When The Attacker Knows A Lot: The *GAGA* Graph Anonymizer

Arash Alavi, Rajiv Gupta, and Zhiyun Qian

University of California, Riverside
aalav003@ucr.edu, gupta@cs.ucr.edu, zhiyunq@cs.ucr.edu

Abstract. When releasing graph data (e.g., social network) to public or third parties, data privacy becomes a major concern. It has been shown that state-of-the-art graph anonymization techniques suffer from a lack of strong defense against De-Anonymization (DA) attacks mostly because of the bias towards utility preservation. In this paper, we propose *GAGA*, an Efficient Genetic Algorithm for Graph Anonymization, that simultaneously delivers high anonymization and utility preservation. To address the vulnerability against DA attacks especially when the adversary can re-identify the victim not only based on some information about the neighbors of a victim but also some knowledge on the structure of the neighbors of the victim's neighbors, *GAGA* puts the concept of *k(d)-neighborhood-anonymity* into action by developing the first general algorithm for any d distance neighborhood. *GAGA* also addresses the challenge of applying minimum number of changes to the original graph to preserve data utilities via an effective and efficient genetic algorithm. Results of our evaluation show that *GAGA* anonymizes the graphs in a way that it is more resistant to modern DA attacks than existing techniques – *GAGA* (with *d=3*) improves the defense against DA techniques by reducing the DA rate by at least a factor of 2.7× in comparison to the baseline. At the same time it preserves the data utilities to a very high degree – it is the best technique for preserving 11 out of 16 utilities. Finally, *GAGA* provides application-oriented level of control to users via different tunable parameters.

Keywords: graph anonymization · data privacy · network security.

1 Introduction

Social network data are routinely released for different purposes such as advertising, academic research, medical diagnosis, criminology, etc. Since these data contain sensitive information about the users, when releasing such data to the public or third parties, data privacy is a major concern. To achieve privacy preservation, various graph anonymization techniques have been proposed to anonymize the social graph data before its release [4, 13, 17, 23, 28, 29].

The main drawback of existing anonymization techniques is that they trade-off anonymization with utility preservation. Previous works including [4,17,23,28, 29] evaluated their approaches only in terms of data utility performance and lack

a complete evaluation of their ability to withstand modern De-Anonymization (DA) attacks [10, 14, 22, 24, 27]. As shown in [11], existing graph anonymization techniques not only produce anonymized graphs that are *susceptible to DA attacks*, they also *make more than the minimum required changes* to the original graph even though applying minimum number of changes is their primary objective.

In this paper, we propose *GAGA*, an Efficient Genetic Algorithm for Graph Anonymization, that simultaneously achieves high anonymization via utility preservation. *GAGA* defends against modern DA attacks as follows:

– ***K(d)-neighborhood-anonymity* for any *d*** is supported by *GAGA* to provide defense against modern DA attacks. Existing anonymization techniques have been found to be ineffective against modern DA attacks especially when the attacker has complex knowledge about the structure of some-order neighbors (e.g. , neighbors of neighbors) of a victim, which can be obtained by the attackers by either modifying the social network graph before releasing by creating fake users and linking to victim and its neighbors (active attack) or by trying to find themselves in the released graph and from this and many other auxiliary graphs (e.g., other social networks) discover the structure of neighbors of the victim (passive attack).

At the same time, the combination of various features enables *GAGA* to preserve utilities and to give more application-oriented level of control to its users (researchers, advertisers, developers, etc.) as follows:

– ***Edge switching*** is supported by *GAGA* to preserve *degree* and its related utilities (e.g., *role extraction*) since *edge switch* is the only known technique that can effectively preserve these utilities [11]. Therefore, in contrast to most anonymization techniques that add some fake edges to the original graph, *GAGA* gives higher priority to edge switching over edge adding or removing. Meanwhile it has been observed that *k-neighborhood-anonymity* based algorithm is generally the best approach to partially or conditionally preserve other utilities. Thus, to cover as much as possible utilities, *GAGA* applies *edge switch* to the *k(d)-neighborhood-anonymity* model. The genetic algorithm further minimizes the number of edge switches for better utility preservation.

– **Controls via *k* and *d*** are provided by *GAGA* allowing application-oriented level of control by its users. If defending against modern DA attacks is desired, larger values of k and d are used. If merely utility preservation is demanded, small values of k and d are used.

– **Controls via genetic algorithm (GA)** are also provided by *GAGA*. GA is effective and efficient for solving optimization problems (here, applying minimum number of changes). Besides, its tunable parameters allow not only control over solutions quality in terms of preserving utility but also runtime performance of the tool (albeit, graph anoymization is usually an offline process). These parameters include: initial population size; number of switches (s), adds, and removes (mutation rate) in each GA iteration; probabilities that control the search space; finding local maxima (a set of good enough solutions) as opposed to the global maximum (best solutions).

Our evaluation leads to the following key conclusions.

- First our comparison of *GAGA* with existing anonymization techniques with respect to multiple DA attacks, for a subgraph of Facebook friendship network, shows that *GAGA* is the best for defending against all DA attacks when it employs *d=3-neighborhood* structures. We set *Union Cluster* [25] as our evaluation baseline and show that *GAGA* improves the defense against DA techniques by reducing the rate of successfully de-anonymized users by at least a factor of 2.7× when *d=3* in comparison to the baseline and leads to zero de-identified users in some cases.
- Second our comparison of *GAGA* with existing anonymization approaches, for a real world input graph, shows that under 16 graph and application utility metrics, *GAGA* is overall the best at preserving utilities – it is the best for 11 out of 16 utilities and close to the best for the remaining 5.
- Finally, our comparison of *GAGA* with Zhou and Pei's [29] work that uses *d=1-neighborhood* anonymity model shows that *GAGA* incurs only 69% of the cost of Zhou and Pei's approach when it also employs *d=1-neighborhood* structures, indicating that previous works cannot preserve most utilities though this is their primary objective.

The remainder of the paper is organized as follows. We discuss background and motivate our work in section 2. Section 3 presents the details of *GAGA*. Section 4 evaluates *GAGA* and compares it with state-of-the-art techniques. We discuss related work in section 5. The paper ends with conclusions in section 6.

2 Background and Motivation

Preserving data privacy has been widely studied. One of the main approaches used to preserve data privacy is based upon the concept of anonymity. Graphs and databases have played an important role in this domain [1–3, 18, 20]. In this paper we address the data privacy preservation in graphs, specifically for graphs representing social networks. A number of graph anonymization techniques have been proposed to preserve users' privacy. We discuss the limitations of these techniques first from the perspective of defense against DA attacks and then from the perspective of utility preservation to motivate our approach.

As a concrete motivating example, consider the sample graph shown in Figure 1. The social graph on the left side is going to be publicly published. Assume that an adversary knows that Alice has 2 friends and each of them has 4 friends, then the vertex representing Alice can be re-identified uniquely in the network (black vertex in Figure 1). The reason is that no other vertices have the same *2-neighborhood* graph to the *2-neighborhood* graph for Alice. Existing graph anonymization techniques fail to anonymize this example graph so that an adversary cannot re-identify any user certainly. The *k-degree-anonymity* based algorithm in [17] removes/adds edges from/to the original graph to create a graph in which for every vertex there are at least *k-1* other vertices with the same degree. Based on *k-degree-anonymity*, the graph is *2-degree-anonymized*.

In the other approach, *k-neighborhood-anonymity* based algorithm in [29] adds edges to the original graph to create a graph in which for every vertex there are at least *k-1* other vertices with the same *1-neighborhood* graphs. Based on *k-neighborhood-anonymity*, the graph is *2-neighborhood-anonymized*. Hence, the existing *k-anonymity* approaches are inadequate when the attacker has more complex knowledge about the neighborhood structures.

Based upon the above discussion, we conclude that we must support *k(d)-neighborhood-anonymity* for any *d*, instead of *k-degree-anonymity* or *k-neighborhood-anonymity* for *d=1-neighborhood* considered in prior works. That is, our approach will provide an algorithm that efficiently enables *d-neighborhood* privacy preservation for any *d* to protect against attacks that use complex neighborhood acknowledgements of the target vertex.

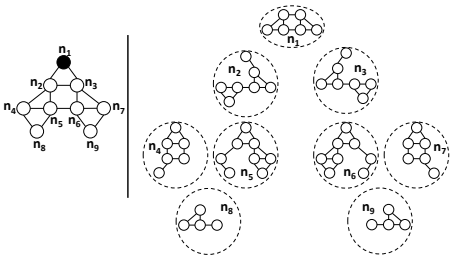

Fig. 1. Graph to be publicly published on the left and *2-neighborhood* structures for each vertex on the right. Black vertex represents Alice.

Next we consider the issue of utility preservation. *SecGraph* introduced by Ji et al. [11], evaluates different anonymization algorithms using various utilities. According to their study, *k-neighborhood-anonymity* preserves most of the graph and application utilities. The one application utility which *k-neighborhood-anonymity* algorithm cannot preserve is the *role extraction* utility where it considers the uniqueness of each vertex based on their structure in the graph. Among all anonymization algorithms, the *Rand Switch* approach introduced in [28] where existing pair of edges are switched randomly *n* times, is the only one that can preserve *role extraction*.

Because of the above reason, in this paper we give higher priority to edge switching over edge adding and removing since *edge switching* can effectively preserve degree and its related utilities (e.g., role extraction) leading to preserving more utilities. We further apply *edge switching* to the *k(d)-neighborhood-anonymity* model and use Genetic Algorithm as the main approach for utility preservation.

Summary: With more knowledge about the local neighborhood structures in a social network, an adversary has more chances to re-identify some victims. We show that existing anonymization techniques not only do not present a complete model to *defend against DA attacks*, specially structure-based attacks, but also they *make more than the minimum required changes*. In contrast, an additional goal of our approach is applying fewer changes and thus providing a better trade-off between anonymization and utility preservation. As a motivating example, Figure 2(a) depicts the original graph, and the anonymized graphs generated by

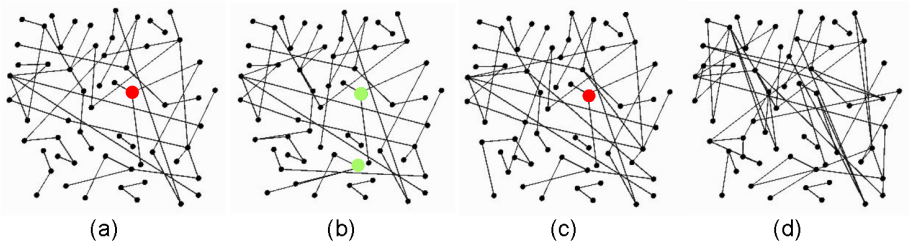

Fig. 2. Sample social network graph (a) Original graph to be publicly published, bigger colored node shows the victim vertex (b) Anonymized graph using our approach with *k=2* and *d=2*, bigger colored nodes show the victim vertex and the vertex with similar 2-neighborhood (c) Anonymized graph using *k-degree Anonymization* in [17] with *k=2* (d) Anonymized graph using *Random Walk Anonymization* in [21] with *r=2*.

our approach, *k-degree Anonymization*, and *Random Walk Anonymization* techniques using the minimum values for the parameters of each approach (*k=2* and *d=2* for our approach, *k=2* for *k-degree Anonymization*, and *r=2* for *Random Walk Anonymization*). Assume an adversary knows that a user has 3 friends and only one of them has another friend, then the user can be re-identified easily (colored bigger vertex in Figure 2(a)), since this is the only user with that friendship neighborhood structure.

(Our Approach) In Figure 2(b), our approach applies minimum number of changes of 3 edges switches and 1 edge removal to the original graph (i.e., we preserve degrees for all vertices except for only two vertices). We anonymize the graph in a way that for each vertex there is at least one other vertex with similar *2-neighborhood* structure (i.e., there is another user with similar *2-neighborhood* friendship to the target user *2-neighborhood* depicted with two colored vertices which reduces the re-identification chance by 50%). Note that for simplicity of presentation, we consider *k=2* and *d=2*. Larger values of *k* and *d* reduce the attacker's chance of success.

(*K-degree Anonymization*) In Figure 2(c), by applying slightly more changes compared to our approach, the *k-degree-anonymity* concept introduced in [17] is achieved which is weaker in comparison to *k(d)-neighborhood-anonymity*. This means that for each vertex there is at least one other vertex with similar degree which is already satisfied with our approach. Hence, the adversary can still re-identify the target user easily.

(*Random Walk Anonymization*) In Figure 2(d), while introducing much more noise compared to our approach, this technique only ensures some level of link privacy. The reason of comparing our approach with *Random Walk Anonymization* technique is that it is the only graph anonymization technique which takes the concept of neighborhoods structures into consideration. That is, in social network graph *G*, replace an edge *(u,v)* by the edge *(u,z)* where *z* denotes the terminus point of a random walk algorithm. As a result, noise is introduced into the graph leading to huge data loss.

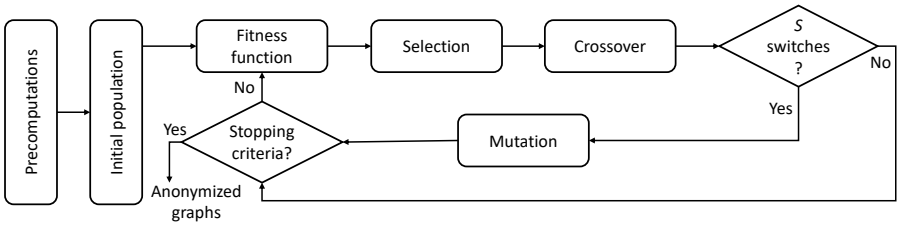

Fig. 3. *GAGA* overview.

3 GAGA

In this section, we present an Efficient Genetic Algorithm for Graph Anonymization (*GAGA*). *GAGA* creates an optimal anonymized graph by applying minimum number of changes to the original graph in comparison to existing approaches which make the graph less responsive to various queries. *GAGA* can preserve data privacy against many complicated DA attacks. To achieve these goals we use Genetic algorithm (GA) as the main approach. Our reasoning is that first, Genetic Algorithms are very effective in achieving optimal or near-optimal solutions in a variety of single- and multi-objective problems (e.g., classification, game theory, bioinformatics, etc.). Hence, achieving an optimal anonymized graph as a two-fold optimization problem (i.e., achieving a *k(d)-neighborhood-anonymized* graph and applying minimum number of changes simultaneously) fits a genetic algorithm-based model very well. Second, different tunable parameters in genetic algorithm helps to avoid leaving some areas of the search space undiscovered, resulting in widening the search space more that the other approaches. In this section, we describe how we apply GA to the graph anonymization problem. Figure 3 shows an overview of *GAGA*. Now we discuss each step of GA that we used in *GAGA*:

3.1 Precomputation Step

Before applying the GA to the original graph, we perform precomputations that evaluate the original graph so that we can choose the best parameters to create the optimal *k(d)-neighborhood-anonymized* graph. As a result, the original graph is categorized as one of: *good*, *bad*, or *ugly* scenarios. In the *good* scenario, the original graph is close to a *k(d)-neighborhood-anonymized* solution and hence it needs a small number of changes. In the *bad* scenario, many vertices do not satisfy the *k(d)-neighborhood-anonymity* and hence the original graph needs changes to large number of vertices. In the *ugly* scenario, few vertices violate the *k(d)-neighborhood-anonymity* but they have a very different neighborhood compared to other vertices; hence it requires huge changes to a small number of vertices. Our precomputations involve the following steps:

Step 1. Percentage of violating vertices: We identify the vertices that violate the *k(d)-neighborhood-anonymity* (i.e., there are less than k-1 other ver-

tices with similar *d-neighborhood* to the *d-neighborhood* of these violating vertices) and compute the percentage of violating vertices. A low percentage of violating vertices means that by applying some changes to a small group of vertices, we can create a *k(d)-neighborhood-anonymized* graph. We further observe that the changes can be small or big themselves. Hence, we consider a threshold value (T_{pv}) and if the percentage is below the threshold value, we consider the graph as one of the *good* ones (i.e., small changes to small number of vertices are required) or an *ugly* one (i.e., big changes to small number of vertices are required). If the percentage is above the threshold value, we consider the graph as *bad* (i.e., some changes to large number of vertices are required).

Step 2. Violating vertices' neighborhoods analysis: After the previous step, the original graph is categorized as *good/ugly* or *bad*. To distinguish between *good* and *ugly* scenarios, we analyze the neighborhoods around violating vertices and compare them with the neighborhoods of vertices that satisfy the *k(d)-neighborhood-anonymity*. If some of the violating vertices have a very different neighborhood than others (we simply compare degrees for this purpose), we categorize the graph as *ugly*. Otherwise, we categorize the graph as *good*. To analyze the rate of difference, we again define a threshold value (T_u) so that if the value is above the threshold we consider the graph as *ugly* scenario. Otherwise, if the value is below the threshold, we consider the graph as *good* scenario. We illustrate the scenarios using three sample graphs in Figure 4. The treatment for each of three scenarios is described next:

Good Scenario. In the *good* scenario, in the beginning of the GA process, we focus only on violating vertices according to a probability and apply the GA to them. For this purpose, we select the vertices –selection in GA– from violating vertices to apply the changes (switches, adds, removes) –mutation in GA– so the number of violating vertices will decrease. As we proceed forward towards the end of the process, we select some vertices from other non violating vertices and apply the changes to them based on a probability. This increases the probability of searching more areas of the search space causing the graph to become *k(d)-neighborhood-anonymized* faster.

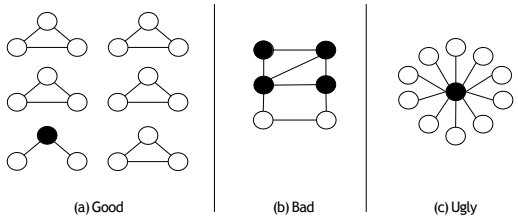

Fig. 4. Black vertices represent the violating vertices. Assume that k=2, d=1, and T_{pv}=10%; i.e., the graph is *2(1)-neighborhood-anonymized* if for each vertex of the graph, there is at least one other vertex with similar *(1)-neighborhood* graph. (a) *Good scenario*: 5% of vertices violate the *2(1)-neighborhood-anonymity*. (b) *Bad scenario*: 66% of vertices violate the *2(1)-neighborhood-anonymity*. (c) *Ugly scenario*: 9% of vertices violate the *2(1)-neighborhood-anonymity* but the violating vertex has a very different neighborhood than other vertices.

Bad Scenario. In the *bad* scenario, there is no advantage to focus only on violating vertices neighborhoods. So we apply the GA to the whole graph. For this purpose, we select the vertices –selection in GA– from all the vertices in graph to apply the changes. In comparison to the *good* scenario, the *bad* scenario, in general requires more changes and thus more time to create the *k(d)-neighborhood-anonymized* graph. As we will see in Section 4, even in *bad* scenarios, our results are much more efficient in terms of minimum number of changes and hence utility preservation compared to the existing techniques.

Ugly Scenario. In the *ugly* scenario, we again focus on violating vertices in the beginning of the GA process like in *good* scenario but we apply more changes in each step of GA compared to *good* scenario so that the graph becomes *k(d)-neighborhood-anonymized* faster. Again, as we move forward, we select some vertices from other non-violating vertices to increase the probability of searching more areas of the search space.

3.2 Initial population

In this step we randomly apply edge switches on the original graph to create a fixed number of chromosomes as the initial population. We present chromosome representation details in section 3.6. As we discussed earlier, we create a larger initial population in *bad* scenarios compared to the *good* and *ugly* scenarios.

3.3 Fitness function and Selection

For each chromosome, the fitness value – which defines how good a solution the chromosome represents – is computed and the chromosomes are selected for reproduction based on their fitness values. Therefore, first, we need to define a function which computes the distance between the modified graph and the original graph (fitness function) and second, we need a function to compute the distance between the modified graph and the solution of a *k(d)-neighborhood-anonymized* graph (selection function). We define the fitness function as below:

$$fitness(G, \tilde{G}) = \frac{1}{size((E \setminus \tilde{E}) \cup (\tilde{E} \setminus E))} \tag{1}$$

Given the original graph *G(V,E)*, V is the set of vertices and E is the set of edges in G, and the modified graph $\tilde{G}(\tilde{V},\tilde{E})$, we evaluate the distance between the modified graph and the original graph by computing the number of edges in the union of relative complement of E in $\tilde{\text{E}}$ and relative complement of $\tilde{\text{E}}$ in E. Finally, we consider the inverse of the computed number of different edges so that a graph with higher fitness value has fewer changes. After we compute the fitness values, we use roulette wheel selection so that the chromosomes with a higher fitness value will be more likely to be selected. With this method, in each step of GA we select those chromosomes which need fewer modifications to the original graph. As we discussed earlier, we need to define a selection function as well. We define the selection function as the inverse of the number of vertices

in the graph that do not satisfy the *k(d)-neighborhood-anonimity* concept for a given k and d. Using this selection function, in each step of GA, we select those chromosomes which are closer to the solution.

3.4 Crossover and mutation

Crossover and mutation are the two basic processes in GA. Crossover process copies individual strings (also called parent chromosomes) into a tentative new population for genetic operations and mutation is used to preserve the diversity from one generation to the next. Mutation prevents the GA from becoming trapped in a local optima. For crossover, the main function we employ is *edge switch* as follows. Given graph *G(V,E)* and a pair of edges $(u,v) \in E$ and $(w,z) \in E$ such that $(u,w) \notin E$ and $(v,z) \notin E$, we remove edges *(u,v)* and *(w,z)* and we add edges *(u,w)* and *(v,z)* to the graph. Note that *edge switch* can be considered as the process of combining the parent chromosomes where the parents are the chromosome and a copy of itself.

For mutation, we remove/add one or some number of random edges to some chromosomes. Specifically in our GA, first we try to perform *edge switch* for a certain number of times (s in Figure 3). If we fail to reach to a solution by applying s edge switches, then we start to remove/add one or some number of random edges to some chromosomes to create the new generation and then we repeat the GA for the new generation. If it is a *good* scenario, we remove/add very small number of edges in each step and if it is a *bad* scenario, we remove/add greater number of edges in each step. To decide whether to add or remove edges, if the selected vertex has a degree higher than average graph degree, we remove an edge while if the vertex degree is lower than average degree, we add an edge.

3.5 Checking stopping criteria

GAGA always returns at least one *k(d)-neighborhood-anonymized* graph as the solution by trying to apply minimum number of changes (switches, adds, removes) to the original graph. Therefore, in general we only have one stopping criteria except for invalid cases\inputs i.e. suppose a graph *G(V,E)* with $|V|$=n is given and a *k(d)-neighborhood-anonymized* graph is requested for some k>n. The problem has no solution unless we add fake vertices. *GAGA* does not introduce any fake vertices as in some previous works [6, 16]. As noted in [29], adding fake vertices is not desirable because this can change the global structure of the original graph. By maintaining the original utilities of the published social network, *GAGA* ensures that the changes are likely to have little or no impact on solutions to many applications/queries.

3.6 Implementation highlights

We implemented *GAGA* in Java. The implementation challenges are as follows.

Chromosomal representation. As discussed earlier, we need to represent the graph in an effective way such that *k(d)-neighborhood-anonymity* concept can

be put into action and the d distance neighborhood for each vertex can be easily considered. For this purpose, we represent the graph as a *HashMap* structure where each key represents a vertex of the graph and the value represents the *d-neighborhood* structure around the vertex.

Thresholds and parameters. As we discussed, k and d are the main parameters of *GAGA* which provide the data owners some application-oriented level of control over achieving the desired level of data preservation and anonymization. Besides the k and d parameters, *GAGA* contains other thresholds and parameters used in GA: initial population size, s as the number of maximum edge switches before remove/add one or some number of random edges, ,thresholds T_{pv} and T_u to categorize the scenario of the graph, a parameter to indicate finding local maxima as opposed to the global maximum for scenarios where the user/data owner can tolerate some number of violating vertices. *GAGA* receives the above parameters as the input.

Graph isomorphism test. The graph isomorphism tests are frequently conducted in the selection phase of GA. For this purpose, we used the VF2 algorithm introduced in [7] as a (sub)graph isomorphism algorithm with efficient performance specially for large graphs. Since the nature of any isomorphism test is that it takes time, we perform multiple level of prechecks to avoid applying the algorithm as much as possible. As a simple example, when two subgraphs have different number of vertices (or edges), or different degree sequences, we do not apply the VF2 algorithm.

4 Experimental Evaluation

In this section, first, we evaluate the effectiveness of *GAGA* against the existing De-Anonymization (DA) attacks using real world graph. Second, we evaluate *GAGA* under various utility metrics and compare the results with the state-of-the-art graph anonymization approaches. Finally, we compare the performance of *GAGA* with work by Zhou and Pei [29]. All the experiments were conducted on a PC running Ubuntu 16.04 with an Intel Core i7-4770 CPU running at 3.4 GHz and 23 GB RAM.

4.1 Evaluating *GAGA* against DA attacks

As discussed in Section 2, Ji et al. [11] implemented the *SecGraph* tool to conduct analysis and evaluation of existing anonymization techniques. In this subsection, we compare *GAGA* with the state-of-the-art anonymization techniques using *SecGraph* against different DA attacks.

Dataset and DA attacks. We use Facebook friendship network collected from survey participants using the Facebook app [15] consisting of 61 nodes and 270 undirected edges representing the friendship between users. We evaluate the anonymization approaches against the following five practical DA attacks: **1) Narayanan and Shmatikov [22]:** They proposed a re-identification algorithm to de-anonymize the graph based on the graph topology. Here the attacker, in

addition to having detailed information about a very small number of members of the target network, also has access to the data of another graph (a subgraph from the target graph or another social network). Thus, the power of the attack depends on the level of the attacker's access to auxiliary networks; **2) Srivatsa and Hicks [24]:** They presented an approach to re-identify the mobility traces. They used the social network data of the participating users as the auxiliary information. They used heuristic based approach on Distance Vector, Randomized Spanning Trees, and Recursive Subgraph Matching to propagate the DA; **3) Yartseva and Grossglauser [27]:** They proposed a simple percolation-based graph matching algorithm that incrementally maps every pair of node with at least r (predefined threshold) neighboring mapped pairs. They also showed that the approach used in [22] has a sharp phase transition in performance as a function of the seed set size. That is, when the seed set size is below a certain threshold, the algorithm fails almost completely. When the number of seeds exceeds the threshold, they achieve a high success rate. This is again consistent with the evaluation of [22] which shows that the power of the attack depends on how large the auxiliary networks are; **4) Korula and Lattanzi [14]:** They presented a similar approach to [27] where they use an initial set of links of users across different networks as the seed set and map a pair of users with the most number of neighboring mapped pairs; **5) Ji et al. [12]:** They proposed two DA attack frameworks, namely De-Anonymization and Adaptive De-Anonymization. The later attack is used to de-anonymize data without the knowledge of the overlap size between the anonymized data and the auxiliary data. In their attack, besides the vertices' local properties, they incorporate global properties as well.

Our evaluation methodology is basically the same as in [11]. We compare *GAGA* with the following anonymization techniques: ***Add/Del*** approach introduced in [17] which adds k randomly chosen edges followed by deletion of other k randomly chosen edges. ***Deferentially Private Graph Model*** (***DP***) proposed in [23], in which a partitioned privacy technique is employed to achieve differential privacy. ***K-Degree Anonymization*** (***KDA***) technique presented in [17], in which some edges are added to or removed from the original graph so that each vertex has at least k-1 other vertices with the same degree. ***Random Walk Anonymization*** (***RW***) approach proposed in [21], where the graph is perturbed with replacing the edges by random walk paths in order to provide link privacy. ***t-Means Clustering Algorithm*** (***t-Means Cluster***) introduced in [25], uses conventional t-Means algorithm to create clusters with size of at least k. ***Union-Split Clustering*** (***Union Cluster***) technique presented in [25], is similar to *t-Means Clustering Algorithm* while cluster centers are not chosen arbitrarily to bypass the variability in clustering results.

We present the results in Table 1. The criteria for parameters settings for each anonymization technique are the same to settings as in [11] which follows the same settings in original works. That is for *Union Cluster*, k is the size of each cluster; for *Add/Del*, f is the fraction of edges to be modified; for *KDA*, k is the anonymization parameter indicating the number of similar nodes with respect to degree; for *DP*, ε is the parameter that determines the amount of

noises that must be injected into the graph where a larger value of ε means that it is easier to identify the source of the graph structure and hence a lower level of graph privacy is preserved; for *t-Means Cluster*, t is the parameter which shows the minimum size of each cluster; for *RW*, r is the number of steps; and finally, for *GAGA*, k indicatess the number of similar nodes with respect to neighborhood structures and d shows the level of *d-neighborhood*. For DA attacks, we randomly sample a graph with probability *s=80% and s=90%* from the original graph as the auxiliary graph and then we apply the graph anonymization approaches to obtain the anonymized graphs. A larger value for s results in successfully de-anonymizing more users since with a large s, the anonymized graph and the auxiliary graph are likely to have similar structures. We also feed each DA technique 20 pre-identified seed mappings. Then we use the auxiliary graph to de-anonymize the anonymized graph. We use *Union Cluster* as the baseline for our evaluation. For each anonymization technique, *SecGraph* provides the number of successfully de-anonymized users – this number for *Union Cluster* is given in parenthesis. For rest of the techniques, Table 1 provides the factor by which number of successfully de-anonymized users is reduced in comparison to the baseline. **Note that *GAGA* is the optimal solution against all DA attacks (bold values in Table 1) as for all DA attacks *GAGA* offers the most defense – in fact the number of de-anonymized users is either 0 (perfect defense) or 1 (near perfect) of 50–57. *GAGA* (wih d=3) reduces the de-anonymization rate by at least a factor of 2.7× over the baseline. A factor of ∞ means no user has been de-anonymized successfully**. Larger values of d make *GAGA* more powerful against DA attacks. This is because each DA attack uses a combination of structural properties/semantics while each anonymization technique usually focuses on one structural property/semantic (e.g., vertex degree in *KDA* [17] or *1-neighborhood-anonymity* in [29]). However in *GAGA* we use *k(d)-neighborhood-anonymity* for any *d-neighborhood* which makes all complex neighborhoods structures similar to at least *k-1* other neighborhoods structures followed by other structural properties/semantics changes. Note that no values for *DP* and *RW* when *s=90%* are given because the anonymized graphs obtained in these two cases do not have enough edges; however, we are able to report the results for them when *s=80%*.

4.2 Evaluating *GAGA* for Utilities

Now we compare *GAGA* with the state-of-the-art anonymization techniques using *SecGraph* from the graph and application utility preservation perspective.

Dataset and utility metrics. We use DBLP co-authorship network [15] consisting of 8734 nodes and 10100 undirected edges representing the co-authorship where two authors are connected if they publish at least one paper together. We apply the same graph anonymization approaches that we used in previous subsection along with *GAGA* to anonymize the original graph and then measure how each data utility is preserved in the anonymized graph compared to the original graph. We use the following 16 popular graph and application utility metrics to measure the utility preservation:

Table 1. Comparing *GAGA*'s preservation of privacy with existing approaches introduced in [11] against five DA attacks. Using Union Cluster as the baseline, the factor by which number of de-anonymized users is reduced by each other technique is presented.

DA	s	*Union Cluster* (k=5)	*Add/Del* (f=0.23)	*KDA* (k=5)	*DP* (ε=10)	*t-Means Cluster* (t=5)	*RW* (r=2)	*GAGA* (k=5 , d=1)	*GAGA* (k=5 , d=2)	*GAGA* (k=5 , d=3)
Ji et al. [12]	0.8	1 (2 of 42)	1.1×	1.1×	1.2×	1.3×	1.8×	1.5×	2.3×	**2.7× (1 of 57)**
	0.9	1 (2 of 37)	1.1×	1.1×	-	1.4×	-	1.4×	2.2×	**3.1× (1 of 57)**
Korula and Lattanzi [14]	0.8	1 (2 of 48)	1.2×	1.2×	1.2×	1.2×	1.3×	1.5×	**∞**	**∞ (0 of 51)**
	0.9	1 (2 of 45)	0.9×	1.2×	-	1.1×	-	1.2×	2.5×	**∞ (0 of 50)**
Narayanan and Shmatikov [22]	0.8	1 (3 of 51)	0.8×	1×	1×	1.6×	1.4×	1.5×	**3.1×**	**3.1× (1 of 53)**
	0.9	1 (3 of 44)	0.8×	1×	-	1.5×	-	1.5×	3.3×	**3.6× (1 of 53)**
Srivatsa and Hicks [24]	0.8	1 (2 of 42)	1.1 ×	1.1×	1.2×	1.3×	1.6×	1.5×	1.9×	**2.7× (1 of 57)**
	0.9	1 (2 of 38)	1×	1.1×	-	1.3×	-	1.4×	1.9×	**3× (1 of 57)**
Yartseva and Grossglauser [27]	0.8	1 (4 of 52)	1.4×	1.7×	1.8×	2×	2.2×	2.1×	3.6×	**4× (1 of 52)**
	0.9	1 (4 of 44)	1.3×	1.5×	-	1.7×	-	2×	3.9×	**4.7× (1 of 52)**

Authorities Score: which is the sum of the scores of the hubs of all of the vertex predecessors. ***Betweenness Centrality:*** which indicates the centrality of a vertex. It is equal to the number of shortest paths from all vertices to all others that go through the specific vertex. ***Closeness Centrality:*** which is defined as the inverse of the average distance to all accessible vertices. ***Community Detection:*** a communication in a graph is a set of vertices where there are more connections between the members of the set than the members to the rest of the graph. *SecGraph* uses the hierarchical agglomeration algorithm introduced in [26] to measure the *Community Detection*. ***Degree:*** which indicates the degree distribution of the graph. ***Effective Diameter:*** which is the minimum number of hops in which some fraction (say, 90%) of all connected pairs of vertices can reach each other. ***EigenVector:*** let A be the adjacency matrix of a graph G, the *EigenVector* is a non-zero vector v such that $Av = \lambda v$, where λ is a scalar multiplier. ***Hubs Score:*** which is the sum of the authorities scores of all of the vertex successors. ***Infectiousness:*** which measures the number of users infected by a disease in a infectious diseases spreading model where each user transmits the disease to its neighbors with some infection rate. ***Joint Degree:*** which indicates the joint degree distribution of the graph. ***Local Clustering Coefficient:*** which quantifies how close the vertex neighbors are to being a complete graph. ***Network Constraint:*** which measures the extent to which a network is directly or indirectly concentrated in a single contact. ***Network Resilience:*** which is the number of vertices in the largest connected cluster when vertices are removed from the graph in the degree decreasing order. ***Page Rank:*** which computes the ranking of the vertices in the graph. ***Role Extraction:*** which automatically determines the underlying roles in the graph and assigns a mixed-membership of the roles to each vertex to summarize the behavior of the vertices. *SecGraph* uses the approach in [9] to measure the *Role Extraction*. ***Secure Routing:*** to address the security vulnerabilities of P2P systems, Marti et al. [19] proposed an algorithm to leverage trust relationships given by social links. *SecGraph* uses their approach to measure the *Secure Routing* utility metric.

Table 2. Comparing the utility preservation of *GAGA* with the utility preservation of existing approaches introduced in [11] with respect to various utilities.

Utility	***Add/Del*** *(f=0.06)*	***DP*** *(ε=10)*	***GAGA*** *(k=5, d=1)*	***KDA*** *(k=5)*	***RW*** *(r=2)*	***t-Means Cluster*** *(t=5)*	***Union Cluster*** *(k=5)*
Authorities Score	0.4995	0.324	**0.7079**	0.4013	0.3792	0.6773	0.6976
Betweenness Centrality	0.8256	0.762	**0.9606**	0.8459	0.8247	0.9378	0.8755
Closeness Centrality	0.9123	0.785	0.9604	0.9788	0.8612	0.9632	0.9832
Community Detection	0.3926	0.1766	0.8783	0.8747	0.2933	0.5324	0.9103
Degree	0.9877	0.9265	**0.9998**	0.9979	0.9648	0.9972	0.9989
Effective Diameter	0.9559	0.6268	**0.9632**	0.9205	1.7626	0.9394	0.9629
EigenVector	0.8562	0.4443	**0.9927**	0.6573	0.5573	0.9598	0.9909
Hubs Score	0.6844	0.2971	**0.7259**	0.5274	0.3967	0.6997	0.686
Infectiousness	0.8033	0.8675	0.8393	0.8364	0.7093	0.8513	0.8622
Joint Degree	0.7645	0.6102	**0.9875**	0.7713	0.2679	0.6832	0.7943
Local Clustering Coefficient	0.9846	0.9074	**0.9977**	0.997	0.9561	0.9909	0.9939
Network Constraint	0.9885	0.9777	0.9992	0.9999	0.987	0.9994	0.9999
Network Resilience	0.9989	0.9954	0.9997	0.9999	0.9913	0.9999	0.9999
Page Rank	0.3722	0.3323	**0.3766**	0.3681	0.3625	0.3758	0.3742
Role Extraction	0.5519	0.2271	**0.6685**	0.3134	0.2418	0.5282	0.6248
Secure Routing	1.0346	1.1149	**1.0024**	1.007	0.9571	0.9505	1.1717

Table 2 presents the results and provides the parameters that were used for each approach. Each value in the table represents one of the the following: *Cosine Similarity* in case of Authorities Score, Betweenness Centrality, Closeness Centrality, Degree, Hubs Score, Infectiousness, Joint Degree, Local Clustering Coefficient, Network Constraint, Network Resilience, Page Rank, Role Extraction, and Secure Routing; *Ratios* in case of Effective Diameter and EigenVector; and *Jaccard Similarity* in case of Community Detection between the anonymized and original graphs.

For *GAGA*, we set k=*5* and d=*1* and hence as a result, 297 edge adds and 307 edge removes (including 145 edge switches in total) have been applied to the graph. Accordingly, we set the similar parameters for other approaches so that the number of changes to the original graph can be compared with *GAGA* fairly. For example, we used the same k=*5* and t=*5* for *KDA*, *Union Cluster*, and *t-Means Cluster* accordingly. For *Add/Del*, we set f to 0.06, that is because 297 edge adds, and 307 edge removes in *GAGA* map to 307+297 edge adds/deletes for *Add/Del*. We also used a reasonable value for ε in *DP* that is the same value in original work, as we mentioned earlier larger value of ε means smaller changes to the graph so we set ε to the reasonable value of 10. For *RW*, we set r to the minimum value of 2. In general, our evaluation results are consistent with the results presented in [11]: most of the graph and application utilities can be partially or conditionally preserved with most anonymization algorithms.

Despite the fact that no anonymization scheme is optimal to preserve all utilities, note that for most of the utilities (11 out of 16 highlighted as bold values in Table 2) *GAGA* is the best approach to preserve these utilities. For some other utilities, *Union Cluster* and *KDA* have good performance. However, as we discussed in the previous subsection, *Union Cluster* and *KDA* are very vulnerable to DA attacks. This makes *GAGA*

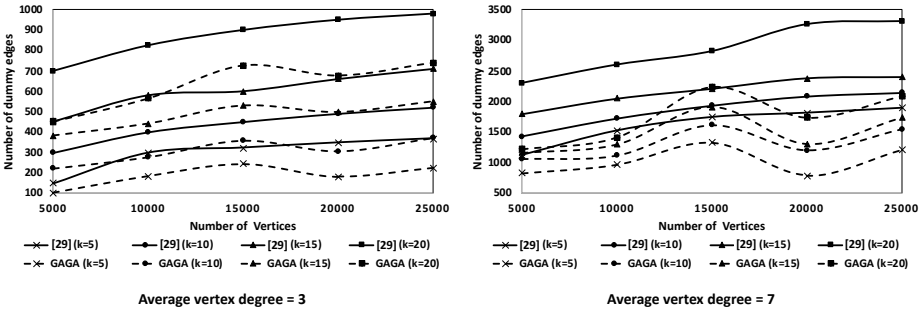

Fig. 5. Comparing the cost of *GAGA* with the cost of Zhou and Pei [29] on various data sets.

the most efficient practical approach which can preserve most of the utilities and at the same time also defend well against modern DA attacks.

4.3 *GAGA* vs. Zhou & Pei [29]

As we discussed in Section 2, Zhou and Pei [29] presented the *k-neighborhood-anonymity* model to preserve users' privacy against some neighborhood attacks. They evaluated the anonymization cost of their approach using various data sets generated by the R-MAT graph model [5]. To compare our work with Zhou and Pei's work, we used the same model with the same default parameters to generate the same data sets. Figure 5 compares the anonymization cost of *GAGA* with their work. Recall that as discussed earlier, Zhou and Pei [29] only support *1-neighborhood* and only apply edge addition to the original graph. However, *GAGA* supports any d and applies three different changes to the graph: switch, add, and remove. Therefore, to compare the cost of *GAGA* to their approach we use $d=1$ and we compute the sum of all edge additions and deletions that *GAGA* applies to the original graph. The results show that in all cases *GAGA* is far more efficient in terms of the anonymization cost (i.e., number of changes to the original graph) than Zhou and Pei's approach when obtaining the same level of privacy preservation. Notice how our approach is efficient even for denser graphs where the average vertex degree is 7 – while the number of dummy edges for Zhou and Pei varies from around 1100 to 3300, the total number of edge adds and removes applied by *GAGA* varies only from 830 to 2230.

We present the results in further detail in Table 3. The first column shows the number of vertices used to generate the graphs using R-MAT graph model. For brevity, we report only the cases of 5,000 and 25,000 vertices. The third and eighth column give the number of violating vertices along with the corresponding scenario with respect to different k values (g is the *good* scenario, b is the *bad* scenario, and u is the *ugly* scenario). We give the average degree of violating vertices in fourth and ninth column. A high average degree means that some

Table 3. *GAGA* anonymization cost on various data sets.

Num. of vertices	k	Average vertex degree=3					Average vertex degree=7				
		Num. of violating vertices (scenario)	Avg. Deg. of violating vertices	Num. of adds	Num. of removes	Avg. (GAGA cost÷Zhou and Pei [29] cost) (%)	Num. of violating vertices (scenario)	Avg. Deg. of violating vertices	Num. of adds	Num. of removes	Avg. (GAGA cost÷Zhou and Pei [29] cost) (%)
5000	5	36(g)	16	39	63	64	321(b)	24	217	614	67
25000	5	116(u)	25	97	129		700(b)	38	388	825	
5000	10	115(b)	14	103	120	72	429(b)	22	315	744	71
25000	10	178(u)	23	177	194		1009(b)	34	457	1093	
5000	15	184(b)	12	175	209	81	505(b)	21	339	827	68
25000	15	284(u)	21	241	309		1187(b)	32	528	1207	
5000	20	217(b)	11	192	261	72	553(b)	20	397	829	61
25000	20	354(u)	19	317	422		1299(b)	32	659	1421	

violating vertices have much higher degree than the graph's average degree (3 or 7) and as a result greater number of removes than adds are needed to anonymize the graph.

Note that since Zhou and Pei [29] only consider *d=1* scenario, the degree of the vertices can be considered as a good parameter to represent the structure of neighborhoods. Thus, we also present the average degree for violating vertices in the tested data sets. Since in *GAGA* we consider *k(d)-neighborhood Anonymization* for any *d-neighborhood*, degree is not a good parameter to represent the complex structure of *d*-neighborhoods. Thus, we report the number of adds, and removes (including edges switches). Finally, we compare the anonymization cost of *GAGA* with Zhou and Pei's [29] cost in the "Avg. (*GAGA cost*÷Zhou and Pei [29] cost)" column. **In all cases, our approach is more efficient. On average, our approach incurs only 69% of the cost of Zhou and Pei's approach in terms of number of changes to the original graph**.

5 Related Work

As we discussed in section 2, several graph anonymization techniques have been proposed. Casas-Roma et al. [4] compare *random-based* algorithm [8] and *k-degree-anonymity* algorithm [17] in terms of graph and risk assessment metrics and it was shown that *k-degree-anonymity* is more effective. The evaluation was limited to 3 small data sets, moreover, only 6 metrics to measure the graph utility preservation are used and no DA attacks were considered in the evaluation. The sole use of degrees in representing graphs and characterizing anonymization introduces limitations. First, it makes anonymization vulnerable to attacks that use more complex graph characteristics such as neighborhood structure of a target vertex. Second, a graph is represented by degree sequence which is not desirable since two different graphs can have same degree sequence. To overcome the limitations of *k-degree-anonymity*, Zhou and Pei [29] introduced the concept of *k-neighborhood-anonymity* [29] that considers graph structure. As we discussed, they only consider *d=1-neighborhood* which is not efficient for complex DA at-

tacks. Finally, Ji et al. [11] implemented the *SecGraph* tool to analyze existing anonymization techniques in terms of data utility and vulnerability against modern DA attacks. They conclude that it is a big challenge to effectively anonymize graphs with desired data utility preservation and without enabling adversaries to utilize these data utilities to perform modern DA attacks. Therefore in this paper, aiming to address the limitations in *k-anonymity* graph anonymization techniques, we implemented and evaluated *GAGA* that not only provides defense against modern DA attacks, but also preserves most of the utilities.

6 Conclusions

In this paper we addressed the limitations in graph anonymization techniques. We proposed, implemented, and evaluated *GAGA*, an efficient genetic algorithm for graph anonymization. Our results show that *GAGA* is highly effective and has a better trade-off between anonymization and utility preservation compared to existing techniques.

First, by applying the concept of *k(d)-neighborhood Anonymization* for any *d*, *GAGA* preserves data privacy against the modern DA attacks. Second, with the help of genetic algorithm and giving higher priority to edge switching over edge adding and removing, *GAGA* preserves the graph and application utilities. Moreover, *GAGA* gives application-oriented level of control on anonymization and utility preservation to the users/data owners via selection of k and d parameters. There are other parameters and thresholds (*GA initial population*, s, T_{pv}, T_u, etc) used in *GAGA*. These could be further tuned to obtain the optimal solutions for any graph.

Acknowledgement

This work is supported by NSF grant CCF-1617424 to the University of California Riverside.

References

1. Aggarwal, C.C., Yu, P.S.: Privacy-Preserving Data Mining: Models and Algorithms. Springer Publishing Company, Incorporated, 1 edn. (2008)
2. Atzori, M.: Weak k-anonymity: A low-distortion model for protecting privacy. In: Proceedings of the 9th International Conference on Information Security. pp. 60–71. ISC'06, Springer, Berlin, Heidelberg (2006), https://doi.org/10.1007/11836810_5
3. Bayardo, R.J., Agrawal, R.: Data privacy through optimal k-anonymization. In: 21st International Conference on Data Engineering (ICDE'05). pp. 217–228 (April 2005), https://doi.org/10.1109/ICDE.2005.42
4. Casas-Roma, J., Herrera-Joancomartí, J., Torra, V.: Comparing Random-Based and k-Anonymity-Based Algorithms for Graph Anonymization, pp. 197–209. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

5. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A Recursive Model for Graph Mining, pp. 442–446. https://doi.org/10.1137/1.9781611972740.43
6. Chester, S., Kapron, B., Ramesh, G., Srivastava, G., Thomo, A., Venkatesh, S.: Why waldo befriended the dummy? k-anonymization of social networks with pseudo-nodes. Social Network Analysis and Mining **3** (09 2012), https://doi.org/10.1007/s13278-012-0084-6
7. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence pp. 1367–1372 (Oct 2004), https://doi.org/10.1109/TPAMI.2004.75
8. Hay, M., Miklau, G., Jensen, D., Weis, P., Srivastava, S.: Anonymizing social networks. Tech. rep., SCIENCE (2007)
9. Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C., Li, L.: Rolx: Structural role extraction & mining in large graphs. In: SIGKDD. pp. 1231–1239. ACM, New York, NY, USA (2012), https://doi.org/10.1145/2339530.2339723
10. Ji, S., Li, W., Gong, N.Z., Mittal, P., Beyah, R.A.: On your social network de-anonymizablity: Quantification and large scale evaluation with seed knowledge. In: NDSS (2015)
11. Ji, S., Li, W., Mittal, P., Hu, X., Beyah, R.: Secgraph: A uniform and open-source evaluation system for graph data anonymization and de-anonymization. In: Proceedings of the 24th USENIX Conference on Security Symposium. pp. 303–318. SEC'15, USENIX Association, Berkeley, CA, USA (2015)
12. Ji, S., Li, W., Srivatsa, M., He, J.S., Beyah, R.: Structure based data de-anonymization of social networks and mobility traces. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S.M. (eds.) Information Security. ISC'14
13. Jia, J., Wang, B., Gong, N.Z.: Random walk based fake account detection in online social networks. In: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 273–284 (June 2017), https://doi.org/10.1109/DSN.2017.55
14. Korula, N., Lattanzi, S.: An efficient reconciliation algorithm for social networks. Proc. VLDB Endow. **7**(5), 377–388 (Jan 2014), https://doi.org/10.14778/2732269.2732274
15. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014)
16. Li, N., Zhang, N., Das, S.K.: Relationship privacy preservation in publishing online social networks. In: 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing. pp. 443–450 (Oct 2011). https://doi.org/10.1109/PASSAT/SocialCom.2011.191
17. Liu, K., Terzi, E.: Towards identity anonymization on graphs. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. pp. 93–106. SIGMOD '08, ACM, New York, NY, USA (2008), https://doi.org/10.1145/1376616.1376629
18. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. ACM Trans. Knowl. Discov. Data **1**(1) (Mar 2007), https://doi.org/10.1145/1217299.1217302
19. Marti, S., Ganesan, P., Garcia-molina, H.: Sprout: P2p routing with social networks. vol. 3268 (04 2004), https://doi.org/10.1007/978-3-540-30192-9_42

20. Meyerson, A., Williams, R.: On the complexity of optimal k-anonymity. In: Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 223–228. PODS '04, ACM, New York, NY, USA (2004), https://doi.org/10.1145/1055558.1055591
21. Mittal, P., Papamanthou, C., Song, D.: Preserving link privacy in social network based systems. In: NDSS (2013)
22. Narayanan, A., Shmatikov, V.: De-anonymizing social networks. In: 2009 30th IEEE Symposium on Security and Privacy. pp. 173–187 (May 2009), https://doi.org/10.1109/SP.2009.22
23. Sala, A., Zhao, X., Wilson, C., Zheng, H., Zhao, B.Y.: Sharing graphs using differentially private graph models. In: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference. pp. 81–98. IMC '11, ACM, New York, NY, USA (2011), https://doi.org/10.1145/2068816.2068825
24. Srivatsa, M., Hicks, M.: Deanonymizing mobility traces: Using social network as a side-channel. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 628–637. CCS '12, ACM, New York, NY, USA (2012), https://doi.org/10.1145/2382196.2382262
25. Thompson, B., Yao, D.: The union-split algorithm and cluster-based anonymization of social networks. In: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security. pp. 218–227. ASIACCS '09, ACM, New York, NY, USA (2009), https://doi.org/10.1145/1533057.1533088
26. Yang, J., Leskovec, J.: Overlapping community detection at scale: A nonnegative matrix factorization approach. In: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining. pp. 587–596. WSDM '13, ACM, New York, NY, USA (2013), https://doi.org/10.1145/2433396.2433471
27. Yartseva, L., Grossglauser, M.: On the performance of percolation graph matching. In: Proceedings of the First ACM Conference on Online Social Networks. pp. 119–130. COSN '13, ACM, New York, NY, USA (2013), https://doi.org/10.1145/2512938.2512952
28. Ying, X., Wu, X.: Randomizing Social Networks: a Spectrum Preserving Approach, pp. 739–750. https://doi.org/10.1137/1.9781611972788.67
29. Zhou, B., Pei, J.: Preserving privacy in social networks against neighborhood attacks. In: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering. pp. 506–515. ICDE '08, IEEE Computer Society, Washington, DC, USA (2008), https://doi.org/10.1109/ICDE.2008.4497459