Project Proposal, Anonymized Social Networks- CS6150 Advanced Algorithms

Paper: 'Wherefore Art Thou R3579X? Anonymized Social Networks, Hidden Patterns, and Structural Steganography' - Backstrom, Dwork, Kleinberg, 2007

Team Members:

Scott Gale, u1203422, scottdgale@gmail.com

Sudie Roweton, u1210082, sudie.roweton@gmail.com

Dennis Njeru, u1076700, kansla@yahoo.com

Github repository: https://github.com/scottdgale/Algorithms-Project

1. Introduction

Anonymization of social networks allows researchers the valuable opportunity to study the structure of social networks while protecting the privacy of the social entities or users of those networks. The question becomes, is anonymization enough to prevent adversaries from compromising privacy? This paper discusses attacks that an adversary can utilize to compromise the privacy of the social entities that exist in an anonymized copy of a social network. Compromising privacy in this context refers to discovering whether a relationship exists between targeted users.

The paper presents two main categories of attacks: active and passive. Active attacks are those in which the adversary creates fake user accounts and establishes links from those newly created accounts to a targeted group of users that already exist in the network thus creating a subgraph in the network. In contrast, passive attacks are those in which the adversary already exists in the network and colludes with a group of people that already exist in the network; thus, the subgraph exists in the network without having to create new user accounts. The goal of the passive attack is to uncover relationships among users that the adversaries are already linked to. There exists another type of attack, the semi-passive attack, that is a natural extension to the passive attack. The semi-passive attack is similar to the passive attack except the colluding adversaries each connect to specific targeted user(s). The idea is, if the adversaries can find their subgraph, then, by extension, they would have found the targeted user(s). Walk-based and cut-based attacks are different active attack approaches to recover that subgraph in the network. The walk-based active attack will be the focus of our project.

2. Formal description of the algorithmic problem

The structure of a social network is a graph where the nodes represent the users and the edges represent relationships between the users. The algorithmic problem presented in the paper centers around how to determine if two (or more) anonymous nodes in a graph have a relationship (edge), thus, compromising privacy. Formally, there is a targeted set of users $w_1, w_2, w_3, \ldots, w_n$ in an anonymized copy of the social network, G. Then, the problem to be solved is to determine whether or not the edge (w_i, w_j) exists for every i, j.

This is done by creating a relatively small graph and implanting it into a much larger graph. The small graph is connected to targeted nodes, $w_1, w_2, w_3, . . . , w_n$ in the large graph prior to the graph being anonymized. Once the graph is anonymized, the goal is to locate the small implanted graph to determine relationships that exist between the targeted nodes in the larger graph.

3. Algorithm (Including parameters used for our simulations)

Overview

Let $G = (V, E)$ be a n-node anonymized undirected graph.

Choose set of targeted users: $W = \{w_1, w_2, w_3, \ldots w_b\}$ where w_i is an individual node in a network and where $b = O(log^2 n)$. **Note:** Variable b constitutes the maximum number of targeted vertices ($b = 98$ for a graph containing 20,000 vertices). For our implementation we use $b = log^2 n * 0.65$ (or 65% of the maximum). We found that the closer we came to the upper bound of b the less likely the algorithm worked properly . . . we could not always recover W.

Goal: Identify targeted nodes in G denoted by $W = \{w_1, w_2, w_3, \ldots w_b\}$. Try to discover if the edge (w_i, w_j) exists . . . looking for relationships between nodes in W.

Algorithm - Active "Walked-Based Attack"

(a) Create k new nodes: $X = \{x_1, x_2, x_3, \ldots, x_k\}$ where $k = (2 + \delta)logn$; where $\delta > 0$ (small constant). **Note:** For our implementation we use $\delta = 1$.

(b) Create a random graph H (defined by X) to insert into G; H will connect to G in a clever way such that we can discover post anonymization.

(c) H is connected to G by creating an edge from a node in H to a node in G. Thus creating edges (x_i, w_i).

(d) Deterministically create edges between nodes (x_i, x_{i+1}) - this allows traversal of X in order.

(e) Choose two constants $d_0 \leq d_1 = O(log(n))$. These values set the parameters of the external number of edges that connect H to G. **Note:** For our implementation we set $d_1 = log(n)$ where n is the number of vertices in the graph. We set $d_0 = \frac{1}{2} d_1$.

(f) For each $i, i = 1, 2, 3, \ldots, k$ we choose an external degree $\Delta_i \in [d_0, d_1]$ specifying the number of edges x_i will have to nodes in $G - H$. These are chosen uniformly at random from the interval $[d_0, d_1]$.

(g) Choose a set $N_j \subseteqq \{x_1, x_2, x_3, \ldots, x_k\}$ where each x_i is unique and $N_j \leq c$ where c is small constant. For our implementation we use $c = 3$. Each x_i can appear in at most Δ_i of the sets N_j. If an x_i appears in Δ_i sets then it does NOT contain any other edges to vertices in G.

(h) Construct links to w_j from each $x_i \in N_j$.

(i) Create random edges in H where the edge (x_i, x_j) has $\frac{1}{2}$ probability of existing.

(j) $\Delta_i' =$ the degree of x_i of all nodes (from edges in both G and H).

-------------------------------Network gets anonymized--

Recovering H given G

(a) Search G for every node with degree Δ_1'.

(b) Successively try and add nodes with degree Δ_i' that also have the correct edges to other corresponding nodes in X. **Note:** We used a tree and recursion to solve this problem. We first iterate through all the nodes in the graph acquire a subset of nodes that match the degree of x_0. This becomes our candidate x_0 list and each one as treated as a child in the tree of the root. Next we look at the neighbors of all candidates and compare to see if any match the degree of x_1 - any that are found are added to the tree. The process continues recursively and exits at the first instance the depth of the tree equals the number of nodes in x.

(c) Once H is discovered, discover $W = \{w_1, w_2, w_3, \dots w_k\}$ from the unique relationship between each w_j and N_j. **Note:** This process is fairly straightforward. Since we know the makeup of each N_j we can discover the common nodes connected to each x in N_j to identify a w.

4. Other baseline algorithms (These were mentioned in the paper - we did NOT implement)

Cut-Based Attack:

Another active attack, the “cut-based attack”, is constructed by attaching H to G using very few edges and recovering H using Gomory-Hu cut trees. However since H is “thinly” attached to G, H will be unique not only to the attacker, but also potentially to an observer (i.e. easier to detect). The cut-based attack has the advantage of matching the tight theoretical bound on the number of nodes needed to recover H. However the use of Gomory-Hu trees makes the recovery algorithm more expensive than that of the walk-based attack. Additionally, the cut-based attack can only compromise O(k) users as compared to O(k2) users for the walk-based attack.

Passive Attacks:

In a passive attack, regular users are able to discover their locations in G using their knowledge of the local structure of the network around them. An example of this kind of attack is where a small coalition of passive attackers collude to discover their location. By doing so, they compromise the privacy of some of their neighbors: those connected to a unique subset of the coalition, and hence unambiguously recognizable once the coalition is found. This passive attack is analogous to the walk-based attack, except that the structure of H occurs organically as a natural function of individuals using the system. Since the coalition X has not specifically targeted any nodes, it is possible that although they can uniquely find themselves, they cannot locate any specific users other than themselves. However, empirical evidence suggests that once a coalition is moderately sized, it can compromise the privacy of at least some users. The primary disadvantage of this attack as compared to the active-based attack is that it does not allow one to compromise the privacy of arbitrary users.

5. Project Implementation

Objective: We implement the Active Walk-Based Algorithm as discussed in section 3 then deterministically removed an increasing number of edges and determined at what point the algorithm became ineffective. We used the Java programming language to implement this project. We created our own graph class which provided us maximum flexibility in performing multiple simulations.

Step 1: Create Datasets. We found several large anonymized social network datasets on the Stanford Network Analysis Project. We created our own datasets modeled after these datasets by replicating the number of nodes, edges, average number of edges per node, etc. We also experimented with graphs of various size ranging from 2,000 - 20,000 nodes and from 20,000- 500,00+ edges respectively. One advantage of creating our own datasets was the ability to quickly alter attributes such as number of nodes, number of edges, and number of edges per node. **Note:** We included "dataset_1.txt" to illustrate one of our sample datasets. This is a visual text document of the graph G.

Step 2: Implement the "Walk-Based Attack" algorithm. We implemented the "Walk-Based" algorithm on multiple datasets. We worked with the baseline algorithm to ensure it was working properly and we could recover H with a near 100% probability (We recovered H 100% of the time conducting over 50 simulations). Because we created our own datasets there was no anonymization performed. We addressed this issue by not allowing our recovery function access to graph data and variables other than to identify the uniqueness of a node.

Step 3: Deterministically remove edges to discover the effectiveness of the algorithm. Once the baseline algorithm was functioning properly we deterministically started cutting edges (1%, 2%, . . . n%) up to the point where we could no longer recover H. Edges were removed using a random function operating on all the edges (no distinction between edges in G and H).

Note: Please refer to "code_instruction.pdf" for detailed information on how to run the code that produced these results.

6. Observations

We tested this algorithm on two sizes of datasets:

Dataset 1:

Vertices: 4,044

Targeted Vertices (W): 44

Total Edges: ~100,000 (created randomly)

Average Degree per Edge: 54

Dataset 2:

Vertices: 20,063

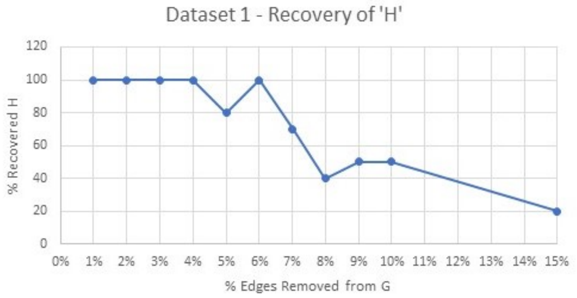
Targeted Vertices (W): 63

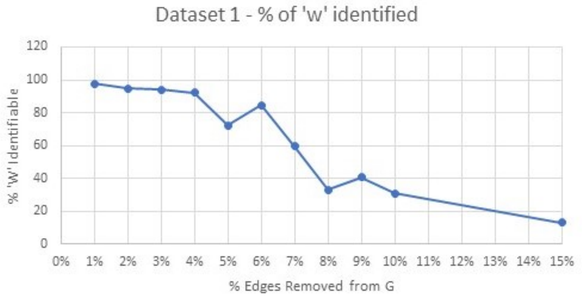
Total Edges: ~547,000 (created randomly)

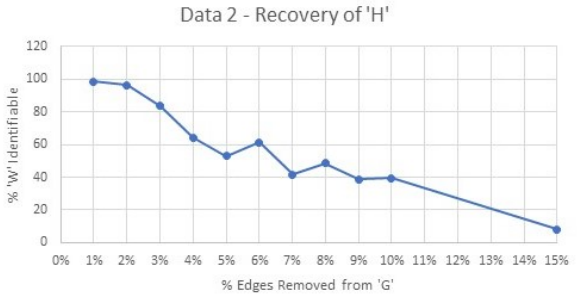
Average Degree per Edge: 54

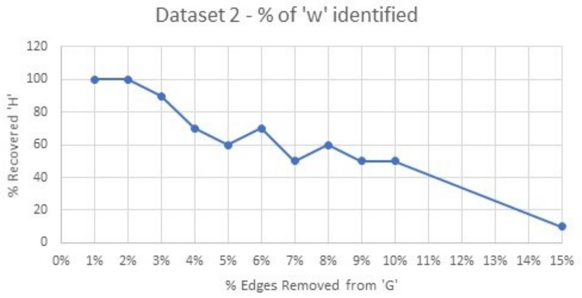
For each dataset size we were able to consistently recover H and successfully identify the relationships that exist in W 100% of the time (over 30 simulations) running the published algorithm without the defensive measure of edge removal. We conducted 200+ simulations on both datasets beginning with the removal of 1% of the edges, up to 10% of the edges at an interval of 1%. We then did an additional test removing 15% of the edges. Each simulation was executed 10 times and the results were averaged to produce our results.

For each dataset we have produced two graphs. First, the probability of recovering H. Second the percentage of w's (targeted vertices) that we could uniquely identify. The recovery of H is binary - either we recover it or we don't - there is no partial recovery. If we FAIL to recover H then we CANNOT recover any of the w's (targeted) nodes since the recovery relies solely on first identifying H. However, just because we successfully recover H doesn't guarantee we can successfully recover all the w's. This is true because all the edges used to recover H may be intact; however, edges between H and w's may have been severed. Thus, the second graph illustrates the probability of w's recovered at each step in the edge removal process.









Note that when we removed 6% of the edges on Dataset 1 (upper left graph), we were still able to recover H 100% of the time (10 simulations). This outlier illustrates the randomness that exists if an organization were to cut edges at random. If edges in x were not cut, then the algorithm is still going to work with high probability.

7. Conclusion

(a) The algorithm and resiliency to cuts does not change with respect to the size of the graph. There were no significant differences in the probability of recovering H or the percentage of w's recovered between the two datasets we tested.

(b) When we cut approximately 7% of edges we cross the 50% threshold of success probability to recover H . Consequently, the probability to recover w's is slightly below that. These results were consistent throughout all of our testing.

(c) When we cut 15% of the edges we approached a 0% probability of recovering H. Therefore, we can conclude that if an organization wanted to protect against this specific attack - cutting 15% of the edges would give a very high probability of failure for recovering any relationships in w.