K-Automorphism: Transforming Graphs into their Anonymized Versions

Grady Denton - Shane Bennett
Florida State University
COP5725 - Peixiang Zhao

Abstract

Social networking sites have exploded in popularity over the past decade. Each new social site user brings with them a rich quantity of data about themselves and their lives. The more popular sites, such as Facebook, process hundreds of petabytes a day, resulting is huge collection of stored data. Such information is a goldmine for researchers wishing to analyze trends of the public. Unfortunately, a large portion of the collected data includes sensitive and private user data which should not be released for analysis. Our project implements a form of data anonymization which can secure user privacy while allowing for an minimally augmented version of the dataset to be published for valuable data mining endeavours

1. Introduction

Public cloud services such as Amazon cloud and Microsoft Azure have gained popularity among smaller businesses. Such services provide a desirable alternative route for companies wishing to set up an IT infrastructure without having to delve into the gritty details themselves. The ease and immediate availability of using cloud services allows startup businesses bypass the heavier initial costs of purchasing their own infrastructure hardware. However, while the benefits of outsourcing infrastructure are obvious and tangible, the security risk involved in storing data in servers outside of one's control are all too often overlooked. The smaller companies to which cloud services are most desirable are especially vulnerable to this security oversight as they generally lack a security mentality.

Social networking sites, such as the hugely popular Facebook.com, use large graph structures as an intuitive way to store private user-information as well as information about the connections that exist between users. The sophisticated graph structures that result are packed full of data that is of great interest for data-mining activities. For example, news organizations are interested in the political leanings of Facebook users--they wish to collect and merge large amounts of politically-motivated user posts and classify them as positive or negative about particular politicians. In order to satisfy the demands of the data-mining community, social networking sites like Facebook must make a subset of their information publicly available--in other words, they must publish it.

The issue with publishing data lies in the need to first filter out private information which should not be made publicly available. The published data needs to meet the requirement of providing all of the necessary information for a study which may include what topics people post about, what groups they belong to, and general demographic information that can be used to group users into categories. This information may not include specific information about individuals, only information about group trends. A public attacker should not be able to identify a specific user to which data belongs, we call this the "identity disclosure" problem.

More formally, the "identity disclosure" problem occurs in any instance in which an adversary can, with a high degree of probability, locate a target entity t as a vertex v of a published social network graph G^*.

The naive solution the identity disclosure problem is to simply remove all identifiable personal information from the graph before publishing it. This entails stripping vertex label information and attributes that contain data such as names of Social Security Numbers. Graph stored data, however, is rich structural information that surrounds each and every vertex of the published data. If an adversary has some prior knowledge about what the structure information that surrounds entity *t*, they can still probabilistically determine which vertex *v* corresponds to that entity. Because other private information associated with the entity was not removed during the naive solution, all of this information is still available and is now known by the adversary to belong to the target *t*. Additionally, after structural information is leveraged to deanonymize two or more target entities, it is trivial to check *G** to determine if there is a connection between the two targets--such linking information may likewise be sensitive.

Prior solutions implement variations of the naive solution which does not protect against structural attacks. Other methods only protect against one of the four agreed upon methods of structural attacks [1]. These attacks include degree-attacks, in which an adversary knows the number of direct connections a target entity *t* has with other nodes of the network, and can locate *t* with a probability of 1/*nd*, where *nd* is the number of unique vertices with edge-degree *d*; hub-fingerprint attacks, in which an adversary knows the location of some unique hubs of the graph and knows the distance from *t* to these hubs and can triangulate the location of *t* with this knowledge; 1-neighbor-graph attacks, in which the attacker the 1-hop neighbors from *t*, and the connections between and among *t* and his neighbors(this is a special 1-hop subgraph attack); and subgraph attacks, in which an adversary knows a subgraph containing target *t*.

In particular, our paper focuses on subgraph attacks, because prior work has not offered protection against such attacks. Also, because 1-neighbor-graph attacks are a subset of subgraph attacks, providing a protection solution for the latter also covers against the former. Despite the solution's focus on subgraph attacks, it has been shown that the solution provides defence against all of the mentioned structural attacks. A definition of structural attacks is helpful in understanding the proposed solution:

First an adversary constructs a query *Q* based on known structural information about target entity *t.* If an adversary executes *Q* on the published graph *G**, and *Q* returns a limited number of matching vertices from *G**, then the adversary may be able to uniquely identify target *t* from the limited possibilities.

As can be determined from the definition, structural attacks rely on an adversary being able to receive a limited set of results to a well-crafted query. The solution proposed by our paper prevents this from ever occurring. The solution proposed allows the graph publisher to select some value *k*, and then the graph is altered in a way such that any structural based query Q will return *at least k* matches. This ensures that an adversary will never be able to identify *t* with a probability greater than 1/k, since any prior structural information the adversary relies on, will be exist in the graph in at least *k* locations.

Thus, the properties of symmetry and structural duplication of k-automorphism are taken advantage of by the proposed solution. Graph automorphism is defined here as:

A graph *G* is said to be automorphic to itself if there exists a function *f* such that for any edge e = (u, v), there exists a second matching edge (f(u), f(v)).

For a graph to meet the criteria for k-automorphism, there must be a bijective function that maps each vertex to *k - 1* other vertices such that f(v) => v_1, v_2, … v_{k-1}. And for every edge e = (u, v,), there must exist *k - 1* matching edges (f(u), f(v)). For example, if f(1) = 3 and f(2) = 4, and there exists an edge connecting vertex 1 and vertex 2, then there must likewise be an edge connecting vertices 3 and 4.

If *G** meets the criteria of k-automorphism, then for any vertex *v* of *G** there are no structural differences between *v* and the *k - 1* vertices that the automorphic function maps it to. Thus, by increasing *k* to some larger value, such as 10 or 20, the chances that an adversary can unique identify *t* are drastically reduced.

2. Related Work

2.1 False Positives in Other Symmetry Based Solutions

This section will focus on the ability of the implementation to ensure user data privacy while maintaining query response correctness. This contrasts with other implementations [1, 2, 3] which are seemingly ideal solutions to preserving privacy, yet impair the functionality of queries by being subject to false positive matches of subgraphs.

Zou, Chen, and Ozsu [1] describe their k-automorphic network model, which converts a graph into a k-automorphic network version to be stored. This protects against identity disclosure and guarantee the protection of privacy against structural attacks. The process of converting a given graph into a k-automorphic graph involves introducing "noise" edges to a graph of nodes and real edges. The idea is that these noise edges introduce a symmetry to the graph which prevents structure based attacks. When this method is used to protect social networks, attackers are no longer able to identify a target through use of structural queries. This process of transforming the graphs into k-automorphic versions is the basis for the paper we are implementing.

However, the implementation presented in [1], the basis for our chosen paper's implementation, introduces false positive query returns due to the very same noise edges that were created as part of the k-automorphic transformation. Our chosen paper circumvents this problem by splitting the query between client and server. Based on the assumption that the query issuer is the owner of the original data, our implementation will simply allow the server to return false positives which the client will use an efficient hash index to filter out. Therefore, the brunt of the query (i.e. matching the subgraph over the very large full graph) is placed on the broad shoulders of the server. But the client can filter the false positives in the much less intensive second step by comparing to the original, non-automorphic graph (or in the case the query issuer is not the original data owner, they can issue said simple query to the owner).

Wu, Xiao, W. Wang, He, and Z.Wang [2] took a similar approach. Instead of using k-automorphic transformations of graphs, they used k-symmetric. Cheng, Fu, and Lie [3] also researched privacy protection of social network graphs. However, instead of using k-automorphism or k-symmetry, they used k-isomorphism. One of the major concerns of these approaches is that, in the process of transforming the original graph to its symmetrized version, they create new nodes and edges which, unfortunately, greatly increases the size of said graph. This of course burdens the server with a larger graph to store, but more importantly, slows query processing. Our chosen implementation proposes to only store a small part of the k-automorphic graph. It is shown that despite uploading a smaller portion of the graph, the correctness of query results is still guaranteed, only now with improved query optimization.

All three of these other implementations [1, 2, 3] are based on using symmetry of graphs to hide structural data, however, all failed to implement a method of filtering out false positives. The method of removing these false positives (by having the original data owner perform a computationally simple filter) is a major contribution by our selected implementation which is one of the factors that grant it superiority over previous proposals.

2.2. Increasingly Sophisticated Attacks

Liu and Terzi [4] developed an implementation of constructing k-anonymous graphs based solely on addition of edges to the input graph using the minimum number of additions. The key focus of this paper is on constructing the output graph with the least amount of performance cost in terms of big O time complexity. The reason for this focus is because the main barrier to adoption of computer security is cost and inconvenience. An algorithm with excessive time or space complexity will result in either noticeably poor performance for the user or increased cost in service level requirements. Adding more edges than necessary will cost more processing power from longer algorithm run times. Also, unnecessary edges result in a larger graph that will waste network bandwidth.

Liu and Terzi were able to develop a dynamic programming algorithm as an oracle. That is, the algorithm will either output the optimum solution, or indicate that an optimum solution does not exist. However, the optimization algorithm itself adds overhead to the whole process. Therefore, they decided to optimize the optimizer. The algorithm was modified into a greedy algorithm with a faster run time that delivered an optimal solution in practice for the vast majority of cases. The greedy graph construction algorithm is no longer an oracle, and will either output a "No" or "Unknown" if it can not determine a solution. An additional greedy algorithm will incrementally modify the interim solution until the k-anonymous degree is satisfied. This solution will be locally optimised, but is not guaranteed to be the most optimal solution. This approach delivers the most optimal solution in terms of real world cost vs benefit.

Zhou and Pei [5] add to this work by considering a slightly more sophisticated attack where background knowledge of the neighborhood, or the subgraph containing all one hop neighbors, is known to the attackers. Zhou and Pei also propose constructing graphs with k-anonymity and l-diversity, basing their approach to that of relational database security.

Tai, Yu, Yang, and Chen [6] extend the work of the previously mentioned implementations [4,5], by expanding the scope of the attack from degrees of individual nodes or the neighborhood to degree pairs. Knowing the degree of a particular node along with the degrees of immediate neighbors increases the probability of identifying the nodes. Tai et al. refer to this more sophisticated attack as a friendship attack. The friendship attack is more sophisticated than the neighborhood attack because it examines higher degrees of separation, rather than just the one hop neighbors. Basically, degree pairs answer the query, "How many friends does my friend have?" Their solution is to construct a k^2-anonymous graph that ensures that for a given node with a particular degree pair, there are at least k-1 nodes possessing an incident edge of the same degree pair. In other words, if I have five friends that each have two friends, and k^2 is two, then at least one of my friends has a total of five friends, and one of their five friends has two friends.

The approach retains the same number of vertices, but can both add and delete edges as necessary. The first phase consists of clustering vertices with similar degrees, selecting a target degree for each cluster, and ensures that each cluster contains at least k vertices. The second phase anonymizes the relationships between clusters. The final phase anonymizes inside the clusters. This algorithm is capable of scaling to large graphs.

3. Technical Section

As discussed, our protection against identity disclosure depends on transforming a given graph *G* into its k-automorphic version *G**. *G* is transformed into *G** in three steps: graph partitioning, block alignment, and edge copy. Essentially, once these steps are completed we will have added noise edges and noise vertices to a given graph G to achieve an anonymized, publishable, k-automorphic graph G*.

The first step, graph partitioning, involves splitting *G* into *k* subgraphs. This is an NP-complete problem, so the authors decided to use a heuristic partitioning algorithm as a speed optimization. The authors used an already developed algorithm, the METIS [7] graph partitioning algorithm to implement this step, as METIS has been shown to be very fast at partitioning large data graphs. In our implementation we call the METIS partitioning algorithm and pass it our desired *k* value. A list of partitions is returned from METIS which is used to build *k* subgraphs of the original graph *G*.

The second step is to perform block alignment, but before this can be done, an Alignment Vertex Table(AVT) must be constructed. The AVT is simply a table in which each column represents one subgraph, the values inside a column represent the vertices within that subgraph, and the rows represent the automorphic functions mappings of each vertex from one subgraph to their corresponding vertices in the other *k - 1* vertices. To build the AVT we first build an intermediate table by doing a breadth-first-search on each of our partitioned subgraphs. We pick a starting points for the BFS by selecting the vertex from each partition which has the greatest number of edges. We do this as part of the heuristic approach to minimize the total number of added edges necessary to achieve k-automorphism--the idea is that hub vertices get paired up at the start so that no vertex with a large degree will be paired with a vertex of a small degree. The intermediate table built during BFS stores each vertex as well as that vertex's distance from the starting hub point. Using this table, we pair up vertices that have similar degree and are of similar distances from the starting point. These pairing get placed together in rows of the AVT.

AVT Construction Algorithm:

1: Set all vertices in each partition as unvisited

2: From each partition, select a starting vertex which has the maximum degree of any vertex in its partition

3: Place each starting vertex into the first row of the AVT

4: Perform BFS from each starting vertex and load each vertex and its hop-count into the intermediate table.

5: Form each new row of the AVT by pairing up k vertices from k blocks that have similar hop-counts and degree values

6: Return the AVT

Once the AVT is built block alignment is performed. During this step, the AVT is used as the automorphic function to map vertices to their paired symmetric vertices. Our algorithm does two passes over the subgraphs. The first pass creates a role model subgraph to which all subgraphs should be identical. The subgraph represented by the first column of the AVT has the edges of every other subgraph copied over to its respective vertices and thus becomes the rolemodel. This role model subgraph now contains all of the edges that should be expressed in every subgraph. During the second pass, the edges of the role model partition are copied onto the corresponding vertices of all other subgraphs.

Block Alignment Algorithm:

```
For each block $P$, 2 to $k$ do
  For each edge($v_1$, $v_2$) in those blocks do
    If edge(AVT($v_1$), AVT($v_2$)) is not in block 1
      Then insert edge(AVT($v_1$), AVT($v_2$)) to block 1
For each block $P$, 2 to $k$ do
  For each edge($v_1$, $v_2$) in block 1 do
    If edge(AVT($v_1$), AVT($v_2$)) is not in block $P$
      Then insert edge(AVT($v_1$), AVT($v_2$)) to block $P$
```

The third and final step of creating the publishable G^* k-automorphic graph is Edge Copy. During this step, any edge that crosses a boundary between partitions must be copied to its respective vertices.

Edge Copy Algorithm:

```
For each child subgraph $g_i$ of parent graph $G$, 0 to $k$ do
  For each subgraph vertex $v$ in $g_i$ do
    If $v$ does not have same degree as corresponding
          parent vertex $V$
      Then For each edge $E$ in $V$ do
        If $v$ does not have edge $e$ matching $E$
         Then insert $E$'s symmetric edge $E$` in G
```

4. Evaluation

One of the primary concerns about the k-automorphism transformation is the loss of information that accompanies it. The process of adding noise edges to the graph introduces the possibility of false positives during data analysis. Therefore, the cost of k-automorphism can be quantified in terms of how many edges are added to G to get G^*.

Cost(G, G*) = (E(G) ∪ E(G*)) − (E(G) ∩ E(G*)) *where* E(G) *is the set of edges in G.*

We tested our implementation by running it on five randomly generated graphs, consisting of 50 nodes and 100 edges. We compare the number of edges of each original graph G to G^*. We first tested with $k = 2$ and found that 83, 92, 70, 75, and 81 edges were added to each of the five test graphs, respectively. This is a average of 82% increase in edge count. Finally, we tested a much larger graph consisting of 50,000 nodes and 100,000 edges. The run time was 50 minutes and the number of edges roughly doubled. This shows that the algorithm scales well. A tenfold increase in nodes and a 100 fold increase in edges only resulted in a 20% increase in cost.

5. Summary

The solution proposed by Zou, Chen, and Ozsu is a simple and elegant one. Although it incorporates several NP-hard problems which are inherently difficult to code and debug even before attempting to run them on large graph data (which multiplies the difficulty by orders of magnitude), the basic idea behind the protection is straightforward: a degree of privacy is breached via structural information in published graph data, so to combat this breach, such structural information can be duplicated several times prior to publishing. This duplication ensures an adversary has no way of knowing if the target he believes he has located, is simple a symmetric version of that target--a decoy, in short. This is a "safety in numbers" strategy that, when combined with previous methods anonymizing vertices by stripping identifying labels and attributes, provides protection against a wide-range or identity disclosing attacks. In fact, the k-automorphism algorithm guarantees protection against all types of structural attacks [1]. Given the increasingly popular nature of social networking application, having a systematic method of securing the privacy of individuals while still being able to publicly publish network data for the purpose of important data mining is hugely important.

6. Team Work

Grady was responsible for figuring out how to include and link the necessary libraries (including Boost and METIS) so that programs would compile, and for writing and maintaining the appropriate makefiles. He also wrote the code for reading and writing the graphs, for many of the complicated typedefs, for the Edge Copy algorithm and for many smaller sections of code.

Shane was responsible for deciphering the paper the project was based on, and preparing test cases to validate the program He also coded the Block Alignment algorithm, one of the AVT constructing algorithms, and other snippets of code.

Both contributors worked closely together in debugging many parts of the code, divided all writing assignments evenly and met consistently and frequently for the duration of the project.

7. Project Repository

All code developed for this project can be found at:

https://github.com/McFlip/k-automorphic-graph

References

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

[2] J. Cheng, A. W. Fu, and J. Liu. K-isomorphism: Privacy preserving network publication against structural attacks. In SIGMOD, pages 459–470, 2010.

[3] W. Wu, Y. Xiao, W. Wang, Z. He, and Z. Wang. K-symmetry model for identity anonymization in social networks. In EDBT, pages 111–122, 2010.

[4] K. Liu and E. Terzi. Towards identity anonymization on graphs. In SIGMOD, pages 93–106, 2008.

[5] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In ICDE, pages 506–515, 2008.

[6] C.-H. Tai, P. S. Yu, D.-N. Yang, and M.-S. Chen. Privacy-preserving social network publication against friendship attacks. In KDD, pages 1262–1270, 2011.

[7] G. Karypis and V. Kumar. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. In SIAM Journal on Scientific Computing, Vol. 20, No. 1, pp. 359—392, 1999.