Towards identity anonymization in social networks

Kenneth L. Clarkson[*] Kun Liu[†] Evimaria Terzi[‡]

1 Introduction

Social networks, online communities, peer-to-peer file sharing and telecommunication systems can be modeled as complex graphs. These graphs are of significant importance in various application domains such as marketing, psychology, epidemiology and homeland security. The management and analysis of these graphs is a recurring theme with increasing interest in the database, data mining and theory communities. Past and ongoing research in this direction has revealed interesting properties of the data and presented efficient ways of maintaining, querying and updating them. The proliferation of social networks has inevitably raised issues related to privacy preserving data analysis as illustrated in recent papers: e.g., [2, 11, 23, 18, 22].

Compared with existing anonymization and perturbation techniques of tabular data (see, *e.g.*, the survey book [1]), working with graphs and networks is much more challenging. Some aspects of graph data that enhance the challenge are the following:

- It is difficult to model the background knowledge and the capability of an attacker. Any topological structures of the graph can be exploited by the attacker to derive private information. Two nodes that are indistinguishable with respect to some structural metrics may be distinguishable by other metrics.

- It is difficult to quantify the information loss. A graph contains rich information but there is no standard way to quantify the information loss incurred by the changes of its nodes and edges.

- It is even difficult to devise graph-modification algorithms that balance the goals of preserving privacy with the utility of the data. Although in tabular data where each tuple can be viewed as an independent sample from some distribution, the nodes and edges in a graph are all related to

[*]IBM Almaden Research Center. Email:`clarkson@us.ibm.com`

[†]Yahoo! Labs. Email: `kun@yahoo-inc.com`. This work was done when the author was at IBM Almaden Research Center.

[‡]Boston University Email: `evimaria@cs.bu.edu`. This work was done when the author was at IBM Almaden Research Center.

each other. Therefore, the impact of a single change of an edge or a node can spread across the whole network.

Recall that in a social network, nodes correspond to individuals or other social entities, and edges correspond to social relationships between them. The privacy breaches in social network data can be grouped to three categories: 1) *identity disclosure*: the identity of the individual who is associated with the node is revealed; 2) *link disclosure*: sensitive relationships between two individuals are disclosed; and 3) *content disclosure*: the privacy of the data associated with each node is breached, *e.g.*, the email message sent and/or received by the individuals in a email communication graph. A perfect privacy-protection system should consider all of these issues. However, protecting against each of the above breaches may require different techniques. For example, for content disclosure, standard privacy-preserving data mining techniques [1], such as data perturbation and k-anonymization can help. For link disclosure, the various techniques studied by the link-mining community [9, 23] can be useful.

The techniques presented in this chapter aim towards protection of *identity disclosure* of individuals in a social network. In their recent work, Backstrom *et. al.* [2] point out that the simple technique of anonymizing graphs by removing the identities of the nodes before publishing the actual graph does not always guarantee privacy. It is shown in [2] that there exist adversaries that can infer, in polynomial time, the identity of the nodes using graph-isomorphism algorithms designed for a restricted class of graphs. However, the problem of designing techniques that could protect individuals' privacy has not been addressed in [2].

Motivated by [2], we focus our attention on protecting the identities of individuals from adversaries that have prior knowledge of the degrees of some nodes. As an example, consider a social-network graph and an adversary who connects to a node with a degree x that is unusually high. It would be rather unexpected for there to be another node of degree exactly x. Now assume a "naive" privacy-protection strategy that simply removes the names of the nodes in the network before releasing the graph. Then it is obvious that if degree x is unique in the graph, then the adversary can re-identify his high-degree neighbor by simply asking the query "Find all nodes with degree x". In this chapter we present methods that prevent this. For that we describe a k-anonymity notion for graphs that is similar to the k-anonymity notion introduced for tabular data [19]. In other words, this chapter describes a methodology for answering the following question: *How can a graph be minimally modified to protect the identity of each individual involved?*

The material presented here is an extension of our earlier work presented in [15]. Since the publication of our original paper, several other methods have been developed for identity anonymization on graphs. We give a summary of these methods in the next section.

2 Related work

Since the introduction of the concept of anonymity in databases [19], there has been increasing interest in the database community in studying the complexity of the problem and proposing algorithms for anonymizing data records under different anonymization models [4, 16, 17]. Though much attention has been given to the anonymization of tabular data, the privacy issues of graphs and social networks, and the notion of anonymization of graphs, have only been recently touched.

Backstrom *et. al.* [2] were the first to study attacks against simple privacy-preserving methods for social networks. In their seminal paper ([2]) they show that simply removing the identifiers of the nodes does not always guarantee privacy. Adversaries can infer the identity of the nodes by solving a set of restricted isomorphism problems based on the uniqueness of small random subgraphs embedded in a network.

As we have already discussed in the introduction, this observation gave rise to three types of risks of challenges in privacy-preserving data mining methods: how to prevent identity disclosure, link disclosure, and content disclosure in social networks. To combat these challenges, several authors have recently developed different types of privacy models, adversaries, and graph-modification algorithms. Unfortunately, none of the work is likely to solve all the problems in one shot. Protecting against each kind of privacy breach may require different techniques, or a combination of them. Below we give a list of papers dealing with these challenges. This list should be conceived as indicative rather than complete. For a more thorough review of the literature, see [24].

Apart from our work in [15] which we describe here, the problem of identity-anonymization in social networks has been studied in [11, 18].

Hay *et. al.* [11] observe that the structural similarity of the nodes in the graph determines the extent to which an individual in the network can be distinguished from others. Based on the notion of k-anonymity [19], Hay et al. [11] proposed a scheme of anonymity through structural similarity. Vertices that look structurally similar may be indistinguishable to an adversary. A strong form of structural similarity between vertices is automorphism equivalence. The anonymization technique proposed in [11] is a node-clustering approach. It generalizes an input network by grouping vertices into partitions and publishing the number of vertices in each partition along with the densities of edges within and across partitions. Data analysts can still use the anonymized graphs to study macro-properties of the original graph.

Pei and Zhou in [18] consider yet another definition of graph anonymity: a graph is k-anonymous if for every node there exists at least $k-1$ other nodes that share isomorphic neighborhoods; in this case the neighborhood of a node is defined by its immediate neighbors and the connections between them. This definition of anonymity in graphs is different from ours. In a sense it is a more strict one.

Protection of links between individual graph entities has been studied in [13, 23, 22]. Zheleva and Getoor [23] consider the problem of protecting sensitive relationships among the individuals in the anonymized social network. This is closely related to the link-prediction problem that has been widely studied in the link-mining community [9]. In [23] simple edge-deletion and node-merging algorithms are proposed to reduce the risk of sensitive link disclosure.

Sensitive link and relationship protection is also discussed by Ying and Wu [22]. They study how anonymization algorithms that are based on randomly adding and removing edges change certain graph properties. More specifically, they focus on the change caused in the eigenvalues (spectrum) of the network. The authors additionally explore how the randomized network can be exploited by an adversary to gain knowledge about the existence of certain links.

Korolova *et al.*[13] considered the problem where an attacker wants to derive the link structure of the entire network by collecting the neighborhood information of some compromised users, who are either bribed or whose accounts are broken into by the attacker. Analysis shows that the number of users needed to be compromised in order to cover a constant fraction of the entire network drops exponentially with increase in a lookahead parameter ℓ. Parameter ℓ determines if a registered user can see all of the links and nodes within distance ℓ from him.

Content disclosure is normally an issue when the private data associated with a user on the network is disclosed to others. A very interesting example recently arose from Facebook's "Beacon" service, a "social ads" system where your own expressed brand preferences and Internet browsing habits, and even your very identity, are used to market goods and services to you and your friends. For example, adding the latest season of LOST to your queue on Blockbuster.com might result in Facebook placing an ad for Blockbuster straight on your friends' news feeds. This helps Facebook and its partners (Blockbuster in this example) make money because, as Facebook's CEO Mark Zuckerberg extols, "nothing influences a person more than a recommendation from a trusted friend." This may be fine in some situations, but there may be some things that one is not prepared to share with the entire world. From the users' perspective, they want to ask how to avoid the disclosure of their personal private information while still enjoying the benefit of social advertisement. Companies on the other hand want to assure the users that their privacy is not compromised while doing social advertisement. Privacy concerns regarding content disclosure exist in other application scenarios such as social recommendation, etc. Protecting against this kind of disclosure is an important research and engineering problem. However, the work in the literature thus far does not take into account how graph structures affect the content disclosure; they rather focus on standard data perturbation and anonymization for tabular data.

3 Problem definition

Let $G(V, E)$ be a simple graph; V is a set of nodes and E the set of edges in G. We use $\mathbf{d}_G$ to denote the *degree sequence* of G. That is, $\mathbf{d}_G$ is a vector of size $n = |V|$ such that $\mathbf{d}_G(i)$ is the degree of the i-th node of G. Throughout the chapter, we use $\mathbf{d}(i)$, $\mathbf{d}(v_i)$ and $\mathbf{d}_G(i)$ interchangeably to denote the degree of node $v_i \in V$. When the graph is clear from the context we drop the subscript and use $\mathbf{d}(i)$ instead. Without loss of generality, we also assume that entries in $\mathbf{d}$ are in nonincreasing order, that is, $\mathbf{d}(1) \geq \mathbf{d}(2) \geq \ldots \geq \mathbf{d}(n)$. Additionally, for $i < j$ we use $\mathbf{d}[i, j]$ to denote the subsequence of $\mathbf{d}$ that contains elements $i, i+1, \ldots, j-1, j$.

Before defining the notion of a *k-degree anonymous graph*, we first define the notion of a k-anonymous vector of integers.

Definition 1 *A vector of integers* $\mathbf{v}$ *is k-anonymous if every distinct value in* $\mathbf{v}$ *appears at least k times.*

For example, vector $\mathbf{v} = [5, 5, 3, 3, 2, 2, 2]$ is 2-anonymous.

Definition 2 *A graph $G(V, E)$ is k-degree anonymous if its degree sequence* $\mathbf{d}_G$ *is k-anonymous.*

Alternatively, Definition 2 implies that for every node $v \in V$ there exists at least $k-1$ other nodes that have the same degree as v. This property prevents the re-identification of individuals by adversaries with *a priori* knowledge of the degree of certain nodes. This echoes the observation made by Hay *et. al.* [12].

Figure 1 shows two examples of degree-anonymous graphs. In the graph on the left, all three nodes have the same degree and thus the graph is 3-degree anonymous. Similarly, the graph on the right is 2-degree anonymous since there are two nodes with degree 1 and four nodes with degree 2.

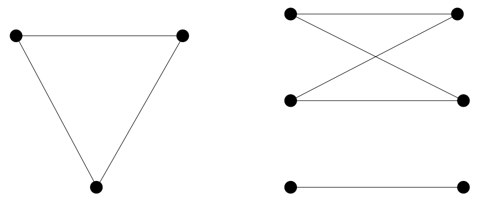


Figure 1: Examples of a 3-degree anonymous graph (left) and a 2-degree anonymous graph (right).

Degree anonymity has the following monotonicity property.

Proposition 1 *If a graph $G(V, E)$ is k_1-degree anonymous, then it is also k_2-degree anonymous, for every $k_2 \leq k_1$.*

We use the definitions above to define the Graph Anonymization problem. The input to the problem is a simple graph $G(V, E)$ and an integer k. The requirement is to use a set of graph-modification operations on G in order to construct a k-degree anonymous graph $\widehat{G}(\widehat{V}, \widehat{E})$ that is structurally similar to G. We require that the output graph is over the same set of nodes as the original graph, that is, $\widehat{V} = V$. Thus, we focus on two graph-modification operations:

addition and deletion of edges. Given two graphs $G(V,E)$ and $\widehat{G}(\widehat{V},\widehat{E})$ defined over the same set of nodes ($\widehat{V}=V$) we measure their structural similarity using two metrics: the *degree-anonymization cost* and the *structural cost.* The formal definitions of these metrics are given below.

Definition 3 *For two graphs $G(V,E)$ and $\widehat{G}(\widehat{V},\widehat{E})$ with degree sequences $\mathbf{d}$ and $\widehat{\mathbf{d}}$ defined over the same set of nodes ($V=\widehat{V}$), we define the* degree-anonymization cost *between G and $\widehat{G}$ to be the L_1-norm of the difference of their degree sequences. That is,*

$$\textsc{Da}\left(\widehat{\mathbf{d}},\mathbf{d}\right) \quad := \quad L_1\left(\widehat{\mathbf{d}}-\mathbf{d}\right) := \sum_i \left|\widehat{\mathbf{d}}(i)-\mathbf{d}(i)\right|. \tag{1}$$

Definition 4 *For two graphs $G(V,E)$ and $\widehat{G}(\widehat{V},\widehat{E})$ defined over the same set of nodes ($V=\widehat{V}$), we define the* structural difference *between G and $\widehat{G}$ to be the symmetric difference of their sets of edges. That is,*

$$\Delta\left(\widehat{G},G\right) \quad := \quad \left|\widehat{E}\setminus E\right| + \left|E\setminus\widehat{E}\right|. \tag{2}$$

Given the above definitions we define the Graph Anonymization problem as follows.

Problem 1 (Graph Anonymization) *Given a graph $G(V,E)$ with degree sequence $\mathbf{d}$, and an integer k, find a k-degree anonymous graph $\widehat{G}(V,\widehat{E})$ with degree sequence $\widehat{\mathbf{d}}$ such that (a) $\textsc{Da}\left(\widehat{\mathbf{d}},\mathbf{d}\right)$ is minimized and (b) $\Delta\left(\widehat{G},G\right)$ is also minimized.*

Note that the above problem definition has two optimization objectives: it requires both the degree-anonymization as well as the structural cost of the anonymization to be minimized. We first show that a solution to the Graph Anonymization problem that is optimal with respect to the one objective is not necessarily optimal with respect to the other. This is illustrated in the following observation.

Observation 1 *Consider graph $G\left(V,E\right)$ and $G'\left(V',E'\right)$ with $V=V'=\{w,x,y,z\}$, $E=\{(w,x),(y,z)\}$ and $E'=\{(w,z),(x,y)\}$. One can verify that $\textsc{Da}\left(\mathbf{d}_G,\mathbf{d}_{G'}\right)=0$ while $\Delta\left(G,G'\right)=4$. At the same time it is easy to construct another graph $G''\left(V'',E''\right)$ with $V''=\{w,x,y,z\}$ and $E''=\{(w,x),(y,x)\}$ such that $\Delta\left(G,G''\right)=2<\Delta\left(G,G'\right)$. At the same time $\textsc{Da}\left(\mathbf{d}_G,\mathbf{d}_{G''}\right)=2$.*

One way of dealing with problems that have more than objective functions is to define the problem as a multiple-objective optimization problem, where the goal is to *simultaneously* optimize both the objectives. In this case, the situation can arise that two graphs $\widehat{G}$ and $\widehat{G}'$ are *incomparable*, e.g., $\textsc{Da}\left(\mathbf{d}_{\widehat{G}},\mathbf{d}_G\right)<$

$\text{DA}\left(\mathbf{d}_{\widehat{G}'}, \mathbf{d}_G\right)$, but $\Delta\left(\widehat{G}, G\right) > \Delta\left(\widehat{G}', G\right)$. In these cases, all one can hope for are *Pareto-optimal solutions* [5]. A graph $\widehat{G}$ is a Pareto-optimal solution if there does not exist another graph $\widehat{G}'$ that is at least as good as $\widehat{G}$ in both objectives, and strictly better than $\widehat{G}$ in at least one of the two. The problem of finding all (approximate) Pareto-optimal solutions has been studied in [7].

In our case, we take a different approach: we prioritize our objectives. That is, we first focus on minimizing degree-anonymization cost; then among all those graphs that have the same value of DA() cost, we try to pick the one with the minimum $\Delta()$ cost.

Note that the GRAPH ANONYMIZATION problem always has a *feasible* solution. For example, all edges not present in the input graph can be added. In this way, the graph becomes complete and all nodes have the same degree; thus, any degree-anonymity requirement is satisfied (due to Proposition 1).

3.1 Restriction to edge additions

Problem 1 allows both for edge-addition as well as edge-deletion modification operations. For the case where we only focus our attention on edge additions, the anonymized graph $\widehat{G}$ will be a *supergraph* of the original graph.

In this case minimizing $\text{DA}\left(\mathbf{d}_{\widehat{G}}, \mathbf{d}_G\right)$ is equivalent to minimizing $\Delta\left(\widehat{G}, G\right)$, because in the case of edge additions,

$$\begin{aligned}\Delta\left(\widehat{G}, G\right) &= |\widehat{E} \setminus E| = |\widehat{E}| - |E| \\ &= \frac{1}{2} L_1\left(\widehat{\mathbf{d}} - \mathbf{d}\right) = \frac{1}{2}\text{DA}\left(\mathbf{d}_{\widehat{G}}, \mathbf{d}_G\right).\end{aligned}$$

It is obvious that the case where only edge-deletion operations are allowed is equivalent, because edge deletions can be considered as edge additions in the complement of the input graph.

4 Overview of the Approach

We propose a two-step approach for the GRAPH ANONYMIZATION problem. For an input graph $G(V, E)$ with degree sequence $\mathbf{d}$ and an integer k, we proceed as follows:

1. First, starting from $\mathbf{d}$, we construct a new degree sequence $\widehat{\mathbf{d}}$ that is k-anonymous and such that the degree-anonymization cost $\text{DA}(\widehat{\mathbf{d}}, \mathbf{d})$ is minimized.
2. Given the new degree sequence $\widehat{\mathbf{d}}$, we then construct a graph $\widehat{G}(V, \widehat{E})$ such that $\Delta\left(\widehat{G}, G\right)$ is minimized.

These two steps give rise to two problems, which we formally define and solve in subsequent sections. Performing step 1 translates into solving the DEGREE ANONYMIZATION problem defined as follows.

Problem 2 (DEGREE ANONYMIZATION) *Given* $\mathbf{d}$*, the degree sequence of graph* $G(V,E)$*, and an integer* k *construct a* k*-anonymous sequence* $\widehat{\mathbf{d}}$ *such that* $\text{DA}\left(\widehat{\mathbf{d}},\mathbf{d}\right) = L_1(\widehat{\mathbf{d}}-\mathbf{d})$ *is minimized.*

Similarly, performing step 2 translates into solving the GRAPH CONSTRUCTION problem that we define below.

Problem 3 (GRAPH CONSTRUCTION) *Given a graph* $G(V,E)$ *and a* k*-anonymous degree sequence* $\widehat{\mathbf{d}}$*, construct graph* $\widehat{G}(V,\widehat{E})$ *such that* $\mathbf{d}_{\widehat{G}} = \widehat{\mathbf{d}}$ *and* $\Delta\left(\widehat{G},G\right)$ *is minimized.*

In the next sections we develop algorithms for solving Problems 2 and 3.

5 Degree Anonymization

In this section we give algorithms for solving the DEGREE ANONYMIZATION problem. Given the degree sequence $\mathbf{d}$ of the original input graph $G(V,E)$, the algorithms output a k-anonymous degree sequence $\widehat{\mathbf{d}}$ such that the degree-anonymization cost $\text{DA}(\mathbf{d}) = L_1(\widehat{\mathbf{d}}-\mathbf{d})$ is minimized.

We first give a dynamic-programming algorithm (`DP`) that solves the DEGREE ANONYMIZATION problem optimally in time $O(n^2)$. Then, we show how to modify it to achieve $O(nk)$ running time, and finally, for the restricted case of edge additions or edge deletions only, $O(n)$ time.

Given a (sorted) input degree sequence $\mathbf{d}$, let $\text{DA}\left(\mathbf{d}[1,i]\right)$ be the degree-anonymization cost of subsequence $\mathbf{d}[1,i]$. Additionally, let $I\left(\mathbf{d}[i,j]\right)$ be the degree anonymization cost when all nodes $i, i+1, \ldots, j$ are put in the same anonymized group. Alternatively, this is the cost of assigning to all nodes $\{i,\ldots,j\}$ the same degree, d^*. That is,

$$I\left(\mathbf{d}[i,j]\right) = \sum_{\ell=i}^{j} |d^* - \mathbf{d}(\ell)|,$$

where d^* is the (integer) degree with property

$$d^* = \arg\min_{d} \sum_{\ell=i}^{j} |d - \mathbf{d}(\ell)|.$$

From [14] we know that d^* is the median of the values $\{\mathbf{d}(i),\ldots\mathbf{d}(j)\}$, and therefore given i and j, computing $I\left(\mathbf{d}[i,j]\right)$ can be done optimally in $O(j-i)$ time (see [6] for details).

We now construct a set of dynamic-programming equations that solve the DEGREE ANONYMIZATION problem. For $i < 2k$, let

$$C\left(i\right) \quad := \quad I\left(\mathbf{d}[1,i]\right). \tag{3}$$

For $i \geq 2k$, let

$$C(i) := \min \Big\{ I(\mathbf{d}[1,i]), \min_{k \leq t \leq i-k} \big\{ C(t) + I(\mathbf{d}[t+1,i]) \big\} \Big\}. \tag{4}$$

When $i < 2k$, it is impossible to construct two different anonymized groups each of size k. As a result, the optimal degree anonymization of nodes $1, \ldots, i$ consists of a single group.

When $i \geq 2k$, the degree-anonymization cost for the subsequence $\mathbf{d}[1,i]$ is the optimal degree-anonymization cost of the subsequence $\mathbf{d}[1,t]$, plus the anonymization cost incurred by putting all nodes $t+1, ..., i$ in the same group (provided that this group is of size k or larger). The range of variable t, as defined in Equation (4), is restricted so that all groups examined, including the first and last ones, are of size at least k.

Running time of the `DP` algorithm. For an input degree sequence of size n, the running time of the `DP` algorithm that implements Recursions (3) and (4) is $O(n^2)$; first, the values of $I(\mathbf{d}[i,j])$ for all $i < j$ can be computed in an $O(n^2)$ preprocessing step. Then, for every i the algorithm goes through at most $n - 2k + 1$ different values of t for evaluating the Recursion (4). Since there are n different values of i, the total running time is $O(n^2)$.

In fact the running time of the `DP` algorithm can further improve from $O(n^2)$ to $O(nk)$. The core idea for this speedup lies in the following simple observation: *no anonymous group should be of size larger than $2k-1$. If any group is larger than or equal to $2k$, it can be broken into two subgroups with equal or lower overall degree-anonymization cost.* Using this observation, the preprocessing step that computes the values of $I(\mathbf{d}[i,j])$, does not have to consider all the combinations of (i,j) pairs, but for every i consider only j's such that $k \leq j - i + 1 \leq 2k - 1$. Thus, the running time for this step drops to $O(nk)$.

Similarly, for every i, we do not have to consider all t's in the range $k \leq t \leq i - k$ as in Recursion (4), but only t's in the range $\max\{k, i-2k+1\} \leq t \leq i-k$. Therefore, Recursion (4) can be replaced by

$$C(i) := \min_{t \in S_i} \big\{ C(t) + I(\mathbf{d}[t+1,i]) \big\}, \tag{5}$$

where

$$S_i := \{ t \mid \max\{k, i-2k+1\} \leq t \leq i-k \}.$$

For this range of values of t we guarantee that the first group has size at least k, and the last one has size between k and $2k-1$. Therefore, for every i the algorithm goes through at most k different values of t for evaluating the new recursion. Since there are $O(n)$ different values of i, the overall running time of the `DP` algorithm is $O(nk)$.

Therefore, we have the following.

Theorem 1 *Problem 2 can be solved in polynomial time using the `DP` algorithm described above. The running time of the `DP` algorithm when the degrees of the nodes can either increase or decrease is $O(nk)$.*

5.1 Restriction to edge additions

Again, if we restrict our attention to Problem 1 where only edge additions are allowed, then the degrees of the nodes can only increase in the Degree Anonymization problem. That is, if $\mathbf{d}$ is the original sequence and $\widehat{\mathbf{d}}$ is the k-anonymous degree sequence, then for every $1 \leq i \leq n$ we have that $\widehat{\mathbf{d}}(i) \geq \mathbf{d}(i)$. In this case, the `DP` algorithm described in the previous section can solve the Degree Anonymization problem. The only difference is in the evaluation of cost $I(\mathbf{d}[i,j])$ that corresponds to the L_1 cost of putting all nodes $i, i+1, \ldots, j$ in the same anonymized group. Note that the indices correspond to the ordering of the nodes in nonincreasing order of their degree in $\mathbf{d}$. Therefore, if the degrees of the nodes can only increase, every group will be assigned the degree of the node with the highest degree. That is,

$$I(\mathbf{d}[i,j]) \quad = \quad \sum_{\ell=i}^{j}(\mathbf{d}(i) - \mathbf{d}(\ell)). \tag{6}$$

In this case, the running time of the `DP` algorithm can be further improved as follows.

Letting $f_i(t){:=}C(t) + I(\mathbf{d}[t+1,i])$, recursion (5) is

$$C(i){:=}\min_{t \in S_i} f_i(t).$$

For given i, if $f_i(s) \geq f_i(t)$ for all $t \in S_i$ with $t > s$, then plainly $f_i(s)$ is greater than or equal to the minimum of $f_i(t)$, taken over *all* $t \in S_i$. So it is enough to have the values of $f_i(s)$, for $s \in S_i$ such that $f_i(s) < f_i(t)$ for all $t \in S_i$ with $t > s$. Consider now such s and the next largest such s'. We have $f_i(s') - f_i(s) > 0$, which suggests $f_{i+1}(s') - f_{i+1}(s) > 0$, but is this true, and what happens as i increases further? The next lemma delimits the possibilities.

Lemma 1 *For fixed s and s' with $s' > s$, let $F(i) := f_i(s') - f_i(s)$. Then*

$$F(i+1) - F(i) = \mathbf{d}(s'+1) - \mathbf{d}(s+1) \leq 0.$$

This implies the following.

- *Suppose $\mathbf{d}(s'+1) = \mathbf{d}(s+1)$. Then $F(i') = F(i)$ for all $i' \geq i$.*
- *Suppose $\mathbf{d}(s'+1) < \mathbf{d}(s+1)$. Then $F(i) \leq 0$ implies $F(i') \leq 0$ for all $i' \geq i$. If $F(i) > 0$, then $F(i') > 0$ for i' with $i' < i + F(i)/(\mathbf{d}(s+1) - \mathbf{d}(s'+1))$, and $F(i') \leq 0$ for $i' \geq i + F(i)/(\mathbf{d}(s+1) - \mathbf{d}(s'+1))$.*

Proof. The expression for $F(i+1) - F(i)$ follows from expanding out the definitions and manipulating, as follows. We have

$$\begin{aligned} f_{i+1}(s) - f_i(s) &= C(\mathbf{d}[1,s]) + I(\mathbf{d}[s+1,i+1]) - [C(\mathbf{d}[1,s]) + I(\mathbf{d}[s+1,i])] \\ &= I(\mathbf{d}[s+1,i+1]) - I(\mathbf{d}[s+1,i]) \\ &= \sum_{s+1 \leq \ell \leq i+1} (\mathbf{d}(s+1) - \mathbf{d}(\ell)) - \sum_{s+1 \leq \ell \leq i} (\mathbf{d}(s+1) - \mathbf{d}(\ell)) \\ &= \mathbf{d}(s+1) - \mathbf{d}(i+1), \end{aligned}$$

and so

$$\begin{aligned}F(i+1)-F(i) &= f_{i+1}(s')-f_{i+1}(s)-(f_i(s')-f_i(s))\\ &= f_{i+1}(s')-f_i(s')-(f_{i+1}(s)-f_i(s))\\ &= \mathbf{d}(s'+1)-\mathbf{d}(i+1)-(\mathbf{d}(s+1)-\mathbf{d}(i+1))\\ &= \mathbf{d}(s'+1)-\mathbf{d}(s+1),\end{aligned}$$

which proves the first claim. From this if follows immediately that $F(i')-F(i) = (i'-i)(\mathbf{d}(s'+1)-\mathbf{d}(s+1))$, implying the remaining claims. $\square$

The discussion just before the lemma says that to compute $\min_{t\in S_i} f_i(t)$ for increasing values of i, it is enough to be able to maintain, as i goes from 1 to n, the list of indices

$$L_i := \{s \in S_i \mid f_i(s) < \min_{s<t\in S_i} f_i(t)\}. \tag{7}$$

Specifically, L_i is represented as a doubly linked list. How does this list change as i increases? If $s \notin L_i$, then $f_i(s) \geq f_i(t)$ for some $t > s$, and so by the lemma, $f_{i'}(s) \geq f_{i'}(t)$ for $i' \geq i$; that is, if s is not included now, it will not be included later.

If L_i has been maintained up to time i, then its first entry is sufficient to allow the computation of (4): for given i and $s < i$, the value of $f_i(s)$ can be computed in constant time, using previously computed values stored in array C and the prefix sums $\sum_{\ell<i}\mathbf{d}(\ell)$.

To maintain the list, we will consider the "life cycle" of an entry in it. Such an entry s is "born" at time $i = s+k$, when $i-k$ is added to the end of L_i. An entry can "die," that is, no longer qualify to be in L_i, in a few ways.

An entry can die when it gets too old, namely, when $s < \max\{k, i-2k+1\}$, and so is no longer in S_i.

An entry can also die when a new entry is added whose f value is less than or equal to its f value; here if $s \in L_i$ dies, all $t > s$ in L_i must also die, since they have larger $f_i(t)$ values. Thus when $i-k$ is added, the entries $s \in L_i$ can be examined, in decreasing order, to check if $f_i(i-k) < f_i(s)$; as such s are found, they are deleted from L_i.

We will also make an entry die in one other way: if $f_i(s)$ and $f_i(s')$ are consecutive entries in L_i, so that $s < s'$ and $f_i(s) < f_i(s')$, then from the lemma, there is a future time $i' = D(s, s')$ at which $f_{i'}(s) \geq f_{i'}(s')$, so that s should die. We will maintain that, for each consecutive pair s, s' of entries in L_i, there is a "death notice" for s, at time $D(s, s')$. At that time, when the "death notice" is processed, the entry for s is removed from L_i (if it hasn't already died), and a death notice is added for the former neighbor $s'' < s$, whose new rightward neighbor is the former rightward neighbor of s. The death notice $D(s, s')$ includes a pointer to the list node for s, which has a backpointer to the death notice, so that if the node for s is removed for other reasons, the death notice is removed also.

When an entry dies, $O(1)$ work is done for it, including the generation of at most one new death notice. The processing of that death notice, in the future, requires $O(1)$ work, so the life cycle of a node requires $O(1)$ work.

We have proven the following theorem.

Theorem 2 *Problem 2 for the restricted case of edge additions (or deletions) operations, where the degrees of the nodes can only increase (or decrease) can be solved optimally using the* `DP` *algorithm described above in time $O(n)$.*

6 Graph Construction

In this section we present algorithms for solving the Graph Construction problem. Given the original graph $G(V, E)$ and the desired k-anonymous degree sequence $\widehat{\mathbf{d}}$ output by the `DP` algorithm, we construct a k-degree anonymous graph $\widehat{G}(V, \widehat{E})$ such that $\Delta\left(\widehat{G}, G\right)$ is minimized.

6.1 Basics on Realizability of Degree Sequences

Before giving the actual algorithms for the Graph Construction problem, we first present some known facts about the realizability of degree sequences for simple graphs. Later on, we extend some of these results in our own problem setting.

Definition 5 *A degree sequence $\mathbf{d}$, with $\mathbf{d}(1) \geq, \ldots, \geq \mathbf{d}(n)$ is called* realizable *if and only if there exists a simple graph whose nodes have precisely this sequence of degrees.*

Erdös and Gallai [8] have stated the following *necessary* and *sufficient* condition for a degree sequence to be realizable.

Lemma 2 *([8]) A degree sequence $\mathbf{d}$ with $\mathbf{d}(1) \geq \ldots \geq \mathbf{d}(n)$ and $\sum_i \mathbf{d}(i)$ even, is realizable if and only if for every $1 \leq \ell \leq n-1$ it holds that*

$$\sum_{i=1}^{\ell} \mathbf{d}(i) \quad \leq \quad \ell(\ell-1) + \sum_{i=\ell+1}^{n} \min\{\ell, \mathbf{d}(i)\} \tag{8}$$

Informally, Lemma 2 states that for each subset of the ℓ highest-degree nodes, the degrees of these nodes can be "absorbed" within the nodes and the outside degrees. The proof of Lemma 2 is inductive ([10]) and it provides a natural construction algorithm, which we call `ConstructGraph` (see Algorithm 1 for the pseudocode).

The `ConstructGraph` algorithm takes as input the desired degree sequence $\mathbf{d}$ and outputs a graph with exactly this degree sequence, if such graph exists. Otherwise it outputs a "No" if such graph does not exist. The algorithm is iterative and in each step it maintains the residual degrees of vertices. In each iteration it picks an arbitrary node v and adds edges from v to $\mathbf{d}(v)$ nodes of *highest* residual degree, where $\mathbf{d}(v)$ is the residual degree of v. The residual degrees of these $\mathbf{d}(v)$ nodes are decreased by one. If the algorithm terminates and outputs a graph, then this graph has the desired degree sequence. If at some point the algorithm cannot make the required number of connections for a specific node, then it outputs "No" meaning that the input degree sequence is not realizable.

Note that the `ConstructGraph` algorithm is an *oracle* for the realizability of a given degree sequence; if the algorithm outputs "No", then this means that there does not exist a simple graph with the desired degree sequence.

Algorithm 1 The `ConstructGraph` algorithm.

```
    Input: A degree sequence $\mathbf{d}$ of length $n$.
    Output: A graph $G(V, E)$ with nodes having degree sequence $\mathbf{d}$ or "No" if
    the input sequence is not realizable.
 1: $V \leftarrow \{1, \ldots, n\}$, $E \leftarrow \emptyset$
 2: if $\sum_i \mathbf{d}(i)$ is odd then
 3:     Halt and return "No"
 4: while 1 do
 5:     if there exists $\mathbf{d}(i)$ such that $\mathbf{d}(i) < 0$ then
 6:         Halt and return "No"
 7:     if the sequence $\mathbf{d}$ are all zeros then
 8:         Halt and return $G(V, E)$
 9:     Pick a random node $v$ with $\mathbf{d}(v) > 0$
10:     $\mathbf{d}(v) \leftarrow 0$
11:     $V_{\mathbf{d}(v)} \leftarrow$ the $\mathbf{d}(v)$-highest entries in $\mathbf{d}$ (other than $v$)
12:     for each node $w \in V_{\mathbf{d}(v)}$ do
13:         $E \leftarrow E \cup (v, w)$
14:         $\mathbf{d}(w) \leftarrow \mathbf{d}(w) - 1$
```

Running time of the `ConstructGraph` algorithm: If n is the number of nodes in the graph and $d_{\max} = \max_i \mathbf{d}(i)$, then the running time of the `ConstructGraph` algorithm is $O(nd_{\max})$. This running time can be achieved by keeping an array A of size $d_{\max}$ such that $A[\mathbf{d}(i)]$ keeps a hash table of all the nodes of degree $\mathbf{d}(i)$. Updates to this array (degree changes and node deletions) can be done in constant time. For every node i at most $d_{\max}$ constant-time operations are required. Since there are n nodes the running time of the algorithm is $O(nd_{\max})$. In the worst case, $d_{\max}$ can be of order $O(n)$, and in this case the running time of the `ConstructGraph` algorithm is quadratic. In practice, $d_{\max}$ is much less than n, which makes the algorithm very efficient in practical settings.

Note that the random node in Step 9 of Algorithm 1 can be replaced by either the current highest-degree node or the current lowest-degree node. When we start with higher degree nodes, we get topologies that have very dense cores, while when start with lower degree nodes, we get topologies with very sparse cores. A random pick is a balance between the two extremes. The running time is not affected by this choice, due to the data structure A.

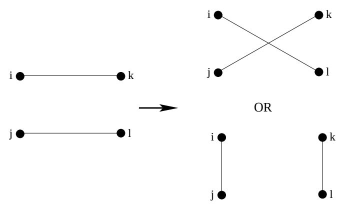


Figure 2: The swap transformation.

6.2 The Greedy_Swap algorithm

Let $\widehat{\mathbf{d}}$ be a k-anonymous degree sequence output by DP algorithm. Let us additionally assume for now, that $\widehat{\mathbf{d}}$ is realizable so that the ConstructGraph algorithm with input $\widehat{\mathbf{d}}$, outputs a simple graph $\widehat{G}_0(V, \widehat{E}_0)$ with degree sequence exactly $\widehat{\mathbf{d}}$. Although $\widehat{G}_0$ is k-degree anonymous, its structure may be quite different from the original graph $G(V, E)$. The Greedy_Swap algorithm is a greedy heuristic that given $\widehat{G}_0$ and G, it transforms $\widehat{G}_0$ into $\widehat{G}(V, \widehat{E})$ with degree sequence $\mathbf{d}_{\widehat{G}} = \widehat{\mathbf{d}} = \mathbf{d}_{\widehat{G}_0}$ such that $\Delta\left(\widehat{G}, G\right)$ is minimized.

At every step i, the graph $\widehat{G}_{i-1}(V, \widehat{E}_{i-1})$ is transformed into the graph $\widehat{G}_i(V, E_i)$ such that $\widehat{\mathbf{d}}_{\widehat{G}_0} = \widehat{\mathbf{d}}_{\widehat{G}_{i-1}} = \widehat{\mathbf{d}}_{\widehat{G}_i} = \widehat{\mathbf{d}}$ and $\Delta\left(\widehat{G}_i, G\right) < \Delta\left(\widehat{G}_{i-1}, G\right)$. The transformation is made using *valid swap* operations defined as follows:

Definition 6 *Consider a graph $\widehat{G}_i(V, \widehat{E}_i)$. A* valid swap *operation is defined by four vertices i, j, k and l of $\widehat{G}_i(V, \widehat{E}_i)$ such that $(i, k) \in \widehat{E}_i$ and $(j, l) \in \widehat{E}_i$ and $(i, j) \notin \widehat{E}_i$ and $(k, l) \notin \widehat{E}_i$, or, $(i, l) \notin \widehat{E}_i$ and $(j, k) \notin \widehat{E}_i$. A valid swap operation transforms $\widehat{G}_i$ to $\widehat{G}_{i+1}$ by updating the edges as follows*

$$\begin{aligned} \widehat{E}_{i+1} &\leftarrow \widehat{E}_i \setminus \{(i,k),(j,l)\} \cup \{(i,j),(k,l)\}, \qquad \textit{or} \\ \widehat{E}_{i+1} &\leftarrow \widehat{E}_i \setminus \{(i,k),(j,l)\} \cup \{(i,l),(j,k)\}. \end{aligned}$$

A visual illustration of the swap operation is shown in Figure 2. It is clear that performing valid swaps on a graph leaves the degree sequence of the graph intact. The pseudocode for the Greedy_Swap algorithm is given in Algorithm 2. At each iteration of the algorithm, the swappable pair of edges e_1 and e_2 is picked to be swapped to edges e'_1 and e'_2. The selection among the possible valid swaps is made so that the pair that reduces the most the evaluation of $\Delta\left(\right)$ function is picked.The Greedy_Swap algorithm halts when there are no more valid swaps that can decrease the $\Delta\left(\right)$ function.

Algorithm 3 gives the pseudocode of the whole process of solving the Graph Construction problem when the degree sequence $\widehat{\mathbf{d}}$ is realizable. The first step involves a call to the ConstructGraph algorithm, which we have described in Section 6.1, Algorithm 1. The ConstructGraph algorithm will return a graph $\widehat{G}_0$ with degree distribution $\widehat{\mathbf{d}}$. The Greedy_Swap algorithm is then invoked with input the constructed graph $\widehat{G}_0$. The final output of the process is a k-degree

Algorithm 2 The `Greedy_Swap` algorithm.

```
    Input: An initial graph $\widehat{G}_0(V,\widehat{E}_0)$ and the input graph $G(V,E)$.
    Output: Graph $\widehat{G}(V,\widehat{E})$ with the same degree sequence as $\widehat{G}_0$, such that $\Delta\left(\widehat{G},G\right)$ is minimized.
1: $\widehat{G}(V,\widehat{E}) \leftarrow \widehat{G}_0(V,\widehat{E}_0)$
2: $(c,(e_1,e_2,e'_1,e'_2)) =$ Find_Best_Swap($\widehat{G}$)
3: while $c > 0$ do
4:     $\widehat{E} = \widehat{E} \setminus \{e_1,e_2\} \cup \{e'_1,e'_2\}$
5:     $(c,(e_1,e_2,e'_1,e'_2)) =$ Find_Best_Swap
6: return $\widehat{G}$
```

Algorithm 3 An overall algorithm for solving the Graph Construction problem; the realizable case.

```
    Input: A realizable degree sequence $\widehat{\mathbf{d}}$ of length $n$.
    Output: A graph $\widehat{G}(V,E')$ with degree sequence $\widehat{\mathbf{d}}$ and $E \cap E' \approx E$.
1: $\widehat{G}_0 =$ ConstructGraph($\widehat{\mathbf{d}}$)
2: $\widehat{G} =$ Greedy_Swap($\widehat{G}_0$)
```

anonymous graph that has degree sequence $\widehat{\mathbf{d}}$ and large overlap in its set of edges with the original graph.

A naive implementation of the algorithm would require time $O(I|\widehat{E}_0|^2)$, where I is the number of iterations of the greedy step and $|\widehat{E}_0|$ the number of edges in the input graph graph. Given that $|\widehat{E}_0| = O(n^2)$, the running time of the `Greedy_Swap` algorithm could be $O(n^4)$, which is daunting for large graphs. However, we employ a simple sampling procedure that considerably improves the running time. Instead of doing the greedy search over the set of all possible edges, we uniformly at random pick a subset of size $O(log|\widehat{E}_0|) = O(\log n)$ of the edges and run the algorithm on those. This reduces the running time of the greedy algorithm to $O(I\log^2 n)$, which makes it efficient even for very large graphs. As we show in our experimental evaluation, the `Greedy_Swap` algorithm performs very well in practice, even in cases where it starts with graph $\widehat{G}_0$ that shares small number of edges with G.

6.2.1 The `Probing` Scheme

In the discussion above we have assumed that the degree sequence input in the `ConstructGraph` algorithm is realizable. However, it might well be the case that the `ConstructGraph` algorithm outputs a “No”, i.e., there does not exist a graph with the required degree sequence. In this case we invoke a `Probing` scheme described below. The `Probing` scheme is a randomized iterative process that tries to slightly change the degree sequence $\widehat{\mathbf{d}}$. The pseudocode of the `Probing` scheme is shown in Algorithm 4.

Algorithm 4 The `Probing` scheme.

```
    Input: Input graph $G(V,E)$ with degree distribution $\mathbf{d}$ and integer $k$.
    Output: Graph $\widehat{G}(V,\widehat{E})$ with $k$-anonymous degree sequence $\widehat{\mathbf{d}}$.
1:  $\widehat{\mathbf{d}} = \texttt{DP}(\mathbf{d})$
2:  $\left(\text{realizable}, \widehat{G}\right) = \texttt{ConstructGraph}(\widehat{\mathbf{d}})$
3:  while realizable = “No” do
4:      $\mathbf{d} = \mathbf{d} + \text{random\_noise}$
5:      $\widehat{\mathbf{d}} = \texttt{DP}(\mathbf{d})$
6:      $\left(\text{realizable}, \widehat{G}\right) = \texttt{ConstructGraph}(\widehat{\mathbf{d}})$
7:  Return $\widehat{G}$
```

For input graph $G(V,E)$ and integer k, the `Probing` scheme first constructs the k-anonymous sequence $\widehat{\mathbf{d}}$ by invoking the `DP` algorithm. If the subsequent call to the `ConstructGraph` algorithm returns a graph $\widehat{G}$, then `Probing` outputs this graph and halts. If `ConstructGraph` returns “No”, then `Probing` slightly increases some of the entries in $\mathbf{d}$ via the addition of uniform noise - the specifics of the noise-addition strategy is further discussed in the next paragraph. The new noisy version of $\mathbf{d}$ is then fed as input to the `DP` algorithm again. A new version of the $\widehat{\mathbf{d}}$ is thus constructed and input to the `ConstructGraph` algorithm to be checked. The process of noise addition and checking is repeated until a graph is output by `ConstructGraph`. Note that this process will always terminate because in the worst case, the noisy version of $\mathbf{d}$ will contain all entries equal to $n-1$, and there exists a complete graph that satisfies this sequence and is k-degree anonymous.

Since the `Probing` procedure will always terminate, the key question is how many times the **while** loop is executed. This depends, to a large extent, on the noise addition strategy. In our implementation, we examine the nodes in increasing order of their degrees, and slightly increase the degree of a single node in each iteration. This strategy is suggested by the degree sequences of the input graphs. In most of these graphs there is a small number of nodes with very high degrees. However, rarely any two of these high-degree nodes share exactly the same degree. In fact, we often observe big differences among them. On the contrary, in most graphs there is a large number of nodes with the same small degrees (close to 1). Given such a graph, the `DP` algorithm will be forced to increase the degrees of some of the large-degree nodes a lot, while leaving the degrees of small-degree nodes untouched. In the anonymized sequence thus constructed, a small number of high-degree nodes will need a large number of nodes to connect their newly added edges. However, since the degrees of small-degree nodes does not changed in the anonymized sequence, the demand of edge end-points imposed by the high-degree nodes cannot be facilitated. Therefore, by slightly increasing the degrees of small-degree nodes in $\mathbf{d}$ we force the `DP` algorithm to assign them higher degrees in the anonymized sequence $\widehat{\mathbf{d}}$. In that way, there are more additional free edges end-points to connect with the anonymized high-degree nodes.

From our experiments on a large spectrum of synthetic and real-world data, we observe that, in most cases, the extra edge-additions incurred by the `Probing` procedure are negligible. That is, the degree sequences produced by the `DP` are almost realizable, and more importantly, realizable with respect to the input graph G. Therefore, the `Probing` is rarely invoked, and even if it is invoked, only a very small number of repetitions are needed. We further discuss this in the experimental section of this chapter.

6.3 Restriction to edge additions

In this section we give yet another graph-construction algorithm for the case where only edge additions are allowed to the input graph. In this case, not only do we need to construct a graph $\widehat{G}$ with a given degree sequence $\widehat{\mathbf{d}}$, but we also require that $E \subseteq \widehat{E}$. We capture these two requirements in the following definition of *realizability of* $\widehat{\mathbf{d}}$ *subject to graph* G.

Definition 7 *Given input graph $G(V,E)$, we say that degree sequence $\widehat{\mathbf{d}}$ is realizable subject to G, if and only if there exists a simple graph $\widehat{G}(V,\widehat{E})$ whose nodes have precisely the degrees suggested by $\widehat{\mathbf{d}}$ and $E \subseteq \widehat{E}$.*

Given the above definition we have the following alternation of Lemma 2.

Lemma 3 *Consider degree sequence $\widehat{\mathbf{d}}$ and graph $G(V,E)$ with degree sequence $\mathbf{d}$. Let vector $\mathbf{a} = \widehat{\mathbf{d}} - \mathbf{d}$ such that $\sum_i \mathbf{a}(i)$ is even. If $\widehat{\mathbf{d}}$ is realizable subject to graph G then*

$$\begin{aligned}\sum_{i \in V_\ell} \mathbf{a}(i) \quad \leq \quad & \sum_{i \in V_\ell} \left(\ell - 1 - \mathbf{d}^\ell(i)\right) \\ & + \sum_{i \in V - V_\ell} \min\{\ell - \mathbf{d}^\ell(i), \mathbf{a}(i)\}, \qquad (9)\end{aligned}$$

where $\mathbf{d}^\ell(i)$ is the degree of node i in the input graph G when counting only edges in G that connect node i to one of the nodes in V_ℓ. Here V_ℓ is an ordered set of ℓ nodes with the ℓ largest $\mathbf{a}(i)$ values, sorted in decreasing order. In other words, for every pair of nodes (u,v) where $u \in V_\ell$ and $v \in V \setminus V_\ell$, it holds that $\mathbf{a}(u) \geq \mathbf{a}(v)$ and $|V_\ell| = \ell$.

Although the proof of the lemma is omitted due to space constraints, one can see the similarity between Inequalities (8) and (9); if G is a graph with no edges between its nodes, then $\mathbf{a}$ is the same as $\widehat{\mathbf{d}}$, $\mathbf{d}^\ell(i)$ is zero, and the two inequalities become identical.

Lemma 3 states that Inequality (9) is just a *necessary* condition for realizability subject to the input graph G. Thus, if Inequality (9) does not hold, we can conclude that for input graph $G(V,E)$, there does not exist a graph $\widehat{G}(V,\widehat{E})$ with degree sequence $\widehat{\mathbf{d}}$ such that $E \subseteq \widehat{E}$.

Although Lemma 3 gives only a necessary condition for realizability subject to an input graph G, we still want to devise an algorithm for constructing a degree-anonymous graph $\widehat{G}$, a supergraph of G, if such a graph exists. We call this algorithm the `Supergraph`, which is an extension of the `ConstructGraph` algorithm (We omit the pseudocode of `Supergraph` due to space limits).

The inputs to the `Supergraph` are the original graph G and the desired k-anonymous degree distribution $\widehat{\mathbf{d}}$. The algorithm operates on the sequence of *additional degrees* $\mathbf{a} = \widehat{\mathbf{d}} - \mathbf{d}_G$ in a manner similar to the one the `ConstructGraph` algorithm operates on the degrees $\mathbf{d}$. However, since $\widehat{G}$ is drawn on top of the original graph G, we have the additional constraint that edges already in G cannot be drawn again.

The `Supergraph` first checks whether Inequality (9) is satisfied and returns "No" if it does not. Otherwise it proceeds iteratively and in each step it maintains the residual additional degrees $\mathbf{a}$ of the vertices. In each iteration it picks an arbitrary vertex v and adds edges from v to $\mathbf{a}(v)$ vertices of *highest* residual additional degree, ignoring nodes v' that are already connected to v in G. For every new edge (v, v'), $\mathbf{a}(v')$ is decreased by 1. If the algorithm terminates and outputs a graph, then this graph has degree sequence $\widehat{\mathbf{d}}$ and is a supergraph of the original graph. If the algorithm does not terminate, then it outputs "Unknown", meaning that there might exist a graph, but the algorithm is unable to find it. Though `Supergraph` is similar to `ConstructGraph`, it *is not* an oracle. That is, if the algorithm does not return a graph $\widehat{G}$ supergraph of G, it does not necessarily mean that such a graph does not exist.

For degree sequences of length n and $a_{\max} = \max_i \mathbf{a}(i)$ the running time of the `Supergraph` algorithm is $O(na_{\max})$, using the same data-structures as those described in Section 6.1.

7 Experiments

In this section we evaluate the performance of the proposed graph-anonymization algorithms.

7.1 Datasets

We use both synthetic and real-world datasets. For the experiments with synthetic datasets, we generate *random*, *small-world* and *scale-free* graphs.

Random graphs: Random graphs are graphs with nodes randomly connected to each other with probability p. Given the number of nodes n and the parameter p, a random graph is generated by creating an edge between each pair of nodes u and v with probability p. We use $\mathcal{G}_R$ to denote the family of graphs generated by this data-generation model and G_R to denote a member of the family.

Small-world graphs: A small-world graph is a type of graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of hops. This kind of graphs have large *clustering* *coefficient* (CC) that is significantly higher than expected by random chance, and small *average path length* (APL) that is close to that of an equivalent random graph. The average path length is defined as the average length of the shortest path between all pairs of reachable nodes. The clustering coefficient is defined as the average fraction of pairs of neighbors of a node that are also connected to each other. These two indices, along with the degree distribution, are considered as standard measures in graph-analysis studies. We generate small-world graphs using the model proposed in [20]. We denote by $\mathcal{G}_W$ the family of graphs generated by this model and G_W the members of this family. The data-generation process is controlled by a parameter α that determines the extent to which the graph exhibits community structure. Values of α in the range $[5, 7]$ generate small-world graphs. We have additionally conducted experiments with small-world graphs generated using the alternative model proposed in [21]. However, since the results we obtained are very similar to the results obtained by using graphs in $\mathcal{G}_W$, we do not report them here due to space limitations.

Scale-free graphs: The scale-free graphs correspond to graphs with power-law degree distribution. In a power-law graph the probability that a node has degree d is proportional to $d^{-\gamma}$. The power-law distribution is determined by the exponent γ. The value of γ may vary, taking values between 2 and 3 for most real networks. We use the model proposed by Barabási and Albert [3] to generate scale-free graphs. The graph-generation process proceeds by inserting nodes sequentially. Each new node is initially connected to ℓ already existing nodes with probability proportional to their degree. We use $\mathcal{G}_{BS}$ to denote the family of graphs generated by this model and G_{BS} to denote members of the family.

For the real-world data, we use the **enron**, the **powergrid** and the **co-authors** graphs.

Enron graph: The Enron email graph (available at `http://www.cs.cmu.edu/enron/`) is derived from a corpus of emails sent to and from managers at Enron Corporation. This data was originally made public by the Federal Energy Regulatory Commission. The dataset contains 151 users. An edge between two users is added if they have corresponded at least five times.

Powergrid graph: In this graph, the nodes represent generators, transformers and substations in a powergrid network; the edges represent high-voltage transmission lines between them. The dataset is available at `http://www.cs.helsinki.fi/u/tsaparas/MACN2006/`.

Co-authors graph: The co-authors dataset consists of 7955 authors of papers in database and theory conferences and it is available at the collection of Computer Science Bibliographies at `http://liinwww.ira.uka.de/bibliography/`. The co-authors graph is constructed by creating undirected edges between authors that have co-authored paper.

Table 1 summarizes the properties of the graphs we used for our experiments. All the graphs are simple, unweighted and undirected.

	#Nodes	#Edges	APL	CC
$\mathcal{G}_W$ ($\alpha = 6$)	1000	5000	9.15	0.77
$\mathcal{G}_R$	1000	5000	3.27	0.01
$\mathcal{G}_{BS}$ ($\gamma = 3$)	1000	2995	3.57	0.02
enron	151	502	3.32	0.46
powergrid	4941	6594	9.12	0.10
co-authors	7955	10055	6.00	0.64

Table 1: Structural properties of the graphs used for the experiments.

7.2 Evaluating Graph Construction algorithms

In this section we evaluate the performance of `Greedy_Swap`, `Greedy_Swap_Additions` and `Supergraph` algorithms. Since the k-degree anonymity is guaranteed by construction, we only need to look at the structural similarity between the input and output graphs. We use a set of evaluation measures listed below and we report our results for different synthetic and real-world graphs.

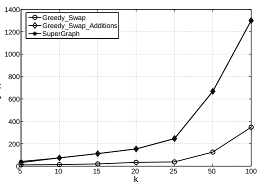
Anonymization cost $L_1(\mathbf{d}_A - \mathbf{d})$: This is the L_1 norm of the vector of differences between the k-anonymous degree sequence obtained using algorithm $Algo \in \{$ `Greedy_Swap`, `Greedy_Swap_Additions`, `Supergraph`$\}$ and the degree sequence of the original graph. The smaller the value of $L_1(\mathbf{d}_A - \mathbf{d})$ the better the qualitative performance of the algorithm. Figures 3(a), 4(a), 5(a) and 6(a) summarize the anonymization cost of the different algorithms as a function of $k = \{5, 10, 15, 20, 25, 50, 100\}$ for synthetic datasets $G_W \in \mathcal{G}_W$ with $\alpha = 6$, $G_{BS} \in \mathcal{G}_{BS}$, and **powergrid** and **co-authors** data. From the plots, we can observe that `Greedy_Swap` always has the lowest anonymization cost. This is because `Greedy_Swap` allows the degrees of nodes to either increase or decrease. On the other hand, `Greedy_Swap_Additions` and `Supergraph` have relatively higher cost due to their increase-only nature.

If all the anonymized degree sequences are realizable, the optimal cost for both `Greedy_Swap_Additions` and `Supergraph` should be the same. However, we note that in the case of $\mathcal{G}_{BS}$ graphs, $L_1(\mathbf{d}_{\texttt{Supergraph}} - \mathbf{d})$ cost is relatively high for all values of k (see Figure 4(a)). This is due to two reasons: 1) The degrees of graphs in $\mathcal{G}_{BS}$ have a power-law distribution. This causes large differences among the degrees of high-degree nodes, meaning that the degrees of high-degree nodes have to be changed significantly in order to meet the degree-anonymous requirement. 2) The `Supergraph` algorithm constructs the degree-anonymous graph by extending the input graph, and it is the only of our proposed algorithms that tries to comply with all the edge constraints imposed by the input graph. Therefore, it can potentially add more noise (in the `Probing` phase) than other algorithms that build the graph from scratch.

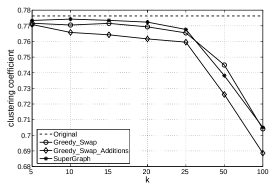
Clustering Coefficient (CC): We compare the clustering coefficients of the anonymized graphs with the clustering coefficients of the original graphs. Figures 3(b), 4(b), 5(b) and 6(b) summarize our findings. In all plots, there is a constant line appearing, this corresponds to the value of the clustering coefficient of the *original* graph, which is unaffected by the value of k. Note that all the plots show that the values of the clustering coefficient, though different in the degree-anonymous graphs, they never deviate too much from their original values; the largest difference in the CC values from the original values is observed for the **co-author** dataset, where the difference is 0.24 for the degree-anonymous graph produced by the `Greedy_Swap_Additions` algorithm when $k = 100$. But even in this case, the other two algorithms output graphs with CC almost equal to that of the original graph. Note that there is no clear trend on how the CC changes when the graph becomes degree anonymous. Both increments and decrements are observed, however the changes are generally negligible.

Average Path Length (APL): In Figures 3(c), 4(c), 5(c) and 6(c) we report the values of the average path length of the degree-anonymous graphs and the original graphs. As expected, the anonymization process of `Greedy_Swap_Additions` and `Supergraph` decrease the average path length of the output graph since only new connections are added. The `Greedy_Swap`, on the other hand, can either increase or decrease the average path length because it simultaneously adds and deletes the edges.

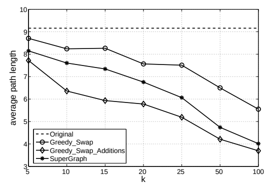
Very similar results have been obtained for other datasets generated using the random-graph model as well as the **enron** dataset. However, we omit the corresponding plots due to space constraints.



(a) $L_1(\mathbf{d}_A - \mathbf{d})$ as a function of k

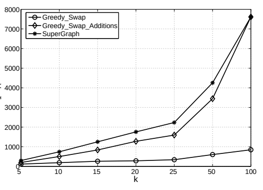


(b) CC as a function of k

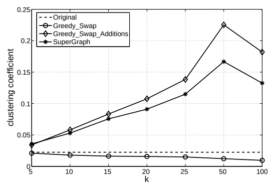


(c) APL as a function of k

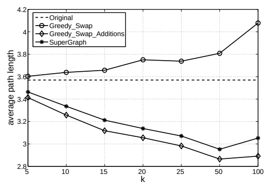
Figure 3: Synthetic datasets: small-world graphs $G_W \in \mathcal{G}_W$, with $\alpha = 6$.



(a) $L_1(\mathbf{d}_A - \mathbf{d})$ as a function of k



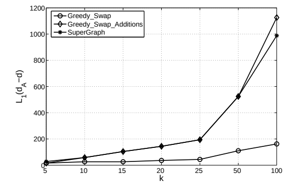
(b) CC as a function of k



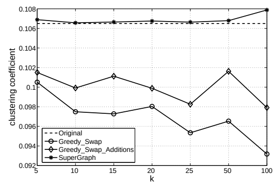
(c) APL as a function of k

Figure 4: Synthetic datasets: scale-free graphs $G_{BS} \in \mathcal{G}_{BS}$.

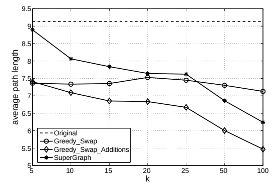
Structural Difference: Recall that the structural difference between two graphs $G(V, E)$ and $\widehat{G}(V, \widehat{E})$, $\Delta\left(\widehat{G}, G\right)$, is the symmetric difference of their



(a) $L_1(\mathbf{d}_A - \mathbf{d})$ as a function of k

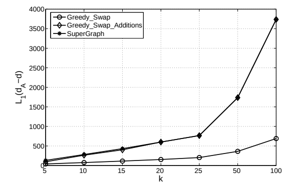


(b) CC as a function of k

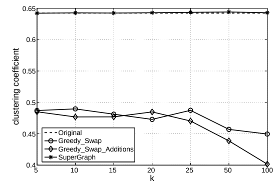


(c) APL as a function of k

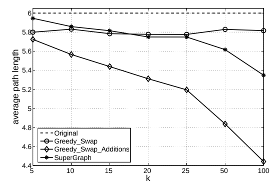
Figure 5: Real datasets datasets: **powergrid** data.



(a) $L_1(\mathbf{d}_A - \mathbf{d})$ as a function of k



(b) CC as a function of k



(c) APL as a function of k

Figure 6: Real datasets datasets: **co-authors** data.

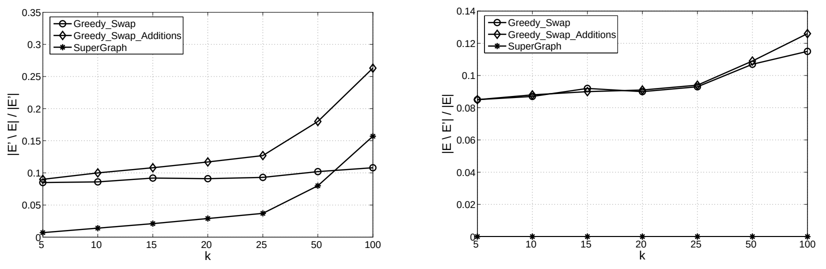
edge sets, *i.e.*, $\Delta\left(\widehat{G}, G\right) := \left|\widehat{E} \setminus E\right| + \left|E \setminus \widehat{E}\right|$. Table 2 summarizes the values of $\Delta\left(\widehat{G}, G\right)$ obtained by different algorithms for the **co-authors** data. It is interesting to observe that `Supergraph` consistently has a low value. This is due to the reason that `Supergraph` constructs the anonymized graph by only extending the input graph, and therefore, $\left|E \setminus \widehat{E}\right|$ is always 0. On the other hand, both `Greedy_Swap` and `Greedy_Swap_Additions` construct a anonymized graph from scratch and then heuristically add and delete edges to make it more similar to the original graph. This procedure leads to higher symmetric differences of their edge sets.

However, readers should not be deluded by the straight numbers in the table and reach a conclusion that `Greedy_Swap` and `Greedy_Swap_Additions` are significantly inferior to `Supergraph`. In fact, we can show that the number of edges that are added and deleted by these two greedy algorithms only account for a very small portion of the overall edge sets. To illustrate this, we decompose the structural difference between G and $\widehat{G}$ into two normalized components: $\frac{\left|\widehat{E} \setminus E\right|}{\left|\widehat{E}\right|}$ and $\frac{\left|E \setminus \widehat{E}\right|}{|E|}$. The former gives the percentage of the edges in the anonymized graph that are newly added, and the latter calculates the percentage of the edges in the original graph that are deleted. The lower these two values, the better the structure of the original graph is preserved. Figure 7(a) and 7(b) show the

	`Greedy_Swap`	`Greedy_Swap_Additions`	`Supergraph`
$k = 5$	1714	1764	66
$k = 10$	1742	1904	141
$k = 15$	1846	2009	216
$k = 20$	1815	2126	300
$k = 25$	1868	2269	384
$k = 50$	2096	3068	868
$k = 100$	2232	4402	1868

Table 2: Structural differences between G and $\widehat{G}$, obtained by different algorithms for the **co-authors** data.

values of $\frac{|\widehat{E} \setminus E|}{|\widehat{E}|}$ and $\frac{|E \setminus \widehat{E}|}{|E|}$ obtained by different algorithm as a function of $k = \{5, 10, 15, 20, 25, 50, 100\}$ for the **co-authors** data. We can easily observe that both `Greedy_Swap` and `Greedy_Swap_Additions` produce very small values of $\frac{|\widehat{E} \setminus E|}{|\widehat{E}|}$ and $\frac{|E \setminus \widehat{E}|}{|E|}$. In particular, `Greedy_Swap` achieves 0.12 for both components even when k is 100, which is better than `Supergraph` that has a value of 0.16 for $\frac{|\widehat{E} \setminus E|}{|\widehat{E}|}$.



(a) $\frac{|\widehat{E} \setminus E|}{|\widehat{E}|}$ as a function of k (b) $\frac{|E \setminus \widehat{E}|}{|E|}$ as a function of k

Figure 7: Decomposed structural differences between G and $\widehat{G}$, obtained by different algorithms for the **co-authors** data.

7.2.1 Exploring the Scale-free Graphs

Previous work on the analysis of complex networks has shown that many of the real-world graphs are scale free, *i.e.*, their node degrees follow a power-law distribution. In this section we demonstrate that our anonymization framework does not destroy the power-law property of the original graph if k is not too big. That is, if the input graph has a power-law degree distribution, so does the degree-anonymous version of it.

	γ		
	`Greedy_Swap`	`Greedy_Swap_Additions`	`Supergraph`
original	**2.07**	**2.07**	**2.07**
$k = 10$	2.45	2.26	2.26
$k = 15$	2.33	2.13	2.13
$k = 20$	2.28	1.97	1.97
$k = 25$	2.25	1.83	1.83
$k = 50$	2.05	1.57	1.57
$k = 100$	1.92	1.22	1.22

Table 3: Real dataset: **co-authors** graph. Value of the exponent (γ) of the power-law distribution of the original and the k-degree anonymous graph obtained using `Greedy_Swap`, `Greedy_Swap_Additions` and `Supergraph` algorithms, for $k = 10, 15, 20, 25, 50, 100$.

In Table 3, we report the values of the estimated exponent (γ) of the power-law distribution of the original **co-authors** data and its degree-anonymous counterpart. We can observe that the new γ values obtained by all three algorithms exhibit high degree of similarity to the original one for $k < 15$. When k gets larger, `Greedy_Swap` still preserves the γ value very well. This result is due to the fact that, the degree-sequence anonymization of `Greedy_Swap` minimally changes the degree sequence of a graph. For significant large values of k (*e.g.*, $k = 100$), a great amount of the nodes in the anonymized graph will have the same degree, and the power-law distribution will change. We claim that this is a natural result for any degree-anonymization algorithm.

8 Conclusions

The degree of a node in a graph, among other structural characteristics, can to a large extent distinguish the node from other nodes. In this chapter, we focused on a specific graph-anonymity notion that prevents the re-identification of individuals by an attacker with certain prior knowledge of the degrees. We formally defined the Graph Anonymization problem that, given an input graph asks for the graph modifications (in terms of additions and deletions of edges) that allow for the transformation of the input to a degree-anonymous graph; *i.e.*, a graph in which every node shares the same degree with $k - 1$ other nodes. We showed that this problem can be decomposed into two subproblems and proposed simple and efficient algorithms for solving them. We also presented experiments on synthetic and real-world graph data and demonstrated the utility of the degree-anonymous graphs as well as the efficiency of our methods.

From the material presented in this chapter as well as in the related work (presented in Section 3) it becomes apparent that privacy-preserving data analysis on graph data raises many more challenges when compared to the challenges that arise from anonymization and perturbation techniques in tabular data. In the latter case, each tuple can be viewed as an independent sample from some distribution. However, in a graph, all the nodes and edges are correlated; a single change of an edge and/or a node can spread across the whole network. Moreover, in graphs it is difficult to model the capability of an attacker. In principle, any topological structure of the graph can be potentially used to derive private information. Another challenge is related to the right definition of the utility of an anonymized graph. For example, we measured the utility using the degree anonymization and the structural cost. However, other measures that associate the structural properties of the original and the anonymized graph can also be used. Further, these measures can be also tailored to a particular graph-analysis task. Overall, there are many directions that need to be explored before the research community agrees upon a unifying, theoretically- and practically-sound, model for privacy in social networks.

References

[1] Aggarwal, C. C., and Yu, P. S. *Privacy-Preserving Data Mining: Models and Algorithms*, vol. 34 of *Advances in Database Systems.* Springer, 2008.

[2] Backstrom, L., Dwork, C., and Kleinberg, J. M. Wherefore art thou R3579X?: Anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)* (Alberta, Canada, May 2007), pp. 181–190.

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

[4] Bayardo, R. J., and Agrawal, R. Data privacy through optimal k-anonymization. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)* (Tokyo, Japan, April 2005), pp. 217–228.

[5] Boyd, S., and Vandenberghe, L. *Convex Optimization.* Cambridge UP, 2004.

[6] Cormen, T., Leiserson, C., and Rivest, R. *Introduction to Algorithms.* MIT Press, 1990.

[7] Diakonikolas, I., and Yannakakis, M. Succinct approximate convex pareto curves. In *SODA* (2008), pp. 74–83.

[8] Erdös, P., and Gallai, T. Graphs with prescribed degrees of vertices. *Mat. Lapok* (1960).

[9] Getoor, L., and Diehl, C. P. Link mining: a survey. *ACM SIGKDD Explorations Newsletter 7*, 2 (2005), 3–12.

[10] Hakimi, S. L. On realizability of a set of integers as degrees of the vertices of a linear graph. *Journal of the Society for Industrial and Applied Mathematics 10*, 3 (1962), 496–506.

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

[12] Hay, M., Miklau, G., Jensen, D., Weis, P., and Srivastava, S. Anonymizing social networks. Technical report, University of Massachusetts Amherst, 2007.

[13] Korolova, A., Motwani, R., Nabar, S. U., and 0002, Y. X. Link privacy in social networks. In *CIKM* (2008), pp. 289–298.

[14] Lee, Y.-S. Graphical demonstration of an optimality property of the median. *The American Statistician 49*, 4 (November 1995), 369–372.

[15] Liu, K., and Terzi, E. Towards identity anonymization on graphs. In *SIGMOD Conference* (2008), pp. 93–106.

[16] Machanavajjhala, A., Gehrke, J., Kifer, D., and Venkitasubramaniam, M. l-diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)* (Atlanta, GA, April 2006), p. 24.

[17] Meyerson, A., and 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 (PODS'04)* (Paris, France, 2004), pp. 223–228.

[18] Pei, J., and Zhou, B. Preserving privacy in social networks against neighborhood attacks. In *Proceedings of the 24th International Conference on Data Engineering (ICDE'08)* (Cancun, Mexico, April 2008).

[19] Samarati, P., and Sweeney, L. Generalizing data to provide anonymity when disclosing information. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)* (Seattle, WA, 1998), p. 188.

[20] Watts, D. J. Networks, dynamics, and the small-world phenomenon. *American Journal of Sociology 105*, 2 (September 1999), 493–527.

[21] Watts, D. J., and Strogatz, S. H. Collective dynamics of small-world networks. *Nature 393*, 6684 (June 1998), 409–410.

[22] Ying, X., and Wu, X. Randomizing social networks: a spectrum preserving approach. In *Proceedings of SIAM International Conference on Data Mining (SDM'08)* (Atlanta, GA, April 2008).

[23] Zheleva, E., and Getoor, L. Preserving the privacy of sensitive relationships in graph data. In *Proceedings of the International Workshop on Privacy, Security, and Trust in KDD (PinKDD'07)* (San Jose, CA, August 2007).

[24] Zhou, B., Pei, J., and Luk, W.-S. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *ACM SIGKDD Explorations 10*, 2 (December 2008), 12–22.