Big Graph Privacy

Hessam Zakerzadeh
University of Calgary
hzakerza@ucalgary.ca

Charu C. Aggarwal
IBM T.J. Watson Research Center
charu@us.ibm.com

Ken Barker
University of Calgary
kbarker@ucalgary.ca

ABSTRACT

Massive graphs have become pervasive in a wide variety of data domains. However, they are generally more difficult to anonymize because the structural information buried in graph can be leveraged by an attacker to breach sensitive attributes. Furthermore, the increasing sizes of graph data sets present a major challenge to anonynization algorithms. In this paper, we will address the problem of privacy-preserving data mining of massive graph-data sets. We design a *MapReduce* framework to address the problem of attribute disclosure in massive graphs. We leverage the *MapReduce* framework to create a scalable algorithm that can be used for very large graphs. Unlike existing literature in graph privacy, our proposed algorithm focuses on the sensitive content at the nodes rather than on the structure. This is because content-centric perturbation at the nodes is a more effective way to prevent attribute disclosure rather than structural reorganization. One advantage of the approach is that structural queries can be accurately answered on the anonymized graph. We present experimental results illustrating the effectiveness of our method.

1. INTRODUCTION

Network data has become increasingly important in recent years because of the greater importance of various application domains such as the Web, social networks, biological networks, and communication networks. The semantics and the interpretation of the nodes and the links may vary significantly with application domain, *e.g.* in a social network nodes can represent individuals and their connections capture friendship, while in a gene regulatory network nodes are genes and connections refer to their interactions. These graph data carries valuable information and are analyzed in various ways to extract new knowledge. For example, social networks provide significant insight about psychological behavior of individuals or gene regulatory networks are widely studied to elucidate mechanism of diseases.

A major problem with the release of various social networks is that the nodes are often associated with sensitive information about the individual, such as their posts, tweets, interests, hobbies or their political opinions. Individuals might be willing to share such information with their friends, close circles or a particular community but not necessarily with the broader public. An example is the social network published by [18] which captures the sexual contacts, shared drug injections and *HIV* status of individuals. In such cases, a straightforward elimination of only the *directly* identifying information, such as the name or the Social Security Number (SSN) is usually used. However, such an approach, referred to as naive anonymization, is generally not successful in protecting privacy for graph data[5, 12], as the case for multidimensional data.

The structural pieces of information (*e.g.* friendship links) in a social network are usually either far less sensitive than the personal information of a user or are publicly available. An example is a co-authorship network in which links are publicly available, however each author may consider details of his/her ongoing research sensitive. On the other hand, for a social network release to be truly useful, such content-centric information needs to be released along with the social network structure.

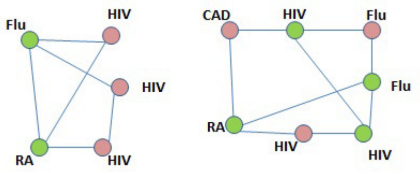
In the context of graphs, a major challenge is that the structural information embedded in graphs such as the node degrees are often highly revealing information. Such information can be leveraged by an adversary to launch privacy attacks. In general, attacks on graph data are categorized as either *active* or *passive* [5]. In active attacks, an adversary has the ability to influence the structure of the graph, such as the social network. In such cases, the adversary may construct a highly distinguishable pattern (subgraph), and establish carefully chosen connections with target victim nodes. The adversary can then leverage their actively created subgraph to efficiently determine the identity of the targeted nodes in the released network, even when it is anonymized. In *passive* attacks, the adversary does not have the ability to influence the structure of the graph. The released graph is only "passively" available to the adversary. The adversary may then use all the publicly available background information to learn more about sensitive information belonging to the individual. This can result in privacy violation. Passive attacks are more realistic because attackers can usually access the network modification platform in only a local or limited way, and consequently are not able to significantly alter the overall graph structure.

The privacy of individuals (nodes) in graphs can be breached in three different ways. The first is referred to as *identity disclosure*. This refers to the fact that the identity of individuals associated with graph nodes are disclosed. The second is that of *link disclosure*. In this case, relationships between individuals are considered sensitive and an adversary aims at disclosing the relationship between victims. The identity and link disclosure attacks are generally related because the disclosure of a sensitive link requires the identification of the node identities as an intermediate step. Both forms of privacy are fully related to structural anonymization, and do not even assume that content is released along with the graph. The final setting is that of *attribute disclosure*. In this case, an attacker aims at finding out sensitive *content attribute(s)* associated with a victim. In this setting, content attributes are always released along with graph nodes, and a user may not wish to disclose this information. The structural information about the graph increases the ease of making content-centric attacks in this case. Even though this kind of disclosure poses a more significant problem, it remains almost completely unaddressed in the literature. This is the main focus of this paper.

Many privacy models[14, 24, 26, 23, 8] have recently been designed for private release of network data. The models make different assumptions about the background information of the adversary. For example, Liu and Terzi[14] proposed a model, namely k-degree, to prevent *identity disclosure* against an attacker who has knowledge about the *victim node's degree*. The k-neighborhood model[24] prevents *identity disclosure* against an attacker enriched with knowledge about the *immediate neighbors of a target victim*. Algorithms to enforce these privacy models typically suffer from two problems. First, they are expensive and particularly difficult to implement for very large graphs. This is important, because the sizes of graphs have continued to increase significantly in recent years. Secondly, all recent methods focus almost exclusively on *structural* anonymization, with little or no focus on the interplay between content and structure in the context of sensitive *attribute* disclosure of nodes. The latter is usually much more sensitive. Individuals do not wish their views, controversial opinions, and proclivities to be released in the open without their knowledge. Structural anonymization dramatically degrades the graph structure, and provides little *attribute-wise* protection in return for this large loss in utility. It is important to point out that different graph domains have different levels of relative sensitivity of links and attributes. For example, in some domains, the sensitivity of attribute disclosure is much greater than link disclosure. Generally, graph data is sufficiently complex that it is impractical to prevent all forms of disclosure with a single anonymization approach. In such cases, it makes little sense to perturb the structure of the graph. Rather, the use of traditional attribute-centric modification is sufficient as long as the graph structure is taken into account during attribute perturbation. Taking the structure into account during attribute perturbation ensures that important real-world properties of graphs, such as homophily and content-structure locality, are preserved by the anonymization process.

In this work, we devise the first approach to cope with these content-centric issues in the context of very large graphs. Our approach is a simple, yet efficient algorithm to prevent attribute disclosure attacks in the passive scenario, while publishing the full graph structure. Our anonymization technique is best suited to the big-graph scenarios where there are millions or billions of nodes and edges. The existing works which prevent different types of disclosure either fail or suffer from long running time when applied on such graphs. Hence our proposed algorithm, which is a *MapReduce* algorithm, is the first attempt to address the privacy of *big graphs*.

Figure 1: Example of two published graphs. Nodes sensitive attributes are shown by each nodes. The graph in (a) does not provide any protection against attribute disclosure attack, while the graph in (b) satisfies 2-diversity.



(a) *Vulnerable graph* (b) *Satisfying 2-diversity*

1.1 The Problem

An attacker may obtain different kinds of structural background knowledge about victim nodes. Hay *et al.* [12] systematically captures three prominent types of background knowledge: *vertex* knowledge, *subgraph* knowledge and *hub fingerprint* knowledge. Vertex knowledge refers to node degrees, while subgraph and hub fingerprint knowledge describe (partial) subgraphs around nodes and their distances from hub nodes, respectively. However, some types of background knowledge are more difficult to acquire so the precise kind of knowledge available depends on the problem setting. Some existing proposals [24, 26] consider very powerful adversaries with subgraph or hub fingerprint knowledge. Our work currently considers adversaries with vertex knowledge (*i.e.* victim nodes degrees) in the context of *big graphs*, because this represents a large class of potential attackers and a readily available form of background knowledge. We anticipate considering more sophisticated attack models in future work. We assume that node-degree information is known to the attacker and their goal is to determine *sensitive attribute(s)* values associated with victim node(s). In other words, the adversary is undertaking an *attribute disclosure attack* with node-degree information. Figure 1 illustrates an anonymized graph in which the *identity* of a person is anonymized with respect to the degree. However, an attacker, supplied with nodes degrees, can easily conclude that individuals with node-degree value of 2 are suffering from *HIV*.

To prevent this attack, we leverage the privacy models in relational data and mandate that the values of sensitive attributes of nodes with the same degree should be well-represented. Although our proposed *MapReduce* algorithm is capable of adopting all types of diversity (*e.g.*, recursive ℓ-diversity, t-closeness), depending on different requirements on sensitive value distribution, we only consider distinct ℓ-diversity in this work because of its simplicity and fundamental nature. Therefore, we enforce the relative frequency of each sensitive value for the set of nodes with the same degree to be at most $\frac{1}{\ell}$.

The ℓ-diversity concepts in the relational setting can be generalized to graphs as follows. Each record corresponds to an individual in the relational model, while nodes represent individuals in the graph data. The quasi-identifiers are a subset of attributes in relational data and records having the same value for their quasi-identifiers form an equivalence class. On the other hand, the node-degree is the quasi-identifier and nodes with the same degree form an equivalence class in graph data. Figure 1b illustrates a published graph satisfying 2-diversity in which degree-based diversification of sensitive attributes assures a confidence of at least $\frac{1}{2}$.

We consider the problem of attribute disclosure attack in the context of big graph data with the use of the ℓ-diversity model. Preserving the privacy of such a graph can be enabled with the use of distributed frameworks such as *MapReduce*. Therefore, we address the following question in this work:
"How to prevent attribute disclosure attacks using the ℓ-diversity privacy model in big graphs?".

2. RELATED WORK

The problem of privacy preservation was first studied in [3]. This approach was based on noise-based perturbation of the underlying data. Subsequently and starting with Samarati's seminal work[19], a significant amount of research has been done on the problem of privacy preservation and numerous privacy-preserving models (*e.g.* k-anonymity[19], ℓ-diversity[15], t-closeness[13], δ-presence [16], or differential privacy[9, 10]) have been proposed to protect published data against attackers. Each privacy model can prevent particular types of attacks and makes some assumptions about the attacker's background knowledge. Besides, each type of data, *e.g.* relational data, streaming data or graph data, poses its own unique requirements and challenges and mandates a different privacy model.

Backstorm *et al.* [5] were the first to point out that simply removing identifier information from nodes in a graph does not prevent identity disclosure. The re-identification can occur through structural information that an attacker can obtain through various sources. Hay *et al.* [12] systematically modeled major variants of adversary structural background information.

Starting with the work of [5], graph privacy received increasing attention. The models generally aim to prevent three categories of attacks: *identity disclosure attack*, *link disclosure attack*, and *attribute disclosure attack*. The identity disclosure attack has been studied in works such as [14, 24, 26]. Liu and Terzi[14] considered an adversary armed with node-degree information and proposed an edge addition/deletion technique for anonymization. The solution alters an input graph such that for every node v, there exist at least $k-1$ other nodes with the same degree as v in the published graph. Zhou and Pei[24] assumed that the attacker is aware of subgraph constructed by immediate neighbors (1-neighborhood subgraph) of a victim node. They proposed an algorithm which organizes nodes into groups, according to their neighborhoods, and then anonymizes the neighborhoods of vertices in the same group. In [26], the authors adopted a more general assumption and investigated a scenario where the attacker knows any subgraph around a victim node (d-neighborhood subgraph). The authors then aimed to construct a graph in which there exist at least k isomorphic subgraphs, in the published graph, to each given subgraph in the input graph.

Other works such as [23, 20, 21] focused on the link disclosure attacks. Zheleva and Getoor[23] devised and investigated the effectiveness of several strategies to hide sensitive links within a graph. The work in [20] investigated the interplay between addition/deletion and edge switch techniques and the graph spectrum. The authors have also proposed a spectrum preserving algorithm to prevent link disclosure attacks. Potential disclosure of sensitive links in graph generation algorithms was also examined in [21].

The attribute disclosure attack is explored in a few works[25, 22]. The authors in [25] extended the k-neighborhood privacy such that the ℓ-diversity is preserved over the sensitive attribute of nodes within each group. Yuan *et al.*[22] considered three gradually increasing levels of attacker's background information. Their proposed solution alters the graph structure by adding fake edges and nodes along with generalizing the sensitive values to provide personalized protection. In addition, the graph nodes and their sensitive attributes or a rich social network, in general, can be modeled as a bipartite graph and be anonymized using the techniques proposed by Cormode *et al.*[7]. However, all the proposed works manipulate the graph structure to provide a certain level of privacy. Structural modification dramatically affects the utility that a graph structure provides. In this paper, we put forward an algorithm to prevent the attribute disclosure attack without manipulating the graph structure.

3. PROBLEM DEFINITION

Let the quadruplet $G(V, E, S, f)$ be a simple graph in which V represents a set of nodes (vertices), E is the set of edges, S is the set of sensitive values and $f : V \to S$ is a mapping function that relates each node to its sensitive value. Table 1 summarizes the list of commonly-used notations in this work. Next we define the notion of ℓ-diversity for a set of nodes.

Table 1: List of notations

notation	explanation
v_i	$i^{th} vertex$
$f(v_i)$	function returning the sensitive value of v_i
$deg(v_i)$	degree of vertex v_i
eq^d	set of nodes with degree d (an equivalence class)
N_i	set of immediate neighbors of v_i
S^X	set of sensitive values of set of nodes X
$\|.\|$	size of a set

DEFINITION 1 (ℓ-DIVERSITY PRIVACY CONDITION). *A subset of nodes $V_j \subset V$ in graph $G(V, E, S, f)$ satisfies the ℓ-diversity privacy condition if and only if $\forall v_i \in V_j$ and multiset of sensitive values (S^{V_j}, g) where $S^{V_j} = \{f(v_i)|v_i \in V_j\}$ and $g(\cdot)$ returns the frequency of each sensitive value in V_j, $\forall x \in S^{V_j}$ the inequality $\frac{g(x,V_j)}{\sum_{y \in S^{V_j}} g(y,V_j)} \leq \frac{1}{\ell}$ holds.*

Next, we generalize the notion of ℓ-diversity for a subset of nodes to the full graph G.

DEFINITION 2 (ℓ-DIVERSIFIED GRAPH). *A graph G with degree set D is called an ℓ-diversified graph if and only if $\forall d \in D$, the set of nodes eq^d with degree d satisfies the ℓ-diversity privacy condition.*

This is the generalization of the distinct ℓ-diversity model[15] to the graph data model. It is possible to define stricter conditions such as recursive or entropy ℓ-diversity. Furthermore, related notions such as t-closeness can be defined.

Our contribution in this work is the proposal of the first *MapReduce* algorithm to create ℓ-diversified big graphs, particularly social networks, so that an attacker supplied by degree information cannot succeed in launching attribute disclosure attacks. The algorithm releases the graph structure intact and in its entirety, thus structural queries can be accurately answered using the published graph. In addition, the algorithm is fully scalable and capable of anonymizing big graphs.

4. MapReduce-BASED PRIVACY ALGORITHMS

As discussed in the social networking literature[6, 11], many big graphs (*e.g.*, social networks, biological networks) are scale-free networks and their degree distribution follow a power-law distribution. This behavior is illustrated in Figure 5. As a rule of thumb, many nodes in a big social network satisfy the privacy condition for practical values of the privacy parameters (*e.g.* ℓ). Therefore, it is more reasonable to first filter the privacy-condition-satisfying nodes out and not involve them in further processing. This dramatically reduces the complexity of privacy preservation process. Algorithm 1 illustrates the steps required to enforce the privacy condition on a big graph. The output of this algorithm is the original graph in which the sensitive values of some nodes are released in group.

In this algorithm, nodes v_i are initially assigned to equivalence classes according to their degrees. In other words, $\mathbb{EQ} = \{eq^d | d \in D\}$ and $eq^d = \{v_i \in V | deg(v_i) = d\}$ where D is the graph degree set. The privacy condition is then checked for each equivalence class eq in $\mathbb{EQ}$, and nodes in equivalence classes not satisfying the ℓ-diversity condition are appended to a list of violating nodes. These privacy-violating nodes are then clustered such that the nodes within each cluster satisfy the ℓ-diversity condition. To cluster nodes, we use an agglomerative clustering in which two clusters are merged in each iteration. However, the merging process must be designed for satisfying the privacy condition. A suitable

Algorithm 1 Big Graph Anonymization Steps

```
1: $AnonymizationScheme(G)$
2:   //$G$ is a simple graph of form ($V$,$E$,$S$,$f$)
3:   $\mathbb{EQ}$= assign nodes with degree $d$ to equivalence class $eq^d$
     and form equivalence classes set
4:   **foreach** ($eq$ in $\mathbb{EQ}$)
5:     **if** ($eq$ does not satisfy the *ℓ-diversity condition)*
       append nodes $v_i \in eq$ to the violating nodes set $VN$
6:   $\mathbb{C}$=cluster nodes $v_j \in VN$ such that each cluster $c \in \mathbb{C}$
     satisfies the *ℓ-diversity condition*
7:   define function $f\prime : V \to S \times \mathbb{N}$ such that
8:     **foreach** ($v_i$ in $V$)
9:       **if** ($v_i$ not in $VN$)
10:        $f\prime(v_i) = (f(v_i), 1)$
11:      **else**
12:        $f\prime(v_i)$ = the multiset of sensitive values of nodes in
           cluster $c \in \mathbb{C}|v_i \in c$
13:  publish $G\prime(V, E, S, f\prime)$
```

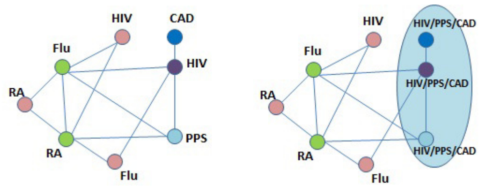
merging criterion is the entropy of sensitive values. Two clusters c_i and c_j are selected when the constituent nodes are connected by at least one edge and cause the maximum change in entropy as a result of the merging. The entropy criterion is stated as follows.

$$\underset{c_i,c_j}{\operatorname{argmax}}\, d(c_i, c_j) = H(c_i \cup c_j) - H(c_i) - H(c_j)$$

where function $H(X)$ denotes the entropy function.

The set of all formed clusters are referred to as $\mathbb{C}$. At the end, the full graph structure is published. However, the sensitive value of node v_j not originally satisfying the privacy condition is replaced by the multiset of sensitive values of nodes in cluster $c \in \mathbb{C}$ that contains v_j ($v_j \in c$). In other words, sensitive values of privacy-violating nodes are released at the cluster level instead of the node level.

Figure 2: Illustration of 2-diversification process of a graph. This process leaves nodes initially satisfying the 2-diversity condition intact and only generalizes sensitive values of 2-diversity-violating nodes.



(a) *A 2-diversity violating graph* (b) *2-divertised version of (a)*

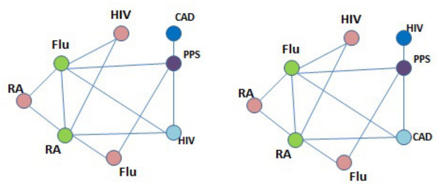
In the clustering step (line 6, Algorithm 1), satisfying the ℓ-diversity condition for each cluster mostly depends on frequencies of sensitive values, which is defined as the function $g(f(v_i), c) : S \to \mathbb{N}$. This definition of $g()$ results in a *Sensitive Value Frequency aWare* (SVFW) clustering. However, $g()$ can be defined in a more relaxed way $g(f(v_i), c) : S \to 1$, which basically ignores the frequency of sensitive values within each cluster and turns the process into a *Sensitive Value Frequency aGnostic* (SVFG) clustering. As soon as the number of distinct sensitive values reaches ℓ within a cluster, further merging is no longer performed on that cluster in SVFG, while it may not be case in SVFW. In fact, the SVFG is a relaxed version of SVFW, and apparently an ℓ-diversity-satisfying cluster under SVFW is an ℓ-diversity satisfying cluster under SVFG too, whereas the vice versa may not hold. It is easy to show that SVFG satisfies the ℓ-diversity condition as each node within a given cluster can be related to any of the sensitive values.

Pathological cases might exist in both SVFW and SVFG in which the clustering ends up with a cluster not satisfying the privacy condition. Under such circumstances, sensitive values of nodes belonging to the cluster must be suppressed. Figure 2a demonstrates a graph in which nodes with degrees one, three and four do not satisfy 2-diversity. Applying the anonymization steps in Algorithm 1 results in the 2-diversity-satsifying graph shown in Figure 2b. In this anonymized graph both SVFW and SVFG result in the same anonymized graph as each sensitive value occurs only once in the formed cluster.

A data user, after obtaining the published anonymized graph must instantiate a graph from it. Instantiation means randomly assigning a sensitive value to those nodes whose sensitive values are released as a multiset. A query can be more accurately answered by averaging over multiple instantiations. Figure 3 exemplifies two possible instantiated graphs from the 2-diversity satisfying graph shown in Figure 2b.

Section 4.2 describes how Algorithm 1 can be converted into *MapReduce* jobs to satisfy the ℓ-diversity condition on big graphs. However, before proceeding further, we provide a very brief introduction to the *MapReduce* framework.

Figure 3: Two possible instantiations of 2-diversified graph shown in Fig. 2b



(a) *Instantiation 1* (b) *Instantiation 2*

4.1 MapReduce

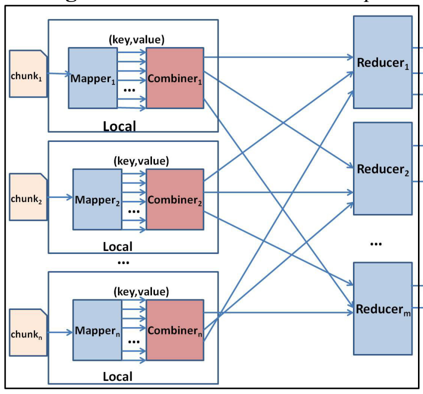
MapReduce is a programming framework proposed by *Google* to enable efficient data processing. As implied by its name, the approach uses distributed *Map* step, followed by a *Reduce* step. These steps must be designed by the application programmer. The *MapReduce* framework splits the data into equal-size chunks, and feeds each chunk to a separate mapper. Each mapper processes its own chunk, and outputs $(key, value)$ pairs. Pairs with the same key are transferred to one reducer by the framework. The set of all reducer outputs are used to construct the final result. An arbitrary function *combiner* can also be defined to reduce the amount of data transfer between mappers and reducers by aggregating the $value$s belonging to each key. There is also a user (driver) program that executes/runs the *MapReduce* program. Figure 4 illustrates the data flow of a *MapReduce* job. We do not define the combiner function in our *MapReduce* jobs as its effect in reducing the data transfer is not significant, considering the mapper output.

4.2 Privacy Algorithm Transformation to MapReduce Jobs

Algorithm 1 consists of two phases, which are referred to as the *pruning phase* and the *clustering phase*, respectively. The pruning phase corresponds to lines 1-5 and the clustering phase to the line 6 in the algorithm. Lines 7-13 involve publishing the resulting anonymized graph, and can also be carried out by a separate job, as will be explained shortly. Here, we demonstrate how each phase can be converted into *MapReduce* jobs.

As a *de facto* standard, we assume edges and attribute informa-

Figure 4: The data flow of a *MapReduce* job



tion of a graph are stored in two separate files: the *relationship* and the *meta-info* file. Each line in the relationship file shows an edge and each line in the meta-info file corresponds to one node and contains the values of different attributes of that node. Minor modification is required for other sorts of graph representations. Besides, for the sake of simplicity, we assume that each node contains only one sensitive attribute, however extension to multi sensitive attributes is straightforward.

4.2.1 Pruning Phase

The pruning phase comprises two *MapReduce* jobs. The first job discovers the immediate neighbors of each node and the second one does the actual filtering (pruning) task based on whether an equivalence class satisfies the ℓ-diversity condition or not.

MapReduce Job for Neighborhood Discovery

This job discovers neighbors of each node. The input to this job is both the relationship and the meta-info files. The mapper and the reducer of this job are as follows:

Mapper: As there are two different input files, mappers are differentiated according to the data chunk they are fed with and their outputs differ depending on whether a mapper gets a chunk of the relationship or meta-info file. If the chunk is coming from the relationship file, then for each record "v_i,v_j" (corresponding to an edge), the mapper outputs two pairs <v_i,v_j> and <v_j,v_i>. However, providing that the mapper is fed by a chunk of meta-info file, it outputs the vertex as the key and its sensitive value as the value.

Algorithm 2 Neighborhood Discovery Job

```
1: Mapper(k,v)
2:    // $v$ can be either of form ($v_i$,$v_j$) or ($v_i$,$f(v_i)$)
3:    if (input chunk belongs to the relationship file)
4:       emit($v_i$,$v_j$)
5:       emit($v_j$,$v_i$)
6:    else
7:       emit($v_i$,$f(v_i)$)
```

Reducer: All neighbors of a given vertex (let's say v_i) as well as its sensitive value ($f(v_i)$) are brought together in a reducer. Thus, the reducer receives pair <v_i, $N_i + f(v_i)$> where N_i is the set of all v_i's immediate neighbors. The '+' sign is a simple string concatenation operator. Each reducer then emits the pair <$v_i/f(v_i)$, N_i>. The output file of this job is fed into the second *MapReduce* job as input.

MapReduce Job for Filtering

This *MapReduce* job filters out nodes originally satisfying the ℓ-diversity condition and leaves us with only violating nodes. As discussed earlier, it can be modified for other types of ℓ-diversity or even t-closeness.

Algorithm 3 Neighborhood Discovery Job

```
1: Reducer(k,V)
2:    // $k$ is a vertex and $V$ contains all neighbours and the sensitive
      value $f(k)$
3:    emit($k/f(k)$,$V$)
```

Mapper: Each mapper reads a data chunk which has been output from the reducer in the first job. The value of each input record in the chunk is of form $(v_i/f(v_i),N_i)$[1]. The mapper then emits a pair where the key is the degree of v_i, $|N_i|$, and the value is <$v_i/f(v_i)$, N_i>. N_i will be used in the clustering phase later.

Algorithm 4 Filtering Job

```
1: Mapper(k,v)
2:    //$v$ is of form $(v_i/f(v_i), N_i)$
3:    emit($|N_i|$,<$v_i/f(v_i), N_i$>)
```

Reducer: All vertices with the same degree (eq^d) as well as their sensitive values S^d are transferred to the same reducer. The reducer can then simply decide on whether the equivalence class eq^d satisfies the ℓ-diversity or not. If so, all vertices in eq^d are ruled out, otherwise they are output. Therefore, each line in the the reducers' output file contains a privacy-violating node and its sensitive value. The output file(s) will be used as input for the next *MapReduce* job in the clustering phase.

Algorithm 5 Filtering Job

```
1: Reducer(k,V)
2:    //$V$ is a list of $(v_i/f(v_i), N_i)$ where $deg(v_i) = d$
3:    $violation$=false
4:    foreach $(v_i/f(v_i), N_i) \in V$
5:       if ($\frac{freq(f(v_i),S^d)}{|S^d|} > \frac{1}{\ell}$)
6:          $violation$=true
7:          break
8:    if ($violation$)
9:       foreach $(v_i/f(v_i), N_i) \in V$
10:         emit($v_i/f(v_i), N_i$)
```

4.2.2 Clustering Phase

The clustering phase has only one *MapReduce* job. This job groups the privacy violating vertices and forms clusters in which the sensitive values are well-represented. An agglomerative hierarchical clustering algorithm can be used for the clustering. However, merging two clusters must enable ℓ-diversity satisfaction. To achieve this goal, we consider the similarity measure between two neighbor clusters c_i and c_j to be the difference in entropy of sensitive values in the resulting cluster and the original clusters, *i.e.* $H(c_i \cup c_j) - H(c_i) - H(c_j)$[2]. A cluster is removed from further merging as soon as it satisfies the ℓ-diversity condition. The sensitive values of vertices in the remaining cluster may need to be suppressed when they do not satisfy the ℓ-diversity condition.

MapReduce Job for Clustering

The clustering job is in charge of clustering vertices according to sensitive values entropy.

Mapper: Each mapper is fed a chunk consisting of privacy violating vertices (output by the $Filtering$ job). It buffers all the input nodes and clusters them according to entropy increase criterion.

[1]Note that by default the key for each record in the mapper is the record's offset in the input file.

[2]If c_i and c_j are not connected, their similarity will be zero. This can be determined using N_i.

Note that the clustering is influenced by the number of mappers and size of the chunks as violating nodes are split among different mappers. That means a better clustering, in terms of fewer suppressed sensitive values, is expected with fewer mappers or larger chunk size. One mapper operating over all the violating nodes is the ideal case, however it may become a bottleneck in case of having large number of violating nodes.

The output of the *Clustering MapReduce* job is referred to as the Generalized Sensitive Value (GSV) file(s). Each line in the GSV file contains a node name along with the multiset of the cluster sensitive values to which the node belongs. Algorithm 6 shows the pseudocode of this job. Function $multiset(c)$ takes a cluster and returns the multiset of its constituent node sensitive values.

Algorithm 6 Clustering Job

```
1: Mapper(k,v)
2:   // $v$ is of form $(v_i/f(v_i), N_i)$
3:   append each pair $(v_i/f(v_i), N_i)$ to $buffer$
4:   if (no more pair)
5:     $\mathbb{C}$ = cluster($buffer$) //either SVFW or SVFG clustering
6:   for ($c$ in $\mathbb{C}$)
7:     for ($v_j$ in $c$)
8:       emit($v_j$, $multiset(c)$)
```

There is no reducer required for this *MapReduce* job and the mapper's output should be considered as the job's output. To publish the final anonymized graph (lines 7-13, Algorithm 1), the relationship file must be released as original, nonetheless the sensitive values of originally-privacy-violating nodes in the meta-info file must be swapped with their corresponding values in the GSV file. There exist two alternatives to carry out the swap:

1. A *MapReduce* job which caches GSV file and takes the meta-file as input. It then goes through the meta-info file and simply does the swaps. This alternative is suitable when the GSV file is of small size.
2. A *MapReduce* job which joins the meta-info and the GSV files and changes the sensitive values of violating nodes in the meta-info with the corresponding value from the GSV file. This approach is more appropriate for a large GSV file.

The new meta-info file must also be released at the end.

5. DATA TRANSFER ANALYSIS

As shown in Figure 4, each *MapReduce* job involves two data transfers. The first involves data transfer between mappers and combiners. The second involves data transfer between combiners and reducers. Since each mapper and its corresponding combiner run on one node, the first data transfer is local. However, the second data transfer may occur across the network and become a bottleneck. As we have not specified a combiner in this work, we only analyze the amount of data transferred in the second case here.

First MapReduce Job, Pruning Phase

The mapper in the first job doubles the size of the relationships file because it outputs two pairs per input record of the relationships file, but leaves the size of meta-info file unchanged. $|E|$ and $|V|$ show the number of input records from the relationships and the meta-info files, respectively and let b denote the number of bytes required to store a node name (or a sensitive value). So, the mapper's input and output data are of size $2b.|E| + 2b.|V|$ and $4b.|E| + 2b.|V|$.

The reducer then outputs one record per node in which there is the node name, its sensitive value and the list of node's neighbors. The average size of each record is $(2 + \mu).b$ where μ is the average node degree in the input graph. As shown in the social networks literature [4, 6], degree distribution in many social networks follows a power law probability distribution $p(x) \sim x^{-\lambda}$ where $2 < \lambda < 3$. Newman[17] also proved that the average degree in a graph following power law distribution is $\mu = \frac{\lambda-1}{\lambda-2}$ (given $\lambda > 2$). Thus, the reducers' output will be of average size $(2 + \frac{\lambda-1}{\lambda-2}).b.|V|$. In summary, the asymptotical data transfers are:

mapper's input:
$$O(|E| + |V|)$$

mapper's output (reducer's input):
$$O(|E| + |V|)$$

reducer's output:
$$O(\frac{\lambda-1}{\lambda-2}.|V|)$$

Hereafter we only consider the asymptotical analysis of the data transfer.

Second MapReduce Job, Pruning Phase

The second *MapReduce* job is fed by the output of the first job which is of size $O(\frac{\lambda-1}{\lambda-2}.b.|V|)$. It then outputs one pair for each input record in which the key is the node's degree and the value consists of node's name, its sensitive value and neighbors. Thus, the mapper's output has the same magnitude of the input which is

$$O(\frac{\lambda-1}{\lambda-2}.|V|)$$

Afterwards, the reducer prunes out nodes originally satisfying the privacy condition and outputs privacy-condition-violating nodes as well as their sensitive values. Number of output nodes is highly dependant to the sensitive values distribution within different equivalence classes and cannot be easily estimated. However, considering the power law for degree distribution, we can estimate the lower bound of output nodes. The number of nodes for a given degree (let's say x) as follows:

$$\frac{n}{|V|} \sim x^{-\lambda} \rightarrow n \sim |V|.x^{-\lambda}$$

Provided that the number of nodes for a given degree (x) is less than the privacy level (ℓ), nodes with degree x are output by the reducer[3]. To find an estimated lower bound for x we must have:

$$|V|.x^{-\lambda} \lesssim \ell \rightarrow x \gtrsim \sqrt[\lambda]{\frac{|V|}{\ell}}$$

so the estimated size of reducer's output is

$$\Omega(\frac{\lambda-1}{\lambda-2}.|V|.\sum_{x=\sqrt[\lambda]{\frac{|V|}{\ell}}} x^{-\lambda})$$

Third MapReduce Job, Clustering Phase

The input to this *MapReduce* job is the output of the previous job and has the same magnitude. For each input record, which is formed of a node name along with its sensitive value, one pair consisting of the node name and its multiset of sensitive values is output. Therefore, the output size is

$$O(|V|.\sum_{x=\sqrt[\lambda]{\frac{|V|}{\ell}}} x^{-\lambda})$$

6. EXPERIMENTAL RESULTS

In this section, we will study the effectiveness, efficiency, and running time characteristics of the anonymization algorithm. The

[3]Although existence of at least ℓ nodes with degree x is not enough for ℓ-diversity condition to satisfy, it is a necessary condition.

experiments were carried out on Hadoop 1.0.4[4], an open source implementation of the *MapReduce* framework, on the *Hadoop* clusters running on *ACENet*[5]. It has 32 nodes connected through a Gigabit Ethernet connection, each having 16 cores and 64 GB of RAM running Red Hat Enterprise Linux 4.8. Table 2 lists the Hadoop parameters in our experiments[6].

Recall Section 2 stated there is no existing work that prevents attribute disclosure attacks with the fairly limited restrictions associated with our assumptions regarding attacker's background knowledge. For example, Zhou and Pei [25] consider an attacker with 1-neighborhood background knowledge and Yuan *et al.* [22] assumes knowledge about node labels is also available to an attacker. Unfortunately, divergent assumptions about the knowledge held by attackers makes the comparison with our solution impossible. We also note that our goal is to address big graphs while the solutions proposed by other, due to the increased complexity associated with protecting against sophisticated attacks, means these alternative approaches have only been shown to work on comparatively small graphs.

Our anonymization algorithm does not change the graph structure, so all graph properties remain intact and structural queries can be answered accurately using the anonymized graph. However, the sensitive values for some nodes are released as a multiset and in cluster level. This ambiguity introduces some error in answering queries involving the sensitive values, however can be ameliorated by averaging over multiple instantiations. To measure the effect of sensitive values ambiguity, we measure the change of certain queries' results between the original and the anonymized graphs. We considered three types of queries:

- **Pair query** (one hop query): this type of query involves pairs of nodes which are connected. It demonstrates how many nodes from one subpopulation have relation with nodes from another subpopulation in the graph. An example query in this type is "how many users of 19 years old are friends with users of 28 years old?".
- **Trio Query** (two hop query): these queries involve three connected nodes in the graph. it in fact counts triples that satisfy a given condition. A sample query of this type is "how many users of 19 years old are friends with users of 28 years old who are friends with users of 50 years old?"
- **Triangle queries**: this sort of query counts the number of triangles (cliques of size three) that holds a query condition. For instance, "how many users of 19 years old are friends with users of 28 and 70 years old who are friends with each other?".

The utility loss is measured by the average relative query error for each type of query over multiple instantiations. For an anonymized graph and set of n queries $\{q_1, q_2, ..., q_n\}$, the average relative error is calculated as $\frac{1}{n}\sum_{i=1}^{n}\frac{|q_i(a)-q_i(o)|}{q_i(o)}$ where $q_i(a)$ and $q_i(o)$ denote the results of running query q_i on the anonymized and the original graphs, respectively.

The anonymization process only affects a (small) portion of nodes in the big graph. That is, the result of executing these three types of queries on a great portion of the anonymized graph is exactly the same as the original graph. Therefore, to better show the impact of anonymization on each type of query, we extract subgraph which results in different answer for a given type of query between the original and the anonymized graphs and only report the error of executing queries on the subgraph. It is worth noting that the total error (the error on the entire graph) will be much smaller than the reported one because the total error must be calculated on the whole graph in which most nodes remained intact. This subgraph for the pair and triangle queries is the subgraph formed by the initially-privacy-violating nodes and their immediate neighbors (neighbors with distance one). For trio queries, this subgraph is constructed by the initially-privacy-violating nodes and as well as nodes within distance one and two from them. Besides, the running time was measured in terms of the wall-clock time (milliseconds). This provides a good indicator of the overall scalability of the method.

[4]`http://www.hadoop.com`

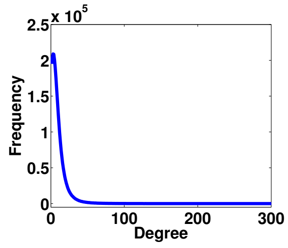
[5]`http://www.ace-net.ca`

[6]The rest of the parameters have the default value.

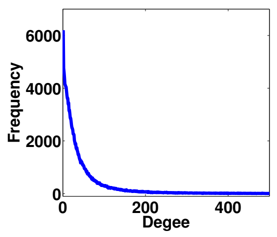
Table 2: Parameters of Hadoop

Parameter Name	Value
$fs.block.size$	64MB
$io.sort.mb$	1024MB
$io.sort.factor$	50
$dfs.replication$	3

Figure 5: Degree distribution in the $Patent$ and $LastFM$ data sets



(a) *Patent data set*



(b) *LastFM data set*

6.1 Data Sets

We used two real big graphs described below:

- **US Patent Citation Graph**: The citation graph includes all citations made by patents granted between 1975 and 1999 and is maintained by National Bureau of Economic Research[2]. This data set contains over 2.9 million nodes and 16.5 million edges. Although it is a directed graph, we consider it as undirected since there is no pair of nodes citing each other. The meta-file also contains the patents' year which is considered as a sensitive attribute.
- **LastFM co-Group Graph**: *Last.fm* is a popular music website which recommends music to users according to their music taste. An anonymous random walk crawl of *Last.fm* is released by Networking group at UC Irvine [1]. The data set contains up to 177K users (nodes) and more than 10M friendship relations (edges) among them. Besides, the meta-file includes users' ages which we consider as the sensitive attribute.

Although these two data sets might fit in the RAM of a commodity PC, they are among the biggest graph data sets publicly available with nodes' attributes. So we utilized them to show the behavior of our proposed solution. As shown in Figure 5, the degree distribution in these graphs follow a power-law distribution. For the sake of clarity, these figures only show the tail and the central part of the degree distribution and the head part is mostly removed.

6.2 Results

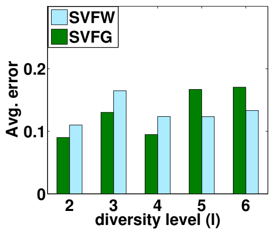
Each anonymized graph was instantiated 30 times and 50 random queries generated by uniformly sampling from the set of sensitive values. The generated queries ran on each instantiated graph and the reported error is averaged over all 1500 queries for each anonymized graph. We set the number of mappers and reducers to 30 and the running time is averaged over three runs. The number of mappers in the clustering phase have also been set to 1 for the

$Patent$ and 10 for the $LastFM$ data set. The selection is due to the number of nodes involved in the clustering. The diversity level (ℓ) also ranges from 2 to 6 in the experiments.

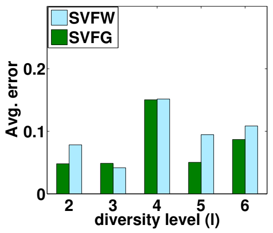
The average relative errors in answering different types of queries on the $Patent$ and $LastFM$ data sets versus the diversity level are shown in Figure 6 through 8. These figures reveal that typically the SVFG technique results in smaller error in answering different types of queries than SVFW technique. We conjecture that this phenomenon is mainly due to fine grain clusters formed in the $SVFG$ technique which can largely eliminates the randomness in the instantiation process. Besides, these figures show that no relation exists between the diversity level and average error.

As Figure 9 confirms, the running time and the number of nodes involved in the diversification process have a direct relationship. The increase in the running time is mainly due to the clustering part and can be alleviated by executing more mappers in the clustering phase.

Figure 6: Average relative error of one-hop queries vs. ℓ

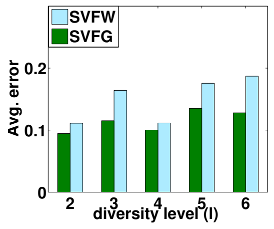


(a) *Patent data set*

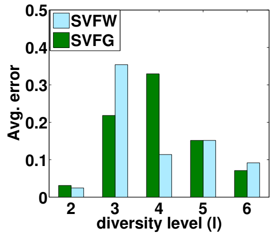


(b) *LastFM data set*

Figure 7: Average relative error of two-hop queries vs. ℓ

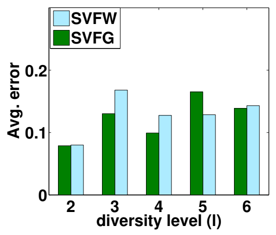


(a) *Patent data set*

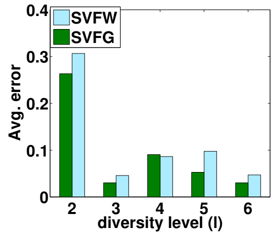


(b) *LastFM data set*

Figure 8: Average relative error of triangle queries vs. ℓ



(a) *Patent data set*

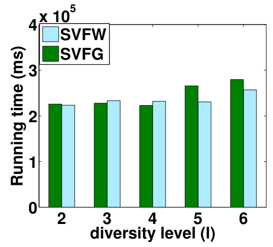


(b) *LastFM data set*

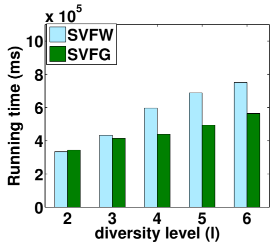
7. CONCLUSION

Even though publishing social networks between individuals or entities provides various benefits, it poses serious privacy concerns for the underlying individuals or entities. Therefore, the social network must undergo a non-trivial anonymization process before it is released. Anonymization techniques typically alter the graph structure to protect privacy, however the structure manipulation may dramatically degrade the utility of the published graph and turns to be counter intuitive. In addition, the existing anonymization solutions do not scale up and are not practically applicable on real big social network graphs. This work is the first attempt to protect the privacy of individuals in big graphs with no structure manipulation by taking advantage of *MapReduce* paradigm. This approach publishes a graph in which sensitive attributes are protected and also is capable of answering structural queries as accurate as the original graph. As our future work, we plan to consider more powerful attackers and also leverage *MapReduce* to design fully scalable anonymization techniques to protect other sorts of sensitive information within big social networks such as sensitive links.

Figure 9: Running time vs. ℓ



(a) *Patent data set*



(b) *LastFM data set*

8. REFERENCES

[1] Lastfm data set. http://odysseas.calit2.uci.edu/doku.php. Accessed: Feb 2014.

[2] Patent data set. http://www.nber.org/patents/. Accessed: Feb 2014.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. *ACM Sigmod Record*, 29(2):439–450, 2000.

[4] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[5] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, 2007.

[6] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[7] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. Class-based graph anonymization for social network data. 2009.

[8] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *VLDB*, 2008.

[9] C. Dwork. Differential privacy. In *ICALP*, 2006.

[10] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TTC*. 2006.

[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM Computer Communication Review*, 1999.

[12] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis. Resisting structural re-identification in anonymized social networks. In *VLDB*, 2008.

[13] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, 2007.

[14] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD*, 2008.

[15] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[16] M. E. Nergiz, M. Atzori, and C. Clifton. Hiding the presence of individuals from shared databases. In *SIGMOD*, 2007.

[17] M. E. Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[18] J. Potterat, L. Phillips-Plummer, S. Muth, R. Rothenberg, D. Woodhouse, T. Maldonado-Long, H. Zimmerman, and J. Muth. Risk network structure in the early epidemic phase of hiv transmission in colorado springs. *Sexually transmitted infections*, 2002.

[19] P. Samarati. Protecting respondents identities in microdata release. *IEEE TKDE*, 13(6):1010–1027, 2001.

[20] X. Ying and X. Wu. Randomizing social networks: a spectrum preserving approach. In *SDM*, 2008.

[21] X. Ying and X. Wu. Graph generation with prescribed feature constraints. In *SDM*, 2009.

[22] M. Yuan, L. Chen, and P. S. Yu. Personalized privacy protection in social networks. *VLDB*, 2010.

[23] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *Privacy, security, and trust in KDD*. 2008.

[24] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE*, 2008.

[25] B. Zhou and J. Pei. The k-anonymity and l-diversity approaches for privacy preservation in social networks against neighborhood attacks. *Knowledge and Information Systems*, 28(1):47–77, 2011.

[26] L. Zou, L. Chen, and M. T. Özsu. K-automorphism: A general framework for privacy preserving network publication. *VLDB*, 2009.