co-occurrence frequency of v_j wrt any value of A_{i_k}, and $v_i \notin s_i$. Here $0 \leq \gamma \leq 1$ and γ is set to a value close to 1.

For example, consider the value "Hindi" for the attribute "language" of person entity and its association with the value "China" for an attribute "country" of entity "settlement". (1) "Hindi" and "China" co-occur rarely. (2) "Hindi" and "China" are individually frequent. (3) Other languages like Chinese, Mongolian, Tibetan constitute $> 95\%$ of languages related to "China". (4) Other countries like India, Pakistan constitute $> 95\%$ of countries related to "Hindi". Hence, this association can be considered as an outlier association.

Clustering of Entity Attributes Computing the above measures for *individual values* may lead to very noisy results especially if the attribute has a large range of values. Hence, we propose to compute the above two measures over the *clusters* of attributes A_{i_k} and A_{j_l}. The attributes A_{i_k} and A_{j_l} could be of various data types, for example, numeric (years, elevation, age), time durations (active period, duration of event), etc.

We discuss the different choices of clustering algorithms for different data types in the following. The clustering of numeric data can be done using K-Means algorithm with Euclidean distance as the distance measure. Categorical data can be simply clustered using the category label. Time durations can be clustered using K-Means algorithm on the centers of the time intervals. Sets of strings could be clustered by first creating a network where strings belonging to the same set are linked to each other, and then partitioning the network using some graph partitioning algorithm like METIS [8]. For example, the "actors" attribute of the type "film" can be clustered by first creating a co-starring network and then partitioning such a network using METIS.

Note that for an association to be interesting, one important condition is that the values (or their respective clusters) participating in the association should be frequent. Hence, as a post-processing step, we remove small clusters. Also note that the patterns discovered and hence the outliers detected will depend on the particular clustering algorithms used, as well as on their parameter values.

Peakedness of Cluster Co-occurrence Curves Given a cluster for an attribute A_{i_k} for the entities of type T_i, we can plot its distribution wrt the clusters of some other attribute A_{j_l} for the entities of type T_j based on cluster associations across the dataset. We can rearrange the points so that we obtain a monotonically non-increasing probability density curve. The points can be normalized such that the distribution adds up to 1. Now, if the distribution is highly peaked, it implies that this cluster of A_{i_k} has a very high correlation with one or a few clusters of A_{j_l}. Thus, establishing an outlier score for the values of A_{i_k} based on its association with the clusters of A_{j_l} is meaningful. If the curve is flat, we can conclude that there is no association pattern between the particular cluster of attribute A_{i_k} and clusters of A_{j_l} and hence this association should not be used for outlierness computation. For example, consider the cluster "Hindi" for the attribute "language" of entity type "film". Nationality of persons related to Hindi movies is frequently "India" and rarely any other nationality. Thus, this cluster has very high peakedness wrt the "nationality" attribute of related "person" entities. On the other hand, consider the "1983" cluster of "birth date" attribute of "person" entities. "1983" is not related to a particular small set of latitudes. Hence, the association between "birth date" of "person" and "latitude" of "settlement" entity should not be used to compute outlierness.

We measure the peakedness of a monotonically non-increasing probability distribution using a simple function defined as follows. Let c_{i_k} denote a cluster of the attribute A_{i_k}, then the peakedness for the association curve of values in cluster c_{i_k} wrt the cluster of associated values for attribute A_{j_l} can be computed as $peakedness(c_{i_k}, A_{j_l}) = max(0, 1 - \beta \times (|c_j| - 1))$ where $|c_j|$ is the #most frequent clusters of the attribute A_{j_l} which together share a total co-occurrence (γ) of $\geq$95% with the cluster c_{i_k}. β controls the degree to which the curve peakedness affects the outlier score, and can be tuned based on the nature of the normal association patterns in the data. We set β to 0.05 in our experiments. Note that this function captures a linear decay in the peakedness value as the #co-occurring clusters increases. Thus, if the histogram is flat, the #co-occurring clusters will be very high and peakedness measure will be 0. On the other hand, if the correlation between c_{i_k} and A_{j_l} is very high, the #co-occurring clusters may be just 1 and will lead to a peakedness value of 1. For example, for "China" and languages "Mandarin, Southern,

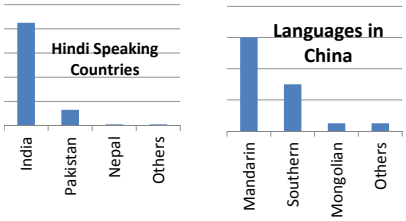


Fig. 3. Peakedness(Hindi, country) and Peakedness(China, language) are both High

Mongolian", peakedness=$1 - 0.05 \times (3-1) = 0.9$, while if "1983" is linked to 100 different latitude values, peakedness will be $max(0, 1 - 0.05 \times (100-1)) = 0$. Figure 3 shows high peakedness cluster and attribute pairs.

Outlier Score of an Association Based on the definition of outliers presented in Section II and the concept of peakedness presented above, the outlier score for the association of the values v_i and v_j can be computed as follows. Let c_{i_k} and c_{j_l} represent the clusters of attributes A_{i_k} and A_{j_l} which contain the values v_i and v_j.

$$score(v_i, v_j) = \frac{freq(c_{i_k}) \times freq(c_{j_l})}{freq(c_{i_k}, c_{j_l})} \times \frac{peakedness(c_{j_l}, A_{i_k}) + peakedness(c_{i_k}, A_{j_l})}{2} \quad (1)$$

where $freq(c_{i_k}, c_{j_l})$ denotes the co-occurrence frequency of values in cluster c_{i_k} with values in cluster c_{j_l}, $freq(c_{i_k})$ and $freq(c_{j_l})$ denote the frequencies of the clusters c_{i_k} and c_{j_l} respectively. The first part of Eq. 1 ensures that the score is high only if the negative association is strong (captures the rarity of co-occurrence) and the second part ensures the score is high only if it makes sense to compute negative associations based on this attribute cluster pair (captures the interestingness of this rarity). For example, for link between "Hindi" and "China", the negative association strength as well as average peakedness is high, and hence the outlier score will be high.

B. Scoring Cliques

The outlier score for an edge e between the entities e_i and e_j of types T_i and T_j respectively, is defined as the average outlier score between the values of the attributes A_i for entity e_i and the values of attributes A_j for entity e_j. After computing the outlier score for every attribute pair of an edge across all candidate cliques, we normalize the scores such that the maximum outlier score of an attribute pair lies between 0 and $\frac{1}{D_i \times D_j}$. Thus, outlier score of any edge lies between 0 and 1.

$$score(edge\ e) = \frac{\sum_{a=1}^{D_i} \sum_{b=1}^{D_j} score(v_a, v_b)}{D_i \times D_j} \quad (2)$$

Note that $score(v_a, v_b)$ is computed based on attribute cluster co-occurrence patterns for *matching cliques for this query* and hence, we need to re-compute these edge scores for every query. Thus, the outlier computation is performed on the data returned after filtering the network based on the predicates on the user query. The outlier scores for the edges will therefore depend on the user query and hence need to be computed online.

We define the outlier score of a clique as the sum of the outlier score of its edges. Though such a definition loses some of the information regarding the outlierness of ternary, quaternary and higher order associations, it makes the computation much faster for a clique of arbitrary size.

$$score(clique\ c) = \sum_{e \in c} score(edge\ e) \quad (3)$$

Thus, the outlier score for a clique of size L lies between 0 and $L(L-1)/2$.

C. Top-K Outlier Score Computation

Since the outlier score of a clique is a linear composition of the outlier scores of various attribute pair scores, the application of top-K pruning [3] becomes natural. The outlier detection algorithm with the top-K pruning check is shown in Algorithm 2. The query is processed one type (Step 4) and one attribute (Step 5) at a time. The list of processed types are maintained in $ProcessedTypes$. For every new type t and attribute A_{t_j}, the following steps are done: (1) clusters are computed for the attribute A_{t_j} of type t (Step 6), (2) small clusters are removed (Step 7), (3) outlier score is computed for each edge connecting the $ProcessedTypes$ to type t wrt A_{t_j} and each attribute of these $ProcessedTypes$ (Steps 11 to 12), (4) scores are normalized so that the score of any match for an attribute pair of types t and t' is less than $\frac{1}{D_t \times D_{t'}}$ (Step 13), (5) the actual scores and upper bound scores are updated (Steps 14 and 15), and (6) top-K pruning check is done (Step 18). After each new type is processed, it is added to the set of $ProcessedTypes$ (Step 19). After computing the outlier score of an attribute pair with types t and t', upper bound of the outlier score for each match C_i is updated as shown in Eq. 4.

$$UpperBoundScores(C_i) = ActualScores(C_i) + \left[\frac{L(L-1)}{2} - pe\right] + \frac{1}{D_t \times D_{t'}} \times [D_t \times D_{t'} - pa] \quad (4)$$

where pe is the #already processed edges out of $\frac{L(L-1)}{2}$ and pa is the #already processed attribute pairs out of $D_t \times D_{t'}$ attribute pairs for the current edge. Note that besides finding outliers, as a by-product of this algorithm, we also obtain the frequent patterns of association among the various attributes of different types.

For example, for Figure 2, there are 3 types of nodes in the query. Let A, B and C have 4, 5 and 6 attributes. We start with a particular type, say A. For the first type, clustering is performed for each of the 4 attributes of type A. Next, associations with type B are considered one attribute at a time. Clustering is performed for the first attribute of type B, association scores are computed for the instantiations of edge AB in the matches using clusters of attributes of A and the first attribute of B. The association score for the instantiations of edge AB is completely computed only when all attributes of type B have been processed. However, upper bound scores are also maintained and further edges and attributes will not be processed if the top-K criteria (maximum upper bound score of any candidate not in the top-K is less than the minimum actual score of the top-K) is satisfied. Otherwise, the algorithm proceeds to process attributes of type C one by one, and computing outlier scores for the instantiations of edges AC and BC in the matches.

Algorithm 2 *ABCOutlier* Detection Algorithm

Input: (1) Candidate Matches $C_1, C_2, \ldots, C_M$, (2) Query Q, (3) Frequency Threshold ψ.
Output: Top-K *ABCOutliers*.

$ProcessedTypes \leftarrow \phi$.
$UpperBoundScores \leftarrow \phi$.
$ActualScores \leftarrow \phi$.
for Each type t in query Q **do**
  for Each attribute A_{t_j} of type t **do**
    Compute clusters for attribute A_{t_j}.
    Remove clusters with size $< \psi$.
    for Each type $t' \in ProcessedTypes$ **do**
      for Each attribute $A_{t'_k}$ of type t' **do**
        $AttrPairScores \leftarrow \phi$.
        for Each match C **do**
          $AttrPairScores \leftarrow$ Score wrt values of A_{t_j} & $A_{t'_k}$ (Eq.1)
        Normalize $AttrPairScores$ such that max entry is $\frac{1}{D_t \times D_{t'}}$.
        Update $ActualScores$ (Eqs. 2 and 3) using $AttrPairScores$.
        Update $UpperBoundScores$ (Eq. 4) using $AttrPairScores$.
        $(u_1, u_2, \ldots, u_M) \leftarrow$ Matches sorted asc wrt $ActualScores$.
        if $UpperBoundScores(u_{K+1}) < ActualScores(u_K)$ **then**
          Top-K Quit. Return.
  $ProcessedTypes \leftarrow ProcessedTypes \cup \{t\}$.

Algorithm Complexity Overall complexity of the cluster computation phase is $O(LDM\kappa I)$ where $L =$ #query nodes, $D =$ average #attributes for an entity, $I =$ #iterations, $\kappa =$ #clusters and $M =$ #matches. Outlier score computation for an edge (Eq. 2) takes $O(D^2)$ time, and therefore the outlier detection algorithm has a complexity of $O(ML(L-1)D^2/2)$. Max M could be N/T. Overall time complexity of the overall pipeline is thus $O(\frac{N}{T} \times (LB + T^2 + LD\kappa I + \frac{L(L-1)D^2}{2}))$. For small queries over large networks, the methodology is linear in the #entities in the network.

V. Experiments

A. Baselines

Since this is a first work on query based outlier detection, it is difficult to find a baseline to compare with. We explore two baselines: *EBC* (Entity Based Clique Outlier Detection) and *CBA* (Community Based Association Outlier Detection). For *EBC*, attributes of an entity are individually clustered and an entity is marked as an outlier depending on the number of attributes for which its value is anomalous compared to values for the same attribute for other entities. Outlierness of the clique is computed as the aggregate outlierness of the entities in the clique with appropriate normalization. For *CBA*, the outlier score for an edge is defined as the KL-divergence between the community distributions of its end points. Community distribution for each entity is computed as follows. The entity network is augmented with categorical and sets-of-strings attribute values as nodes. Entity nodes are linked to each of their attribute nodes. Attribute nodes within the same set of strings are linked to each other. METIS [8] is used to compute hard partitions (K=20) on such an augmented network. Further, the cluster labels of all the attribute neighbors of an entity are aggregated to get its soft community distribution and normalized.

B. Synthetic Datasets

We generate a variety of synthetic datasets to test multiple settings of #entities and relationships in the network, #attributes associated with the entities, #entity types, etc. We vary the #entities (N) as 10K, 20K and 50K. The #attributes per entity is varied as 4, 6 and 10. We use ψ=0.01. For each such experimental setting the dataset is generated as follows.

Dataset Generation We randomly choose a type for each entity. Next, we set #clusters per attribute to 5. Then, we generate fixed #association patterns such that a pattern represents a set of attribute clusters of one type linked to a set of attribute clusters of another type. These denote the normal co-occurrence patterns. For every entity, depending on its type, we randomly choose a set of attribute clusters from the generated patterns. Each cluster is associated with a group of numeric values. For each attribute of the entity, we choose a value randomly for the cluster corresponding to that attribute. Next, we generate all possible edges between the entities such that they follow the association patterns. Finally, we randomly select $10 \times N$ edges where N is the #entities in the network.

Outlier Injection For a given degree of outlierness (Ψ = 2%, 5%, 10%), we choose a random set R of entities in which we will inject outlierness. We inject outliers by corrupting the attribute values of entities. Essentially, for each entity in set R, the values of its attributes are either chosen from some random cluster or chosen as a value outside of any of the valid cluster ranges. Random assignment of such attribute values breaks the association patterns thereby injecting *ABCOutliers*.

Results on Synthetic Datasets For each experimental setting, we run 20 different queries and report the average precision in Table I for 10 types. We found that the results are similar for #types = 4, 6, 10, but show results only for 10 types for lack of space. The values denote the average precision with which the two approaches (the proposed approach *ABC* and the baseline *EBC*) can detect the injected outliers. Note that in this case, recall is the same as precision. The variance values for the *ABCOutlier* and *EBC* algorithms are 2% and 3% respectively. On an average, the #matches for a query were 2136, 4252 and 10621 for datasets of sizes 10K, 20K and 50K respectively. As we can see, the proposed approach performs much better than the baseline. The baseline approach cannot detect the outliers injected by choosing attribute values randomly from among the values for other clusters. Since the proposed approach is association based, it can easily detect such outliers. We also performed experiments using the *CBA* baseline. However, the accuracy values are very low using *CBA*. This is simply because the outliers discovered using community distributions of entities are very different from the outliers discovered considering associations of entity attributes. This further shows that the outliers detected by the proposed approach are quite different from those detected by conventional approaches.

C. Real Dataset

We extracted an entity network from Wikipedia Infobox pages with ∼760K entities and ∼4.1M edges. Wikipedia Infobox data for all these entities is used to extract the attributes and type for the entities. We use ψ=0.01.

Case Study 1 Say the user is interested in finding movies released in US, which are linked to unusual people and

N	Ψ (%)	#Attributes=4		#Attributes=6		#Attributes=10	
		ABC	EBC	ABC	EBC	ABC	EBC
10000	2	**86.6**	75.8	**91.6**	74.7	**95.5**	68
	5	**93.7**	77.6	**93**	79.9	**94.2**	72
	10	**93.4**	73.8	**93.1**	76.5	**96**	72.3
20000	2	**96.9**	72.7	**94.6**	73	**92.3**	64.4
	5	**97.3**	75.1	**94.4**	78.6	**90.9**	75.1
	10	**97.4**	74.8	**96.7**	76.7	**94.5**	74.7
50000	2	**90.3**	69.5	**95.8**	76.9	**95.5**	65.8
	5	**92.9**	68.8	**94.5**	73.1	**95.5**	77.6
	10	**90.8**	79.3	**97.5**	78	**94.9**	66.5

TABLE I
AVERAGE PRECISION ON SYNTHETIC DATASETS (*ABC*=THE PROPOSED ASSOCIATION BASED ALGORITHM ABCOUTLIER, *EBC*= ENTITY BASED CLIQUE OUTLIER DETECTION) (#TYPES=10)

locations. This can be captured using the query: ⟨(film, country = "us"), (person, true), (settlement, true)⟩. There were 16271 cliques reported for this query as matches. The top outlier clique for this query is (film = "the road to el dorado", person = "hernán cortés", settlement = "seville"). **The Road to El Dorado** is a 2000 animated adventure musical comedy film produced by DreamWorks. The movie begins in 16^{th} century (1519) **Seville** (in the south of Spain) and contains a plot that relates to **Hernán Cortés**' fleet to conquer Mexico.

We briefly explain the outlierness of this clique as follows. (1) The movie screenwriters are American screenwriters and so are usually related to places where geographic subdivisions are cities or ceremonial counties. Similarly, Seville's subdivision "comarca" (Spanish local administrative division) is generally related to screenwriters from Spain, Portugal, Nicaragua, Panama, or Brazil. Thus, the association of American screenwriters with Spanish locations is unusual. (2) Hernán Cortés lived in the 15^{th}-16^{th} century time frame. At that time there were no "comarca"s in Spain. Hence, this association of a medieval state with a modern local administrative division is unusual. (3) The movie screenwriters are usually associated with "us-ny", "us-ca" coordinates, while the settlement is from "es" (Spain) coordinates. (4) Comarca is a modern name and is usually associated with 20^{th} and 21^{st} century. Thus, the link between a person who lived in 15^{th} century and a modern settlement is unusual.

Case Study 2 It might be interesting to find American companies with unusual links to Americans, and which are also unusually linked to English movies as well as TV shows. This can be captured using the query: ⟨(company, location_country = "united states"), (film, language = "english"), (person, birth_place = "united states"), (television, true)⟩. There were 1110 cliques reported for this query as matches. The top outlier clique for this query is (company = "viacom", film = "mission:impossible iii", person = "tom cruise", television = "south park"). A blog entry of Hollywoodinterrupted.com in March 2006 alleged that **Viacom** canceled the rebroadcast of the **South Park** episode "Trapped in the Closet" due to threats by **Tom Cruise** to refuse to participate in the **Mission: Impossible III** publicity circle.

We briefly explain the outlierness of this clique as follows. (1) The movie writers are deeply connected to other writers who write movies mostly distributed by 20th Century Fox Animation, Fox 2000 Pictures, Walt Disney Animation and DreamWorks Animation. However against this trend, these writers have written movies distributed by MTV Networks, BET Networks, and Paramount Pictures Corporation. (2) The creators of this TV serial lie in a cluster which consists of creators who are usually related to the companies of smaller sizes – usually below 500. But here the creators are linked to a big company through South Park. (3) Usually Viacom and other companies related to its subsidiaries are involved in making television serials with either a large number of episodes (250+) or a very small number of episodes (< 10) unlike 223 episodes in this case. (4) Usually, Comedy Central is related to company divisions like 20th Century Fox Animation, Walt Disney Animation, Rough Draft Feature Animation, RDS Animations. Also company divisions like MTV Networks, BET Networks, Paramount Pictures Corporation are usually related to CBS, UPN, Cartoon Network. Thus the association between the TV show network and the company divisions is abnormal in this case.

Comparison with Community Based Association Method Table II shows the top five outlier cliques obtained as a result of running the proposed method (*ABCOutlier*) and the community based association (*CBA*) approach for a query on the Wikipedia dataset. The results using the proposed approach are quite different from the results using the community based association approach. While the proposed approach ranks those associations higher which connect entities with rare real attribute pairs, the community based approach does the same considering only the community distributions as attributes. For lack of space, we do not explain the results in detail. Also, compared to *CBA*, our approach does not need to specify the #clusters and is not sensitive to local optima.

D. Outlier Scores and Running Time

In Section III, we proposed a shared neighbors index. Table III shows the index size and the index construction times for the various network sizes we considered. Note that even for large graphs like Wikipedia, index size is small enough for it to fit in memory. Also the index construction times are quite small.

On an average candidate filtering (Section III-B and Step 2 of Algorithm 1) reduces the size of the smallest candidate list to about 5% of its original size. For queries of sizes 2, 3, 4, 5, 6, 7, 8, 9 and 10 we observed an average of 73%, 81%, 71%, 59%, 24%, 54%, 11%, 15% and 38% reduction in matches generation time for the Wikipedia dataset.

Figure 4 shows the outlier score for all the matching cliques for 18 different queries on the Wikipedia network. Very few cliques have high outlier scores. Thus the outlier scores computed using the proposed approach are quite discriminative. Size 3 and 4 clique queries take an average execution time of 192s for the Wikipedia network.

For size 3 queries, we did not notice much gains based on the top-K heuristics. For size 4 queries, the "top-K quit" prevents an average of 15 attribute-pair computations saving some execution time. However, we believe that for large

ABCOutlier Method			*CBA* Method		
film	person	settlement	film	person	settlement
the road to el dorado	hernán cortés	seville, spain	samurai assassin	haruko sugimura	tokyo, japan
drums along the mohawk	adam helmer	german flatts, ny, us	passenger 57	wesley snipes	orlando, fl, us
what the bleep do we know!?	j. z. knight	yelm, wa, us	psycho	alfred hitchcock	london, uk
atanarjuat	zacharias kunuk	igloolik, canada	skidoo	pete the pup	richmond, ny, us
stardust	stephen fry	norwich, england	she's gotta have it	joie lee	brooklyn, ny, us

TABLE II
TOP FIVE OUTLIER CLIQUES FOR QUERY: ⟨FILM, PERSON, SETTLEMENT⟩

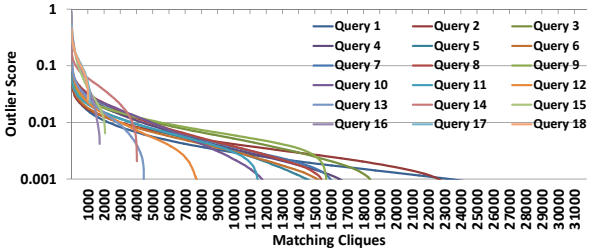


Fig. 4. Outlier Scores for Multiple Queries

#Nodes	#Edges	Index Size (MB)	Time (sec)
10K	100K	0.4	1.5
20K	200K	0.7	2.3
50K	500K	1.8	5.5
760K	4.1M	22	96.7

TABLE III
INDEX SIZES AND INDEX CONSTRUCTION TIMES

queries on larger and dense networks, the time savings could be substantially high. Also, we can re-order the attribute-pairs used for the computation of outlier score (using heuristics like estimated contribution) such that the new order can encourage early top-K quits.

VI. RELATED WORK

Outlier detection on networks has been studied for both static [2], [9] and dynamic [1], [10] scenarios. Quite different from existing work which only considers outlier detection in homogeneous networks [4], [5], [6], the proposed work aims at discovering outliers from *heterogeneous* networks. Moreover, existing outlier detection work for network data sets has focused on finding outliers for the entire network or in the context of a community. Instead of taking a general global perspective, the proposed system aims at giving the user a flexibility to find outliers following a particular schema and predicates encoded in the form of a query. Also, [4], [5] and [6] use community distributions as the only information for computing outlier scores. However, real information networks have very rich information associated with each entity and hence we exploit all such information for detecting *ABCOutliers*. Query based outlier detection on networks can be considered as a special application of graph query processing. Compared to recent work on answering graph queries on heterogeneous networks [7], [12] which provide unranked results, the proposed method attempts to rank such results based on the outlier scores.

VII. CONCLUSION

In this work, we introduced the new concept of Association-Based Clique Outliers, *ABCOutliers*. To the best of our knowledge, this is the first work on query-based outlier detection for heterogeneous information networks. Matching cliques for a user query are discovered efficiently using a novel shared neighbors index. The outlierness of a clique was computed based on the association outlierness of the attributes of the entities within the clique. Using several synthetic datasets, we showed that the proposed approach can mine *ABCOutliers* with high accuracy. We showed interesting and meaningful outliers detected from the heterogeneous Wikipedia network containing thousands of entities enriched by the attributes extracted from the Wikipedia Infoboxes.

VIII. ACKNOWLEDGEMENTS

The work was supported in part by the U.S. Army Research Laboratory under Cooperative Agreement No. W911NF-09-2-0053 (NS-CTA) and W911NF-11-2-0086, U.S. NSF grants IIS-0905215, CNS-0931975, IIS-1017362, and U.S. Air Force Office of Scientific Research MURI award FA9550-08-1-0265. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] C. C. Aggarwal, Y. Zhao, and P. S. Yu. Outlier Detection in Graph Streams. In ICDE, pages 399–409, 2011.
[2] D. Chakrabarti. AutoPart: Parameter-free Graph Partitioning and Outlier Detection. In ECML PKDD, pages 112–124, 2004.
[3] R. Fagin, R. Kumar, and D. Sivakumar. Comparing Top K Lists. In SODA, pages 28–36, 2003.
[4] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han. On Community Outliers and their Efficient Detection in Information Networks. In KDD, pages 813–822, 2010.
[5] M. Gupta, J. Gao, Y. Sun, and J. Han. Community Trend Outlier Detection using Soft Temporal Pattern Mining. In ECML PKDD, pages 692–708, 2012.
[6] 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.
[7] H. He and A. K. Singh. Graphs-at-a-time: Query Language and Access Methods for Graph Databases. In SIGMOD, pages 405–418, 2008.
[8] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal of Scientific Computing (SISC)*, 20(1):359–392, Dec 1998.
[9] C. C. Noble and D. J. Cook. Graph-Based Anomaly Detection. In KDD, pages 631–636. ACM, 2003.
[10] B. Pincombe. Anomaly Detection in Time Series of Graphs using ARMA Processes. *ASOR Bulletin*, 24(4):2–10, 2005.
[11] A. Savasere, E. Omiecinski, and S. B. Navathe. Mining for Strong Negative Associations in a Large Database of Customer Transactions. In ICDE, pages 494–502, 1998.
[12] P. Zhao and J. Han. On Graph Query Optimization in Large Networks. PVLDB, 3(1):340–351, 2010.